

Angular Questions:

1. Angular ?

Angular is a typescript-based web application framework .It helps in creating reactive single page applications (SPA) and is completely based on the concept of components. Google owns Angular (2010), it makes sure that they release a major version 3 of Angular every six months.

Angular has nowadays become the most popular framework for developing mobile and desktop-based web applications / mobile applications by using ionic framework. It comes with a variety of features. Some of the key features of Angular are Modules , components, forms, directives, dependency injection, HTTP services,Routes, pipes, etc.

To work with Angular (any version), we will need to install a few applications on your computer:

- Setup a Node.js development environment with npm (node package manager)
- Setup the Angular CLI (command-line interface) -

```
npm instal -g @angular/cli ( -g = global )
```

- Setup an Editor/IDE

2. SPA (Single Page Applications) ?

Single-page applications are web applications that provide to the user with fast user experience.

It helps to load a portion of the current page dynamically instead of reloading the entire page from the server.

3. Angular CLI?

The [Angular CLI](#) is a command-line tool that allows us to create, develop, scaffold, and manage Angular applications from a command shell.

Examples :

ng new app-name

Ng serve

Ng build -(it will generate one dist folder which we can deploy)

Ng generate component component-name

Ng generate service service-name

Ng generate directive directive-name

Ng generate pipe pipe-name

Ng generate module module-name

4. angular and angular js?

Angular	Angular JS
The name Angular became popular after Angular 2.0.	This was the common name for the first version of Angular (Angular 1.0).
It is a TypeScript-based framework.	It is a JavaScript-based framework.
It provides the feature of dynamic loading of web pages.	It does not provide such dynamic loading of pages.
Angular uses the concept of components as its primary building entity rather than scopes and controllers.	The older version of Angular (Angular JS) uses scope and controller as its primary application developing entity.`

5. Main Building Blocks for angular ?

Modules

Components

Directives,

Pipes ,

Services ,

Dependency Injection ,

Templates,

Forms

HttpClient,

Routes

6. create angular application ?

1. Before starting with Angular, you need to [Install Angular](#). Before going further, install the following.

- [Visual Studio Code](#) (or any other editor of your choice)
- [NodeJs & NPM Package Manager](#)
- Install Angular CLI

2. Once installed we need to check the version of angular using **ng version**

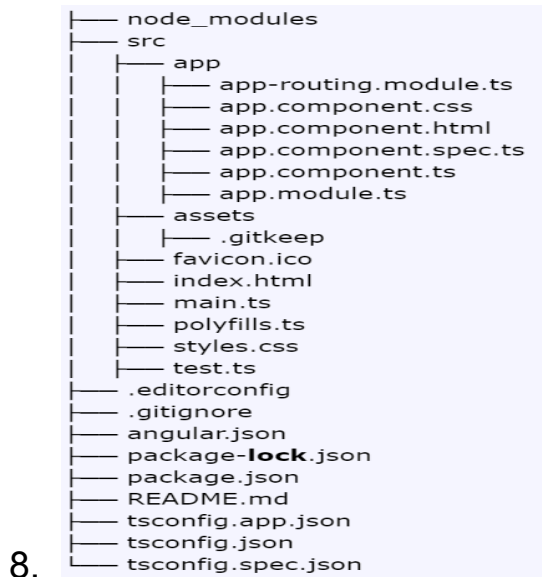
3. Create new angular application (**ng new** ApplicationName)

4. Would you like routing ? (Yes)

5. *Which stylesheet format would you like to use?* You have four options: **CSS, SCSS, SASS**, and **LESS**. **Angular has deprecated Stylus**. Please feel free to choose whichever works best for you. For this tutorial, we choose **CSS**.

6. Run the new angular application / Project using(**ng serve**)

7. Angular Project Folder Structure ?



1 . Node modules will contain all thirdparty libraries

- a. If you install any thirdparty library it will come into the node_modules folder
- b. Inorder to install thirdparty library
 - i. For example to install bootstrap
 - ii. `npm install bootstrap`
 - iii. Import this line `@import "~bootstrap/dist/css/bootstrap.min.css";` in style.scss

Package.json : package.json will register all the libraries which angular application has dependency

Differences between dependencies and devDependencies

Dependencies are packages that are required for the application to run correctly in a production environment.

Ex : bootstrap

devDependencies are packages that are only needed for development and testing purposes.

Ex : `@angular/cli`

Package-lock.json

The package-lock.json file locks down the version of each installed dependency and its sub-dependencies. This ensures that when you or someone else installs dependencies for the project, the exact same versions are installed every time

Editorconfig:

The .editorconfig file is a configuration file used to define and maintain consistent coding styles and formatting conventions across different editors and IDEs within a development team or project.

.gitignore:

Git ignore is used to ignore the files which we don't want to push

Angular.json

The angular.json file is a configuration file used by the Angular CLI (Command Line Interface) to manage various aspects of an Angular project's build process, including settings for compilation, bundling, and deployment.

Tsconfig.json

The tsconfig.json file is a configuration file used by the TypeScript compiler while compiling the code typescript code into JavaScript, as well as other aspects of the TypeScript build process.

Tslint.json

The tslint.json file is a configuration file used by TSLint, which is a static analysis tool for TypeScript code. TSLint helps enforce coding standards and best practices by analyzing TypeScript code for potential errors, stylistic issues, and violations of coding conventions.

polyfills.ts

The polyfills.ts file is a TypeScript file used in Angular projects to import and configure polyfills. Polyfills are pieces of code that provide modern functionality to older browsers that don't support certain features of JavaScript or web standards.

The root folder application contains two subfolders. `node_modules` and `src`. It also contains a few configuration files - [\(1\) #04 Angular files and folder structure | Getting Started with Angular | A Complete Angular Course - YouTube](#)

9. How angular application starts / load ?

[\(1\) #05 Bootstrapping Angular Application | Getting Started with Angular | A Complete Angular Course - YouTube](#)

10. Component in Angular ?

The Angular Component is the main building block of an Angular application.

The Angular Components are plain `JavaScript` classes defined using the **@Component Decorator**.

Every Decorator starts with @ symbol

@Component() decorator imports from @angular/core

The Component is responsible for providing the data to the View. Angular uses `data binding` to get the data from the Component to the View and vice versa.

Every Angular Application will have one root component which is AppComponent

The Components consist of three main building blocks

- Template
- Class
- MetaData

Template :

The Template defines the layout and content of the View. Without the template, there is nothing for Angular to render to the DOM(Document Object Model).

There are two ways you can specify the Template in Angular.

1. Defining the Template Inline

```
@Component({  
  selector: 'app-name',  
  template: `<h1>This is Header</h1>`,  
  
  styleUrls: ['./app.component.css']  
})
```

2. Provide an external Template

```
@Component({  
  selector: 'app-name',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})
```

Class:

The Class provides the data & logic to the View. It contains the [Typescript](#) code

Class Contains the Properties & Methods. The Properties of a class can be displayed in the view using [Data Binding](#).

The simple Angular Class

```
1  
2 export class AppComponent  
3 {  
4   title : string ="app"  
5 }  
6
```


Meta Data :

Metadata Provides additional information about the component of Angular. We use the `@Component` decorator to provide the Metadata to the Component.

When Angular sees a class with `@Component` decorator, it treats the class as a Component.

Important Component metadata properties :

- **Selector**

The Angular looks for the selector in the template and renders the component there.

- **Providers**

The Providers are the [Angular Services](#) that our component is going to use

- **Styles/styleUrls**

The CSS styles or style sheets that this component needs. Here we can use either an external stylesheet (using `styleUrls`) or inline styles (using `Styles`). The styles used here are specific to the component

- **template/templateUrl**

The HTML template that defines our View. It tells Angular how to render the Component's view. The templates can be inline (using a template) or we can use an external template (using a templateUrl). The Component can have only one template. You can either use an inline template or an external template, and not both.

Creating a Component in Angular 8:

1 . To create a component in any angular application, follow the below steps:

- Get to the angular app via your terminal.
- Create a component using the following command:

ng g c <component_name> OR ng generate component <component_name>

2. Create the Component Manually

```
@Component({
})
export class HelloWorldComponent {
  title = 'Hello World';
}
```

Add metadata to @Component decorator

```
1
2 @Component({
3   selector: 'app-hello-world',
4   templateUrl: './hello-world.component.html',
5   styleUrls: ['./hello-world.component.css']
6 })
7 export class HelloWorldComponent {
8   title = 'Hello World';
9 }
10
```

Now Register the Component in Angular Root Module.

Module :

We have created the Angular Component. The next step is to register it with the [Angular Module](#). Our application already has one Module, i.e., `app.module.ts`, which is also a root module.

Every Angular Application will have one root module which is **AppModule(app.module.ts)**

The root module is the Module Angular loads when the App starts.

We use the `@NgModule` class decorator to define an [Angular Module](#) and provide metadata about the Modules. There are four important properties of `@NgModule` metadata. They are declaration, imports, providers & bootstrap.

We include the components, [pipes](#), and [directives](#) part of this Module in the **declaration** array.

We add all the other [Angular Modules](#) in the **imports** array.

Include all the [angular services](#) part of this Module in the **providers** array.

The Component which we will give in the **bootstrap** array will load first.

```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { HelloWorldComponent } from './hello-world.component';
7
8 @NgModule({
9   declarations: [
10     AppComponent, HelloWorldComponent
11   ],
12   imports: [
13     BrowserModule,
14     AppRoutingModule
15   ],
16   providers: [],
17   bootstrap: [AppComponent]
18 })
19 export class AppModule { }
20
21
```

NOW Component is Ready

10. LifeCycle Hook Methods of Angular Components ?

Every Angular Application will go through the following lifecycle hook method in following order

<https://youtu.be/H0MvUtxK5Sg?si=VJ30wQMBE7VwzhiS>

- `ngOnChanges()`
- `ngOnInit()`
- `ngDoCheck()`
- `ngAfterContentInit()`
- `ngAfterContentChecked()`
- `ngAfterViewInit()`
- `ngAfterViewChecked()`
- `ngOnDestroy()`

`ngOnChanges(){ }` :

- This `ngOnChanges()` will be Called before `ngOnInit()` only if there are any Inputs are coming from its parent component -
- NOTE:
If your component is not receiving any input from the parent component , the framework will not call `ngOnChanges()`.

`ngOnInit() { }` :

- It will be called when the component is getting initialized for the first time.
- Called once, after the first `ngOnChanges()`. `ngOnInit()` is still called even when `ngOnChanges()` is not (which is the case when there are no template-bound inputs).

`ngDoCheck() { }` :

- Called immediately after `ngOnChanges()` on every change detection run,

`ngAfterViewInit() { }` :

- It will be called once the component view (html) Initialized

ngAfterViewChecked() {}:

- It will be called if there any changes in html like updating the elements and changing the styles

ngAfterContentInit() {}:

- It will be called when projected content once initialized (projecting content means passing the html code form parent to child component)

ngAfterContentChecked() {}:

- It will be called when projected content gets updated

7

ngOnDestroy() {}:

- It will be called whenever we leaving from the component

To use these lifecycle hook methods we need to implements from Interfaces which are OnChanges , OnInit, DoCheck, AfterViewInit , AfterViewChecked, AfterContentInit, AfterContentChecked , OnDestroy

Example :

```
export class AppComponent implements OnDestroy ,
OnInit {
  ngOnDestroy() {
    console.log("Goodbye World!");
  }
  ngOnInit() {
    console.log("Component Initialized")
  }
}
```

11. View encapsulation?

<https://www.youtube.com/watch?si=h1SipexnbLlh2v14&v=7mRkFdLkFPw&feature=youtu.be>

r

12. Pass the data between two components?

Angular, like any of the other major frontend frameworks, uses components as their central building blocks. The advantages of separation into logical units, especially with large applications, are great: code becomes more manageable, it improves readability, reusability, and maintainability.

How to pass data from one component into another one depends on their relationship with each other. The concept of *parent-child-sibling relationship*, which is essential when working with HTML, CSS and the DOM, applies also to components in Angular, or any other frontend framework.

- Parent to Child
- Child to Parent
- Child to Child
- Un-related components

Parent to Child Communication:

in the parent component:

1. Create a variable that stores the data.
`currentMsgToChild1 = '';`
2. Use double-binding with [(ngModel)] for the newly created variable.
`<input type="text" [(ngModel)]="currentMsgToChild1">`
3. Add the child template and use property binding to assign the variable to the property.

```
<app-child1  
[msgFromParent1]='currentMsgToChild1'></app-child1>
```

in the child component:

1. Declare a variable with the @Input decorator.
`@Input() msgFromParent1: any[];`
2. Render the data in the template.
`<p>{{ msgFromParent1 }}</p>`

Passing Data from Child Component to Parent Component:

We can pass the data from child to parent in two ways one using @ViewChild() Decorator and another one using @Output() decorator

- ***Passing data from child to parent using @ViewChild() Decorator***

@ViewChild() is a property decorator - It gives the parent access to all the attributes and functions of the child component.

Steps to pass data from child to parent using @ViewChild()

in the parent component:

1. Import `ViewChild()` and `ngAfterViewInit()` and implement the lifecycle hook.
`import { ViewChild, AfterViewInit } from '@angular/core';
export class AppComponent implements AfterViewInit {...}
ngAfterViewInit() {}`
2. Import the child component.
`import { Child1Component } from './child1/child1.component';`
3. Use the @ViewChild() directive.
`@ViewChild(Child1Component, {static: false}) child1:
Child1Component;`
4. Declare a variable that holds the data.
`msgFromChild1: any;`

5. In `ngAfterViewInit()` call the variable that holds the data.
`ngAfterViewInit() {this.msgFromChild1 =
this.child1.msgFromChild1; }`

in the child component:

1. Create two variables, one that holds the data via double-binding and one that is an array.
`currentMsgToParent="";
msgFromChild1 = [];`
2. Create an input with double-binding for the newly created variable `currentMsgToParent`.
`<input type="text" [(ngModel)]="currentMsgToParent">`
3. Use event binding to fire the function.
`<button (click)="msgToParent()">send</button>`
4. Create the function that pushes the value of the variable into the array.
`msgToParent() {
this.msgFromChild1.push(this.currentMsgToParent); }`

● ***Passing data from child to parent using @Output() Decorator***

In child component:

1. Declare a *variable* that is assigned to the *@Output decorator* which is set to a new `EventEmitter()`.
`@Output() callParent = new EventEmitter();`
2. Create a variable that holds the data.
`currentMsgToParent = 'cry, cry...';`
3. Create a *function* that calls *emit* on the *event* with the *data* that we want to send.
`msgToParent()
{this.callParent.emit(this.currentMsgToParent);}`
4. In the template create a button with a click event that calls the function. `<button
(click)="msgToParent()">Cry</button>`

In the parent component:

1. Create a variable to hold the data.
`msg: string;`
2. Create a function to receive the data, use \$event as a parameter, and set it equal to the data variable.
`getMsgFromBaby($event) {this.msg = $event;}`
3. In the template use property binding to run the function whenever the event occurs.

```
<app-baby  
(callParent)="getMsgFromBaby($event)"></app-baby>
```

Passing Data from Two unrelated components in angular:

We use **BehaviourSubject** to share the data with multiple components using a shared service.

For example, if we want to notify other components of the data change. Then we have to follow these steps.

In service.ts file

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

//1. Import BehaviorSubject from rxjs module

```
import { BehaviorSubject } from 'rxjs/BehaviorSubject'
```

```
@Injectable({  
  providedIn: 'root'  
})  
export class DataService {
```

//2. Create the data which we want to share with all the components

```
  private msg= new BehaviorSubject('');
```

//3. Now we want to send this message or data, so we create an observable

```
getMsg= this.msg.asObservable();
```

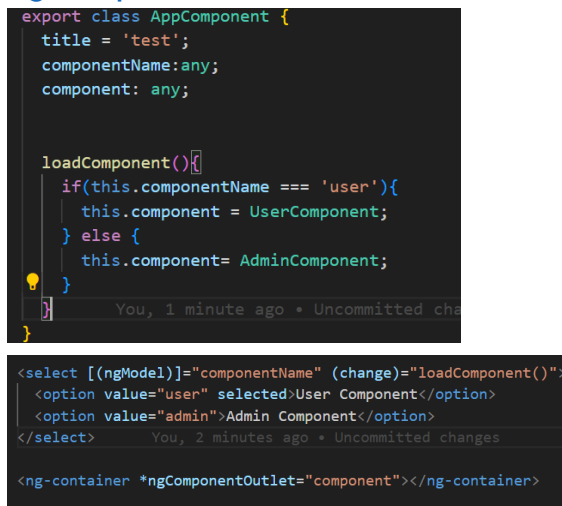
```
constructor() { }
```

```
//4. Create a method that updates the data (Behaviour Subject)  
} updateMsg(){  
  this.msg.next('updated Message');  
}
```

13 . Load Components Dynamically ?

Component templates are not always fixed. An application might need to load new components at runtime

1. **NgComponentOutlet** is used to load the components dynamically



```
export class AppComponent {  
  title = 'test';  
  componentName:any;  
  component: any;  
  
  loadComponent(){  
    if(this.componentName === 'user'){  
      this.component = UserComponent;  
    } else {  
      this.component= AdminComponent;  
    }  
  }  
}
```

```
<select [(ngModel)]="componentName" (change)="loadComponent()">  
  <option value="user" selected>User Component</option>  
  <option value="admin">Admin Component</option>  
</select>  
<ng-container *ngComponentOutlet="component"></ng-container>
```

2. *ngIf
3. ViewContainerRef - which is also from angular core library where we can use to load the components dynamically

14. a template?

Templates are written in HTML, and special syntax can be used within a template to build on many of Angular's features.

15. interpolation?

Interpolation is used to display the value of variable in HTML which related to particular component

interpolation uses the double curly braces **{{ and }}** as delimiters.

EXamples :

- `<p>{{title}}</p>`
- `<div></div>`

16. Property / Attribute Binding ?

Property binding in Angular is used to set the values for properties of HTML elements or directives.

For example - disabling the button , passing the data from parent to child

To bind to an element's property, enclose it in **square brackets, []**

Mainly we will use property binding for setting the paths/url programmatically . below is one example

Example :

```
itemImageUrl = '../assets/phone.svg';
```

```
<img alt="item" [src]="itemImageUrl">
```

18. Class and Style Binding ?

Use class and style bindings to add and remove CSS class names from an element's class attribute and to set styles dynamically.

Class Binding Syntaxes :

- `<tagname [ngClass]="{'class-name': user.status === 'active', 'class-name2':user.status === 'inactive'}"></tagname>`
- `<td [ngClass]="x > 0 ? 'blue' : 'green'">{{ x }}</td>`

Style Binding Syntaxes :

- `[ngStyle]="isActive ? {width: '50px', height: '20px'} : {}"`

19. Event Binding ?

Event binding listens and responds to user actions such as keystrokes, mouse movements, clicks, and touches.

Examples :

```
(click)="someFunction() "  
(dblclick)="someFunction() "  
(keyup)="someFunction() "  
(keypress)="someFunction() "  
(keydown)="someFunction() "  
(mouseup)="someFunction() "  
(mousedown)="someFunction() "  
(mouseenter)="someFunction() "  
(cut)="someFunction() "  
(copy)="someFunction() "  
(paste)="someFunction() "
```

20. Two Way Data Binding ?

Two way data binding means if we make changes model / ts in the component are propagated to the view and that any changes made in the view are immediately updated in the component data.

Simply - Model to View and View to Model

The Angular uses the `ngModel` directive to achieve the two-way binding on HTML Form elements

In-order use ngModel we have to import FormsModule in RootModule

```
<input type="text" name="value" [(ngModel)]="value">
```

21. Pipes ?

Angular Pipes are used to transform the data from one form to another form

Ex: strings , dates , currencies.

Examples : json , uppercase , lowercase , date , async , decimal.

Async Pipe is Important

- **DatePipe**: Formats a date value according to locale rules. - `{{ dateVal | date }}`
- **UpperCasePipe**: Transforms text to all upper case.
- **LowerCasePipe**: Transforms text to all lower case.
- **CurrencyPipe**: Transforms a number to a currency string. - `{{ amount | currency:'INR' }}`

AsyncPipe: Subscribe and unsubscribe to an asynchronous source such as an observable. -

- **JsonPipe**: Display a component object property to the screen as JSON for debugging.

How to do Chaining pipes / using multiple pipes. Below is example

```
{{ dateVal | date | uppercase }}
```

How to create our own / custom pipes

To create a custom pipe, first we need to create a pipe class. The pipe class must implement the PipeTransform interface. We also decorate it with @pipe decorator. Give a name to the pipe under name metadata of the @pipe decorator. Finally, we create the transform method, which transforms given value to the desired output. See the below example how to create and use it

Step1 :

```
import {Pipe, PipeTransform} from '@angular/core';
@Pipe({
  name:'replace-with-a'
})
```

```
export class RelaceWithAPipe implements PipeTransform {
  transform(value: number, unit: string) {
    return value.replace('a','b')
  }
}
```

Step 2 : Declare in NgModule

```
@NgModule({
  declarations: [AppComponent,RelaceWithAPipe ],
  imports: [BrowserModule,FormsModule,HttpModule],
  bootstrap: [AppComponent]
})
```

Step3 : Now in app.component.html we can use that

```
<p>{{strVal | replace-with-a}} </p>
```

There are two kinds of pipes in Angular Pure Pipe and Impure Pipe .

Every Custom Pipe is Pure by default in angular .

*inorder to make them Impure Pipe. we need to pass metadata as
pure : false to @Pipe Decorator*

```
@Pipe({  
  name: 'search',  
  pure: false  
})
```

```
1 import { Pipe, PipeTransform } from '@angular/core';  
2  
3 @Pipe({  
4   name: 'search',  
5   pure: false  
6 })  
7 export class SearchPipe implements PipeTransform {  
8  
9   transform(value: any, argument: any) {  
10  
11     if (argument === '') {  
12       return value;  
13     } else {  
14       return value.filter((item) => item.name.toLowerCase().indexOf(argument.toLowerCase()) > -1)  
15     }  
16   }  
17 }  
18
```

```
<input type="text" [(ngModel)]="searchVal" name="search" />  
  
<button (click)="add()">Add User</button> You, 5 minutes ago  
  
<table class="table">  
  <thead>  
    <tr>  
      <th>Name</th>  
      <th>Email</th>  
      <th>Age</th>  
      <th>Phone Number</th>  
      <th>status</th>  
    </tr>  
  </thead>  
  <tbody>  
    <ng-container *ngFor="let user of users | search: searchVal">  
      <tr *ngIf="user.status === 'active'">  
        <td>{{user.name | uppercase}}</td>  
        <td>{{user.email}}</td>  
        <td>{{user.age}}</td>  
        <td>{{user.phoneNumber}}</td>  
        <td>{{user.status}}</td>  
      </tr>  
    </ng-container>  
  </tbody>  
</table>
```

Difference between pure pipe and impure pipe ?

Pure pipe

Impure pipe

The pure pipe will not run on every
Change detection run

The pipe is executed on
every change detection
run

Pure pipe optimizes application
performance

Impure pipe may slow
down your application

21. Directive?

Directives are classes that add additional behavior to elements in your Angular applications. Use Angular's built-in directives to manage forms, lists, styles, and what users see.

Difference between components and directives

Component are also one of the type of directive with the template
Directives are without the template

Components lifecycles ngOnChanges(), ngOnInit(), ngDoCheck(),
ngDestroy() ,

ngAfterViewInit() , ngAfterViewChecked() , ngContentInit(),
ngContentChecked()

Directive lifecycles ngOnChanges() , ngOnInit() , ngDoCheck() ,
ngOnDestroy()

There are three types of Directives :

- Components - Components are directives with Template (or view)
- Attribute Directives - ([ngClass] , [ngStyle] ,[(ngModel)])
- Structural directives - (*ngIf , *ngFor , *ngSwitch)

- `<h1 *ngIf="!isShowForm">Login Form</h1>`
- `<h1 *ngIf="isShowForm">Register Form</h1>`

```
<div *ngFor="let user of usersList; let i = index">  
  <h1>{{user.id}} : {{user.name}}</h1>  
  <button (click)="delete(i)">Delete</button>  
</div>
```

```
delete(index:any){  
  console.log(index)  
  this.usersList.splice(index, 1)  
}  
// login() {
```

And also we can create our own / custom directives in angular

Creating Custom Directive : we can use ttClass as structural and attribute directive

```

1
2 import { Directive, ElementRef, Input, OnInit } from '@angular/core'
3
4 @Directive({
5   selector: '[ttClass]',
6 })
7 export class ttClassDirective implements OnInit {
8
9   @Input() ttClass: string;
10
11   constructor(private el: ElementRef) {
12   }
13
14   ngOnInit() {
15     this.el.nativeElement.classList.add(this.ttClass);
16   }
17
18 }
19

```

22. Service ?

A service is a class that is used to share the data across the application

How to create a service ?

Ng generate service service-name

```

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

@Injectable({
  providedIn: 'root',
})
export class HeroService {

}

```

Once we created a service we have to register in **RootModule** by using providers metadata

```
you, 1 hour ago | 1 author (you)
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [AppService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Or

providedIn : 'root' ?

If its providedIn root - it will be available across the application - because it will register in Root Module

/* Registering the service in a `@NgModule` will make it available in that Module only (Singleton within the Module Scope) */

Inorder to use the above service- we need inject that into component by using dependency injection mechanism

```
constructor(heroService: HeroService){ }
```

Dependency Injection ?

Dependency Injection, or DI, is a design pattern and mechanism for delivering some parts of an application to other parts of an application .

it can be done by using the `constructor(heroService: HeroService){ }`

In order use services we will use this dependency injection concept

23 . Singleton Service ?

A singleton service is a service for which only one instance exists in an app

We can create singleton service in two ways using providedIn: 'root' and registering in providers array in root module (app.module.ts) providers: [AppService]

24 . a Standalone component , directive , pipe , module ?

Components, directives, and pipes can now be marked as standalone: true. Angular classes marked as standalone do not need to be declared in an [NgModule](#) (the Angular compiler will report an error if you try).

Ng g c component_name --standalone

```
@Component({
  standalone: true,
  selector: 'photo-gallery',
  template: `
    <h1>This is a Standalone Component</h1>
  `,
})
export class PhotoGalleryComponent {
  // component logic
}
```

We need not to register in Standalone components in Root Module

If we want to use one standalone component in another standalone component we need to pass a standalone component in @Component() decorator . see below

```
@Component({
  standalone: true,
  selector: 'photo-gallery',
  imports: [ImageGridComponent] , //ImageGridComponent is
a one more standalone component
  template: `
    <app-image-grid><app-image-grid>
  `,
})
export class PhotoGalleryComponent {
  // component logic
}
```

25 . How to Migrate an Existing Angular Project to Standalone ?

```
ng generate @angular/core:standalone
```

26 . NgZone ?

ngZone is used to run the code outside of an angular application to improve the performance of the application

For Example if we are not depending upon the API response we can make that api call outside of angular application using ngZone

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'ngzone';
  count = 0;

  constructor(private appService: AppService, private ngZone: NgZone){

  }

  You, 12 minutes ago • Uncommitted changes

  getData(){
    this.ngZone.runOutsideAngular(()=>{
      this.appService.getData().subscribe((res)=>{
        console.log(res)
      })
    })
  }

  ngDoCheck(){
    console.log("NgDoCheck"+ this.count++)
  }
}
```

27. a ChangeDetection Strategy ?

Change detection is the process where angular will check all the components from top to bottom if any state has changed - to update the DOM.

There are two types of strategies

1. OnPush
2. Default

```
✓ @Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css'],
  // changeDetection: ChangeDetectionStrategy.OnPush
  changeDetection: ChangeDetectionStrategy.Default
})
```

Its used to improve the performance of the application

1. ChangeDetectionStrategy.OnPush // it will check when the change in input and properties
2. ChangeDetectionStrategy.Default - by default it will be applied //it will check every change detections

28. Angular Routing ?

Routing is used to navigate from page to another page without reloading the whole page called as single page application

In Angular routing is handled by RouterModule from @angular/router

The Router is a separate module in Angular. It is in its own library package, [@angular/router](#).

Using Angular Router you can

- Navigate to a specific view by typing a URL in the address bar
- Passing parameters in routes
- Protect the routes from unauthorized users using [Route Guards](#)

[How to implement routing in an angular application ?](#)

Setp : 1 :-

To make HTML5 routing work, we need to set up the “base href” in the DOM. This is done in the *index.html* file immediately after the head tag.

```
<base href="/">
```

[Step : 2](#)

Next, create an array of route objects. Each route maps the path (URL Segment) to the component - in this object

**** wildcard entry**

In imports array we need to write RouterModule.forRoot(routes)

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from
'@angular/router';
```

```

import { LoginComponent } from
'./login/login.component';
import { HomeComponent } from
'./home/home.component';
import { PageNotFoundComponent } from
'./page-not-found/page-not-found.component';
import { UserDetailsComponent } from
'./user-details/user-details.component';
import { isLoginGuard } from './is-login.guard';

const routes: Routes = [
  { path: 'login', component: LoginComponent },
  { path: 'home', component: HomeComponent,
    canActivate: [isLoginGuard] },
  { path: 'user-details/:userId', component:
UserDetailsComponent },

  { path: '', redirectTo: 'login', pathMatch:
'full' },
  { path: '**', component: PageNotFoundComponent
},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}

```

Step 3 : Import router module and pass routes object in
forRoot method

```
[RouterModule.forRoot(routes)]
```


Step4 - we have to update the path by using routerLink :
"active"

```
<li><a routerLink="/home"  
routerLinkActive="active">Home</a></li>
```

Step5 : once the path matches it will load in router-outlet

```
<router-outlet></router-outlet>
```

Main building blocks of Angular Routing :

1 . Router - its a service - constructor(private router: Router){ }
this.router.navigate(['/login']) - this is navigate to component
on button click

this.router.navigate(['/user-details' , val]) - this is navigate to
component on button click

2. RouterOutlet - its a directive - this helps to load the view when
pathmatches

```
<router-outlet></router-outlet>
```

3. ActivatedRoute - its used to get the params from path

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

```
constructor(private activatedRoute:ActivatedRoute ) {
```

```
}
```

```
ngOnInit(){
```

```
let id:any
```

```
=this.activatedRoute.snapshot.paramMap.get('id');
```

```
}
```

29 . What are Angular Router Guards ?

Route Guards are used to restrict the user until the user performs specific actions like login.

There are 5 types of guards

1. CanActivate
2. CanActivateChild
3. CanDeactivate
4. Resolve
5. CanLoad

Use cases for the CanActivate Guardm

- Checking if a user has logged in
- Checking if a user has permission

One of the use cases of this guard is to check if the user has logged in to the system. If the user has not logged in, then the guard can redirect him to the login page.

Service file :

```
2 import { Injectable } from '@angular/core';
3 import { Router, CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
4
5
6 @Injectable()
7 export class AuthGuardService implements CanActivate {
8
9   constructor(private _router: Router) {
10   }
11
12   canActivate(route: ActivatedRouteSnapshot,
13     state: RouterStateSnapshot): boolean {
14
15     //check some condition
16     if (someCondition) {
17       alert('You are not allowed to view this page');
18       //redirect to login/home page etc
19       //return false to cancel the navigation
20       return false;
21     }
22     return true;
23   }
24 }
25 }
```

```
{ path: 'product', component: ProductComponent, canActivate :
[AuthGuardService] },
```

Use Case for CanDeactivate

The best use case for `CanActivate` guard is the data entry component. The user may have filled the data entry and tries to leave that component without saving his work. The `CanDeactivate` guard gives us a chance to warn the user that he has not saved his work and give him a chance to cancel the navigation.

```
{ path: 'register', component: RegisterComponent,  
  canDeactivate:[DeactivateGuard] },
```

Use Case for Resolve :

We can use this guard to fetch the data from the backend API, before navigating to other page

Use Case for CanLoad :

CanLoad and CanActivate both or same purpose but the main difference between them is CanActivate is only for component and CanLoad for Module

```
{path: "admin", loadChildren: './admin/admin.module#AdminModule'  
, canLoad: [SampleService]},
```

Use Case for CanActivateChild:

This is also same as CanActivate but this for childRoutes

```
{ path: 'product', component: ProductComponent, canActivate  
: [AuthGuardService],  
  canActivateChild : [AdminGuardService],  
  children: [  
    { path: 'view/:id', component: ProductViewComponent },
```

```
]
},
```

30 . Lazy Loading in Angular ?

The Angular apps get bigger in size as we add more and more features. The [Angular Modules](#) help us to manage our app by creating separate modules for each new feature. But, as the app gets bigger in size, slower it loads. That is because angular loads the entire application upfront.

LazyLoading is the concept of loading the modules whenever that module is required. Instead of loading all modules all at one time. We can achieve this

```
{path: "admin", loadChildren: ()
=>import('./admin/admin.module').then(m => m.AdminModule) ,
canLoad: [AdminService]},
```

In order to implement to this lazy loading :

We have to create module by using the command

ng g module module-name (auth) –routing

This will generate two files like auth.module.ts and auth.routing.module.ts

Since this module is required two components as per our requirement like login and registration so we have to two create two component login and registration

After that we have to register them in auth.module.ts

Once that is done we have to create routes array for those two components in auth.routing.module.ts and that routes array we have to pass in forChild(routes)

And then we have to set the path for auth module in app.routing.module.ts file like below

Once the path is matched it will load that module with the help of router-outlet

```
TS app-routing.module.ts M X
src > app > TS app-routing.module.ts > routes
You, 18 minutes ago | 1 author (You)
1 import { NgModule } from '@angular/core';
2 import { Routes, RouterModule } from '@angular/router';
3
4 const routes: Routes = [
5   { path: 'auth', loadChildren: () => import('./auth/auth.module').then(m => m.AuthModule) },
6   { path: 'dashboard', loadChildren: () => import('./dashboard/dashboard.module').then(m => m.DashboardModule) },
7   { path: 'products', loadChildren: () => import('./products/products.module').then(m => m.ProductsModule) },
8   { path: 'orders', loadChildren: () => import('./orders/orders.module').then(m => m.OrdersModule) },
9   // Other routes...
10 ];
11
12 @NgModule({
13   imports: [RouterModule.forRoot(routes)],
14   exports: [RouterModule]
15 })
16 export class AppRoutingModule { }
```

auth.module.ts file

```
TS auth.module.ts U X
src > app > auth > TS auth.module.ts > AuthModule
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 import { AuthRoutingModule } from './auth-routing.module';
5 import { LoginComponent } from './login/login.component';
6 import { RegisterComponent } from './register/register.component';
7
8
9 @NgModule({
10   declarations: [
11     LoginComponent,
12     RegisterComponent
13   ],
14   imports: [
15     CommonModule,
16     AuthRoutingModule
17   ]
18 })
19 export class AuthModule { }
```

auth.routing.module.ts

```
TS auth-routing.module.ts U X
src > app > auth > TS auth-routing.module.ts > AuthRoutingModule
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { LoginComponent } from './login/login.component';
4 import { RegisterComponent } from './register/register.component';
5
6 const routes: Routes = [
7   {path: 'login', component: LoginComponent},
8   {path: 'register', component: RegisterComponent}
9 ];
10
11 @NgModule({
12   imports: [RouterModule.forChild(routes)],
13   exports: [RouterModule]
14 })
15 export class AuthRoutingModule { }
```

31 . What are Angular Forms ?

In general , forms are used to take the data from User.

In order to validate the forms in angular there are two approaches .

1.template driven form validation

2.Reactive form validation

Inorder to implement template driven form validation

Step 1 : we have to import FormsModule in app.module.ts in imports metadata array

Step 2 : write html form with ngForm

```
<form #loginForm="ngForm">
  <input type="text" name="username" ngModel #Username="ngModel" placeholder="User Name" required min="8" />
  <!-- Invalid : {{loginForm.invalid}} <br />
  Valid : {{loginForm.valid}} -->
  <!-- {{username?.touched}}
  {{username?.dirty}} -->
  You, 2 hours ago • initial commit
  <div *ngIf="username?.dirty">
    <div class="error" *ngIf="username?.errors?.['required']">Username is mandatory</div>
    <div class="error" *ngIf="username?.errors?.['minlength']">Username should be atleast 8 character</div>
  </div>
  <button (click)="login(loginForm)" [disabled]="loginForm.invalid">Login</button>
</form>
```

```
login(loginForm:any){
  console.log(loginForm.form)
  console.log(loginForm.form.value)
}
```

Template Driven Form Validation

In Template Driven Forms we specify behaviors/validations using directives and attributes in our template and let it work behind the scenes. All things happen in Templates hence very little code is

required in the component class. This is different from the reactive forms, where we define the logic and controls in the component class.

The Template-driven forms

1. The form is set up using `ngForm` directive
2. controls are set up using the `ngModel` directive
3. `ngModel` also provides the two-way data binding
4. The Validations are configured in the template via `ngModel` directive

Template-driven forms are

1. Contains little code in the component class
2. Easier to set up

While they are

1. Unit testing is a challenge
2. Not reusable

To work with Template-driven forms, we must import the `FormsModule`. We usually import it in the root module or in a `shared module`. The `FormsModule` contains all the form directives and constructs for working with forms

Open the `app.module.ts` and add the `import { FormsModule } from '@angular/forms';` to it.

And also add the `FormsModule` to the *imports metadata property array*

We have to use `ngForm` Directive for form tag

Reactive Form Validation :

To work with Reactive forms, we must import the `ReactiveFormsModule`. We usually import it in the root module or in a shared module

Then `<form [formGroup]="contactForm">`

With FormGroup :

```
contactForm = new FormGroup({
  firstname: new FormControl('',
[Validators.required]),
  lastname: new FormControl('',
[Validators.required]),
  email: new FormControl('',
[Validators.required]),
  gender: new FormControl('',
[Validators.required]),
  isMarried: new FormControl('',
[Validators.required]),
  country: new FormControl('',
[Validators.required])
})

onSubmit() {
  console.log(this.contactForm.value);
}
```

With FormBuilder :

In ts file :

```
import { FormBuilder } from '@angular/forms';
contactForm:any;
constructor(private formBuilder: FormBuilder) {
  this.myForm()
}
myForm(){
  this.contactForm = this.formBuilder.group({
```



```
        firstname: [''],
        lastname: [''],
        email: [''],
        gender: [''],
        isMarried: [''],
        country: [''],
    });
}
```

```
onSubmit() {
    console.log(this.contactForm.value);
}
```

Difference between template driven forms and reactive forms ?

1. Template-driven forms make use of the "FormsModule", while reactive forms are based on "ReactiveFormsModule".
2. In a template-driven approach, most of the logic is driven from the template, whereas in reactive-driven approach, the logic resides mainly in the component or typescript code. Let us get started by generating a component and then we'll update our form code.
3. Template-Driven forms are used for simple forms and Reactive-driven forms are used bigger forms

the Difference between FormGroup and FormBuilder ?

Using FormBuilder over FormGroup helps to improve application performance.

FormGroup will new FormControl - means it will create new object for each form control

```
this.form1 = new FormGroup({})
```

FormBuilder(helper class)

- creating FormBuilder object (Removing 'new' keyword)
- it needs to be injected in constructor

```
constructor(private _fb:FormBuilder) { }
```

```
this.form1 = this._fb.group({})
```

Difference between setValue() and patchValue

Angular **Forms** has three main building blocks i.e **FormControl**, **FormGroup** & **FormBuilder**. All these components have methods **setValue** & **patchValue** and behave differently

We use the **SetValue** to update the **FormControl** , **FormGroup** . When we use it to update the **FormGroup** the **SetValue** requires that the object **must match the structure of the FormGroup exactly**. Otherwise, it will result in an error.

The **PatchValue** is used to update only a subset of the elements of the **FormGroup**. It will only update the matching objects and ignore the rest.

```
setValues(){
  this.reactiveForm.setValue({
    username: 'XYZ',
    email: 'xyz@gmail.com'
  })
}

pathValues() {
  this.reactiveForm.patchValue({
    username: 'XYZ',
  })
}
```

You, 2 minutes ago • Uncommitted changes

Validators :

Validating the Forms is very important, otherwise, we will end up having invalid data in our database. The [Angular Forms](#) Module provides a few built-in validators to help us to validate the form. They are listed below.

```
lastname: new FormControl("",[Validators.maxLength(15),  
Validators.pattern("^[a-zA-Z]+$") , Validators.email ,  
Validators.required]),
```

1. [Required validator](#)
2. [Min length Validator](#)
3. [Max length Validator](#)
4. [Pattern Validator](#)
5. [Email Validator](#)

We also can create our own custom Validator :

Built-in validators are useful but do not cover all use cases. This is where we use the custom validator. It is very easy to create a custom validator in Angular.

Create one custom ts file . name of custom validator is **ValidateUrl**

```
src/shared/url.validator.ts

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

export function ValidateUrl(control: AbstractControl) {
  if (!control.value.startsWith('https') || !control.value.includes('.io')) {
    return { invalidUrl: true };
  }
  return null;
}
```

The above code uses the Notice **AbstractControl** class, which is the base class for **FormControls**, **FormGroups**, and **FormControl**. This allows access to the value of the **FormControl**.

Then in component.ts file

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { ValidateUrl } from '../shared/url.validator';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  myForm: FormGroup;

  constructor(private fb: FormBuilder) {}

  ngOnInit() {
    this.myForm = this.fb.group({
      userName: ['', Validators.required],
      websiteUrl: ['', [Validators.required, ValidateUrl]]
    });
  }

  saveForm(form: FormGroup) {
    console.log('Valid?', form.valid); // true or false
    console.log('Username', form.value.userName);
    console.log('Website URL', form.value.websiteUrl);
  }
}
```

Copy

Using Angular HttpClient

The `HttpClient` is a separate model in Angular and is available under the `@angular/common/http` package. The following steps show you how to use the `HttpClient` in an Angular app.

There are four methods in `http` , `get` ,`post`,`put` , `delete` ,

Import HttpClientModule in Root Module

We need to import it into our root module `app.module`. Also, we need to

it to the `imports` metadata array.

```
import { NgModule } from '@angular/core';  
import { HttpClientModule } from  
'@angular/common/http';
```

```
@NgModule({  
  declarations: [  
    AppComponent  
  ],  
  imports: [  
    HttpClientModule  
  ],  
  providers: [],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }
```

Import Required Module in Component/Service

Then you should import `HttpClient` the `@angular/common/http` in the component or service.

```
1
2import { HttpClient } from '@angular/common/http';
3
```

Inject `HttpClient` service

Inject the `HttpClient` service in the constructor.

```
1
2constructor(public http: HttpClient) {
3}
4
```

Call the HttpClient.Get method

2

You will need to import `HttpHeaders` as well to be able to use it



```
import { HttpHeaders } from '@angular/common/http';
```



So you will build up the headers as follows:



```
let headers = new HttpHeaders();
headers = headers.set('Authorization', 'Basic xzeydyt==');
```

You can then pass that onto the get method as follows:

```
this.http.get('url', { headers: headers });
```

Use `HttpClient.Get` method to send an [HTTP Request](#). The request is sent when we [Subscribe](#) to the `get()` method. When the response arrives map it the desired

object and display the result.

```
return this.http.get<Repos[]>(this.baseUrl + 'users/' + this.userName + '/repos')
  .subscribe(
    (response) => {                                //Next callback
      console.log('response received')
      console.log(response);
      this.repos = response;
    },
    (error) => {                                    //Error callback
      console.error('Request failed with error')
      alert(error);
    },
    () => {                                         //Complete callback
      console.log('Request completed')
    })
  }
```

What is HttpInterceptor ?

HttpInterceptor is used to handle HTTP requests and Error handling globally within an application.

Mostly we will use http interceptor for sending authentication token header by cloning the requested header in http interceptor

.ng g interceptor interceptor-name

app > TS interceptor.interceptor.ts > ...

```
import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor,
  HttpResponse
} from '@angular/common/http';
import { Observable, catchError, throwError } from 'rxjs';

@Injectable()
export class InterceptorInterceptor implements HttpInterceptor {

  constructor() {}

  intercept(req: HttpRequest<any>, next: HttpHandler): any {
    const authReq = req.clone({
      headers: req.headers.set('Authentication', 'Beare sdfsd')
        .set('header3', 'header 3 value')
    });
    return next.handle(authReq).pipe(
      catchError((error: HttpResponse) => {
        let errorMsg = '';
        return throwError((err: Error) => {
          console.log("Naveen")
          console.log(err)
        });
      })
    );
  }
}
```

```

    ],
    providers: [
      {
        provide: HTTP_INTERCEPTORS,
        useClass: InterceptorInterceptor,
        multi: true
      },
      AppService,
    ],
    bootstrap: [AppComponent]
  })
  export class AppModule { }

```

The **HttpRequest** contains URL, method, headers, body, and other request configuration.

The **HttpHandler** dispatches the **HttpRequest** to the next Handler using the method **HttpHandler.handle**.

@ViewChild() **@ViewChildren()** **@ContentChild()** **@ContentChildren()**?

@ViewChild() is a decorator from angular/core library - mainly its used to get the template element references and to access the child component methods and variables

@ContentChild() is a decorator from angular/core library - mainly its used to get the template element references of the projected content

@ViewChildren() is a decorator from angular/core library - mainly its used to get the multiple template element references

@ContentChildren() is a decorator from angular/core library - mainly its used to get the multiple template element references of the projected content

src/app/parent/parent.component.html

```
<p #parentElt>parent works!</p>
<button (click)='clickme()'>Click Me</button>
<button (click)='clickChildFn()'>clickChildFn</button>
<br>
<app-child #childComponentRef>
  ... <h1 #headerElt>Projected Content from parent</h1>
</app-child>
```

parent.component.ts

```

4  @Component({
5      selector: 'app-parent',
6      templateUrl: './parent.component.html',
7      styleUrls: ['./parent.component.css'],
8  })
9  export class ParentComponent {
10     @ViewChild('parentElt') parentElementRef!: ElementRef;
11
12     @ViewChild('childComponentRef') childComponent!: any;
13
14     clickme() {
15         this.parentElementRef.nativeElement.style.color = 'red';
16     }
17     clickChildFn() {
18         this.childComponent.childClickFn();
19     }
20 }

```

Child.component.html

```

<p>child works!</p>
<ng-content select='h1'></ng-content>

```

Child.component.ts

```

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css'],
})
export class ChildComponent {
  msg: string = 'asfasdf';
  @ViewChild('headerElt') headerElt!: ElementRef;
  childClickFn() {
    console.log('Clicked from parent');
    this.headerElt.nativeElement.style.color = 'red';
  }
}

```

@ViewChild() Static true and false?

When you use static true for ViewChild you can access the template reference at the time of ngOnInit() and ngAfterViewInit()

When you use `static: false` for `ViewChild` you can access the template reference at the time `ngAfterViewInit()` only not at `ngOnInit()`

Use `{ static: true }` when you want to access the `ViewChild` in `ngOnInit`.

Use `{ static: false }` will be accessible only in `ngAfterViewInit`. This is also what you want to do for when you have a structural directive (`*ngIf` etc.) in your template.

In most cases `{ static: false }` will work.

```
import { Component, OnInit, AfterViewInit, ViewChild, ElementRef } from '@angular/core'

@Component({
  selector: 'example',
  templateUrl: './example.component.html',
  styleUrls: ['./example.component.scss']
})
export class ExampleComponent implements OnInit, AfterViewInit
{
  @ViewChild('elementA', { static: true }) elementStatic: ElementRef<HTMLInputElement>;
  @ViewChild('elementB', { static: false }) elementDynamic: ElementRef<HTMLInputElement>;

  public ngOnInit(): void
  {
    this.elementStatic.nativeElement; // Ok
    this.elementDynamic.nativeElement; // ERROR TypeError: Cannot read property 'native
  }

  public ngAfterViewInit(): void
  {
    this.elementStatic.nativeElement; // Ok
    this.elementDynamic.nativeElement; // Ok
  }
}
```

```
<div #elementA>A</div>
<div #elementB>B</div>
```

Difference between constructor() { } and ngOnInit(){ }

constructor() is a special method for class that is executed when the class is instantiated. In angular we mainly use this for dependency injection . it will be called first time before the ngOnInit()

ngOnInit() is an angular lifecycle hook method called by Angular when the component gets initialized . it will be called after the ngOnChanges()

```
export class ConstructorNgoninitComponent {
  @Input() parentData: string = '';
  constructor(){
    console.log("Constructor is Called" , this.parentData) //undefined
  }
  ngOnInit(){
    console.log("NgOnInit Called", this.parentData) //parent data
  }
}
```

Difference between JIT and AOT Compiler ?

JIT - (Just in Time)

- It will Compile the code at runtime in the browser
- Because of the code is getting compiled at the time of runtime time - size of bundle will be larger
- Each file is compiled separately.
- Detect the template errors later at run time in the browser
- It will slow down the performance

AOT - (Ahead of Time)

- It compile the code at build time itself
- Because of the code is getting compiled at the time of build time itself - size of bundle will be smaller
- All code compiled together, inlining HTML/CSS in the scripts.

- Detect the template errors earlier at the time of compilation
- It will improve the performance

*From angular 9 version we have new compiler called **IVY** compiler - which is popular one to improve the performance of the application by reducing the bundle size*

*To enable IVY - in `tsconfig.json` file - we need to configure as **`angularEnableIvy : 'true'`***

What Files will generate when we run ng serve ?

Vendor.js file will have all third party code ,

Polyfills.js -it will have a piece of code that adds functionality to older browsers that have incompatibility issues.

Styles.css - all css code will be added in to this file

Main.js - all typescript code will be converted and added into this file

Runtime.js - it will be provide all these files to the browser for execution

Initial Chunk Files	Names	Raw Size
vendor.js	vendor	2.19 MB
polyfills.js	polyfills	333.06 kB
styles.css, styles.js	styles	230.45 kB
main.js	main	10.47 kB
runtime.js	runtime	6.51 kB

How to Improve the performance of the application ?

- 1 . By using the LazyLoading Concept
2. By using the ChangeDetectionStrategy

3. By Using the Pure pipes
4. By using the TrackBy for *ngFor

```
<div *ngFor="let user of users; trackBy:userByName">  
  {{user.name}} -> {{user.score}}  
</div>
```

Ex :

5. By removing unused variables and unwanted imports
6. We need unsubscribe() to all observable while leaving from the component on ngOnDestroy() if required
7. Removing the console.log() after debugging completed while giving the build
8. Running the build command with aot compiler : ng build --prod --aot - it will optimize the code

What Scanners have you used for code scan ?

SonarLint and SonarQube Code Scanner

What are the new Angular Features ?

ViewChild static true to get ElementRef at ngOnInit

@Input({ required: true }) defaultColor: string;

trackBy for ngFor

Guards has been changed to Functions in order to inject them we need to use inject()

Standalone component migration

ngIf and elseBlock

Support For Tailwind CSS

increase Security

Increase the performance

RXJS and Operators ?

Reactive Extensions for Javascript

Few operators from rxjs

1. Observable
2. Observer
3. of
4. from
5. pipe
6. map
7. filter
8. retry
9. retryWhen
10. take
11. takeUntil
12. switchMap
13. mergeMap
14. concatMap
15. exhaustMap
16. forkJoin
17. debounce
18. catchError
19. throwError

Observable :

Observable is used to handle the stream of data. In order to get that stream data we need to subscribe () to it otherwise it will not emit that data . to unsubscribe the observable we need call function unsubscribe()

```

const observable = new Observable((subscriber: any) => {
  subscriber.next(1);
  subscriber.next(2);
  subscriber.next(3);
  setTimeout(() => {
    subscriber.next(4);
    subscriber.complete();
  }, 1000);
});

let subscription = observable.subscribe(
  (next: any) => {
    console.log(next);
  },
  (error) => {
    console.log(error);
  },
  () => {
    console.log('Completed');
  }
);

subscription.unsubscribe();

```

Interval() :-

The RxJS interval() operator returns an observable that emits a sequentially increasing number within the given interval of time.

```

ngOnInit() {
  this.intervalObserver = this.getInterval().subscribe(
    (res: any) => {
      console.log(res);
    },
    (err: any) => {
      console.log(err);
    },
    () => {
      console.log('Completed');
    }
  );
}

getInterval() {
  return interval(1000);
}

```

Difference between of() and from() operator ?

It is important to note the difference between of and from when passing an array-like structure (including strings):

Of would print the whole array at once.

From prints the elements 1 by 1.

```
19 export class AppComponent {
20   title = 'CodeSandbox';
21   arr: any = [1, 2, 3];
22
23   ngOnInit() {
24     //of operator
25     var sub = of(this.arr);
26     sub.subscribe((res: any) => {
27       console.log(res); // output : [1,2,3]
28     });
29
30     //from operator
31
32     var sub2 = from(this.arr);
33     sub2.subscribe((res: any) => {
34       console.log(res); // output : 1,2,3
35     });
36   }
37 }
```

pipe() in rxjs :

pipe() function in RxJS: You can use pipes to link operators together. Pipes let you combine multiple functions into a single function.

```
ngOnInit() {
  this.http
    .get('https://jsonplaceholder.typicode.com/todos')
    .pipe(
      delay(20),
      retry(5)
    )
    .subscribe((res) => {
      console.log(res);
    });
}
```

***retry()* :**

Sometimes we will get an error as a bad gateway and it may also break due to a bad network connection also. When we want to retry again we can use `retry()` function

delay()* / *debounce() : it used to delay the api call for sometime. Mostly `debounce()` is used to implement the auto complete search functionality with the help of `switchMap()` too.

takeUntil() : mostly we will use the `takeUntil` to unsubscribe the observable. When we are leaving from the component

```
@Component({
  selector: "app-flights",
  templateUrl: "./flights.component.html"
})
export class FlightsComponent implements OnDestroy, OnInit {
  private readonly destroy$ = new Subject();

  public flights: FlightModel[];

  constructor(private readonly flightService: FlightService) {}

  ngOnInit() {
    this.flightService
      .getAll()
      .pipe(takeUntil(this.destroy$))
      .subscribe(flights => {this.flights = flights});
  }

  ngOnDestroy() {
    this.destroy$.next();
    this.destroy$.complete();
  }
}
```

Merge() : instead of subscribing to each observable - we can subscribe to all observable at once

mergeAll() : to merge all the Observable

takeUntilDestroyed() : mostly we will use the takeUntilDestroyed to unsubscribe the observable when we are leaving from the component. And also for this we need inject **DestroyRef**

```
export class MyTestClass {  
  constructor(  
    public overlay: OverlayRef,  
    destroyRef: DestroyRef,  
  ) {  
    this.overlay  
      .backdropClick()  
      .pipe(takeUntilDestroyed(destroyRef))  
      .subscribe(() => {  
        //  
      });  
  }  
}
```

map() :map works exactly the same for Observables as it does for arrays. You use a map to transform a collection of items into a collection of different items.

```
from([1, 2, 4])  
  .pipe(map((item) => item * 2))  
  .subscribe((res) => {  
    console.log(res); //output: 2,4,8  
  });
```

filter() : filter is used to filter out elements based on certain conditions.

```
from([1, 2, 4, 2, 1, 3, 5])
  .pipe(filter((item) => item > 2))
  .subscribe((res) => {
    console.log(res); //output: 4,3,5
  });
```

BehaviourSubject() vs Subject():

A BehaviorSubject holds one value. When it is subscribed it emits the value immediately. A Subject doesn't hold a value.

Subject example (with RxJS 5 API):

```
const subject = new Rx.Subject();
subject.next(1);
subject.subscribe(x => console.log(x));
```

Console output will be empty

BehaviorSubject example:

```
const subject = new Rx.BehaviorSubject(0);
subject.next(1);
subject.subscribe(x => console.log(x));
```

Console output: 1

switchMap() vs mergeMap vs concatMap() vs exhaustMap():

SwitchMap only emits the latest observable value and unsubscribe() the previous observable.

flatMap/mergeMap collects all individual observables and returns all observables in a single array without caring about the order of observable. works asynchronously.

concatMap preserve the order and emits all observable value, works synchronously

exhaustMap - it will ignore all the observables while the previous Observable is not completed

forkJoin() :

To group the all the observables and it will wait for all to complete. if any observable throw an error - it will not work

One common use case for this is if you wish to issue multiple requests on page load (or some other event) and only want to take action when a response has been received for all'

```
forkJoin([
  this.httpHandlerCached.getListA(),
  this.httpHandlerCached.getListB(),
  this.httpHandlerCached.getListC(),
  this.route.params.pipe(take(1))
]).subscribe(res => {
  this.listA = res[0];
  this.listB = res[1];
  this.listC = res[2];
  this.doSomethingWithFetchedLists();
});
```

catchError () and throwError () :

catchError is used handle the errors

```
this.http.get("url").pipe(
  catchError(this.errorHandler(error))
)
errorHandler(error:any){
  return throwError(error.message)
}
```



```
return this.http.post('', data).pipe(  
  catchError(error => this.handleErrorObservable(error))  
);
```

```
import {throwError} from 'rxjs';  
  
private handleErrorObservable(error: Response | any) {  
  return throwError(error.message || error);  
}
```

What is Tree shaking?

In Angular, tree shaking, combined with tools like Webpack, helps in reducing the bundle size by removing unused parts of the code (such as functions, variables, or modules) during the build process.

JAVA SCRIPT

1. var vs let operator ?

The difference between let and var is in the scope of the variables - when we create Variables declared by **let** are only available inside the block scope. Variables declared by **var** are available throughout the function / Global scope.

Var variables can be hoisted

Let variable can't be hoisted

```
1 function fun(){
2   var x = 10;
3   if(true){
4     let y = 15;
5     var z = 20;
6   }
7   console.log(z); //output : 20
8   console.log(y); //output : not defined
9 }
10 fun();
```

2 . output of below of this code ?

```
1 function fun(){
2   var arr = [1,2,3,4]
3   arr.length = 0; // making the array empty
4   console.log(arr[0]) // output : undefined
5 }
6 fun();
```

3 . Difference between undefined and not defined ?

Undefined: It occurs when a variable has been declared but has not been assigned any value. Undefined is not a keyword.

Not defined : It occurs when we try to access any variable that is not declared

4 . null vs undefined?

typeof null is object

typeof undefined is undefined

5.Splice Vs Slice?

`splice()` changes the original array whereas `slice()` doesn't but both of them returns array object.

See the examples below:

```
var array=[1,2,3,4,5];  
console.log(array.splice(2));
```

This will return `[3,4,5]` . The **original array is affected** resulting in `array` being `[1,2]` .

```
var array=[1,2,3,4,5]  
console.log(array.slice(2));
```

This will return `[3,4,5]` . The **original array is NOT affected** with resulting in `array` being `[1,2,3,4,5]` .

6. Map Vs Filter vs reduce?

The difference between map and filter method is that – map transforms each element of an array based on a transformation function and returns a new array of the same length, While, filter creates a new array with only the elements that satisfy a specified condition.

Reduce function will go through the each element in array and result into single value

```
1 ▾ function fun(){
2   var arr = [1,2,4,4,5,6];
3
4 ▾ arr.map((elt)=>{
5   return elt*4;
6 })
7 ▾ arr = arr.filter((elt)=>{
8   return elt > 2;
9 })
10 console.log(arr);
11
12 }
13 fun();
```

6. Hoisting ?

Hoisting is JavaScript's default behavior of moving all declarations to the top in global execution context.

This allows functions and variables to be used before they are declared.

7 . Recursive Functions ?



A function that calls itself is called a recursive function

1		node /tm
2	var x=0;	1
3	function fun(){	2
4	if(x<10){	3
5	x++;	4
6	console.log(x)	5
7	fun();	6
8	}	7
9	}	8
10		9
11	fun();	10

8 . the Callback function ?

Any function that is passed as an argument to another function is called as a callback function.

A JavaScript callback is a function which is to be executed after another function has finished execution.

main.js	  	Output
<pre> 1- function greet(name, callback) { 2 console.log('Hi' + ' ' + name); 3 console.log(callback) 4 callback(); 5 } 6 // callback function 7- function callMe() { 8 console.log('I am callback function'); 9 } 10 // passing function as an argument 11 greet('Peter', callMe); </pre>		<pre> node /tmp/goNZ8ZUGh3.js Hi Peter [Function: callMe] I am callback function </pre>

9 . Closures in javascript?

A closure is a feature of JavaScript that allows inner functions to access the outer scope of a function variables.

Lexical scope is the ability for a function scope to access variables from the parent scope. We call the child function to be lexically bound by that of the parent function.

```
1 function fun1(){  
2     var x = 10;  
3     function fun2(){  
4         console.log(x)  
5     }  
6     fun2();  
7 }  
8 fun1();
```

10. Observables Vs Promises?

Observables	Promises
Emit multiple values over a period of time.	Emit a single value at a time.
Are lazy: they're not executed until we subscribe to them using the subscribe() method.	Are not lazy: execute immediately after creation.
Have subscriptions that are cancellable using the unsubscribe() method, which stops the listener from receiving further values.	Are not cancellable.
Provide the map, for, forEach, filter, reduce, retry, and retryWhen operators.	Don't provide any operations.
Deliver errors to the subscribers.	Push errors to the child promises.

11 . call , apply and bind?

All three functions are used to borrow the function from another object .

Call() : The [call\(\)](#) method calls the [function](#) with a given value and allows passing in arguments one by one separating them with commas:

apply() : The [apply\(\)](#) method calls the function with a given value and allows passing in arguments as an [array](#) (or an array-like object).

bind() : The [bind\(\)](#) method returns a new function and allows passing in a **this** array and any number of arguments.

```
let nameObj = {  
  name: "Tony"  
}
```

```
let PrintName = {  
  name: "steve",  
  sayHi: function (age) {  
    console.log(this.name + " age is " + age);  
  }  
}
```

```
PrintName.sayHi.call(nameObj, 42);
```

```

let obj1 = {
  firstName: "navee",
  lastName: "Kumar",
  getName: function(...p){
    console.log(p)
  }
}
let obj2 = {
  firstName: 'oara',
  lastName: 'sdafad'
}
var fun = obj1.getName.bind(obj2 , 'HI');
fun()

```

12 . Async / Await ?

Async/await is a JavaScript feature that allows you to write asynchronous code that looks and behaves like synchronous code.

The async keyword is used to define an asynchronous function. An asynchronous function is a function that returns a promise. The await keyword is used to wait for a promise to resolve before executing the next line of code

```

const getData = async () => {
  const response = await fetch("https://jsonplaceholder.typicode.com/todos/1")
  const data = await response.json()


  console.log(data)
}

getData()

```


'use strict' ;

Use strict is used to run the javascript code in strict mode only

```
users / chaitanaveen.reddy / Downloads /  scripts  
'use strict'; //with use strict  
x=12;  
  
console.log(x); //error  
  
_____  
  
//without use strict|  
x=12;  
console.log(x); //output : 12
```

1. 13 . Regular function vs arrow functions ?

Inside a **regular JavaScript function**, `this` value is dynamic.

it means that the value of `this` depends on *how* the function is invoked. During a *simple invocation* the value of `this` equals the global object (or `undefined` if the function runs in **strict mode**)

`this` value inside of an **arrow function** always equals `this` value from the outer function

```
var obj = {  
  name: 'Naveen',  
  fun1: function () {  
    console.log(this)  
    var arrow = () => {  
      console.log(this)  
    }  
    arrow();  
  },  
  fun2: () => {  
    console.log(this)  
  }  
}  
obj.fun1();  
obj.fun2();
```

There is no **arguments** object in arrow function and **arguments** object will be there in regular function

```
function fun1(){
  console.log("Normal Function")
  console.log(arguments) // [1,2,3,4]
  console.log("-----")
}
fun1(1,2,3,4);

var arrowFun = () => {
  console.log("Arrow Function")
  console.log(arguments) // undefined
}

arrowFun(1,2,4);
```

13 . event bubbling and event propagation ?

Bubbling is the process of an event happening on a parent element when we click on a child element .

We can fix this by using **event.stopPropagation();**

14 . Prototype in javascript ?

All JavaScript objects inherit properties and methods from a prototype:

The JavaScript prototype property allows you to add new properties to object constructors:

15.Difference Between Authorization and Authentication?

Authorization is the process of giving the Access to the user.
Authentication is the process of recognizing the user Identity.

16.CORS issue?

Cors issue will happen when a web page makes a request to a different domain.In order to overcome this issue we can use Access-origin-allow-control at Backend.

HTML

HTML : Hyper text markup language(it is the standard markup language for creating Web pages)and it consists of a series of elements

Structure of HTML:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

The **<!DOCTYPE html>** declaration defines that this document is an HTML5 document(it tells to the browser which markup version we are using)

What are the Schematic Elements?

Schematic elements are meaningful elements such as Header,footer,Img,Input Type,Button,Table.

SCSS/SASS

mixin and include?

The **@mixin** directive makes the CSS code to reuse;

The **@include** directive makes the mixin css code to include another class

```
@mixin important-text {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
}
```

What are the Block level elements?

Block level elements display one by one.

ex:<div>

Inline Block level elements?

It displays Side by Side.

ex:

```
.danger {
  @include important-text;
  background-color: green;
}
```

Variable is SASS ?

We also can declare variables in SASS

\$myFont: Helvetica, sans-serif;

\$myColor: red;

\$myFontSize: 18px;

\$myWidth: 680px;

```
body {
  font-family: $myFont;
  font-size: $myFontSize;
  color: $myColor;
}
```

@extend in SCSS ?

The `@extend` directive lets you share a set of CSS properties from one selector to another.

```
.button-basic {
  border: none;
  padding: 15px 30px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}

.button-report {
  @extend .button-basic;
  background-color: red;
}
```

How to add comments in CSS ?

```
/* content */
```

the 'a' in rgba mean?

```
h1 {
  color: rgba(R, G, B, A);
}
```

R - red , G- Green, b-Blue , a - (alpha/opacity)

CSS HSL Colors?

Hue , Saturation, Lightness

20. What is the difference between CSS border and outline?

- **CSS border** properties allow us to set the style, color, and width of the border.
- **CSS outline** property allows us to draw a line around the element, outside the border.

23. Can we add an image as a list item marker?

To add an image as the list-item marker in a list, we use the [list-style-image](#) property in CSS.

Syntax:

```
list-style-image: url( './' );
```

28. What is CSS overflow?

The CSS overflow controls the big content. It tells whether to add scroll bars. The overflow contains the following property:

- visible
- hidden
- scroll
- auto

1. Visible: The content is not clipped and is visible outside the element box.

2. Hidden: The overflow is clipped and the rest of the content is invisible.

3. Scroll: The overflow is clipped but a scrollbar is added to see the rest of the content. The scrollbar can be horizontal or vertical.

4. Auto: It automatically adds a scrollbar whenever it is required.

To set Background image ?

1. background: white url('good-morning.jpg');
2. background-repeat: no-repeat;
3. background-attachment: fixed;
4. background-position: center;

Pseudo-elements:

- ::before
- ::after

Structural pseudo-classes:

- :first-child
- :nth-child(n)
- :last-child

100vh vs 100% ?

height: 100vh = 100% of the viewport height

height: 100% = 100% of the parent's element height

Opacity?

The opacity CSS property sets the transparency to an element

Tweening in CSS ?

Tweening in [CSS](#) basically means animations -

It is the process of generating intermediate frames between two images In CSS3, the Transforms (matrix, translate, rotate, scale etc.) module can be used to achieve tweening.

```
div.a {  
  transform: rotate(20deg);  
}
```

[Explain the concept of Tweening in CSS - GeeksforGeeks](#)

Z-index?

z-index is the CSS property that controls the stacking order of overlapping elements on a page.

How many ways can we define styles?

Inline - by using the style attribute inside HTML elements.

- Internal - by using a <style> element in the <head> section.
- External - by using a <link> element to link to an external CSS file.

Css Selectors?

Tag,id,class,universal

Box model?

In Css Box model is a container that contains content-padding-border-margin

PSEUDO Classes :

```
/* unvisited link */
a:link {
  color: red;
}

/* visited link */
a:visited {
  color: green;
}

/* mouse over link */
a:hover {
  color: hotpink;
}

/* selected link */
a:active {
  color: blue;
}
```

Link: class represents an element that has not yet been visited.

visited CSS pseudo-class applies once the link has been visited by the user.

Active pseudo class is to show that the element is in active state.

display:none , visibility:hidden?

Both are used to hide the elements but the only difference is occupying the space. Display none will not occupy the space, but visibility hidden will occupy the space.

Display:block,display:none?

Display:none; means the element will not be displayed, and

Display:block; means the element is displayed as a block-level element

flexbox and grid?

flexbox was designed for layout in one dimension - either a row or a column. Grid was designed for two-dimensional layout - rows, and columns at the same time.

Float left and float right?

Float left moves elements to the left, float right will move elements to the right

Position Relative and position Absolute?

Position : absolute

Position: sticky

Position: fixed

Position: relative

Position: static

If the element is Positioned **relative** to its normal position and

if the element is positioned **absolutely** to the position of parent.

An element with position: **sticky**; is positioned based on the user's scroll position

NGRX STORE MODULE :

NgRx store is used to maintain the state management. by Using store states, actions, and reducers in NgRx

Actions are methods dispatched by the component when an event is called.

To trigger an action, we use the reducers

In order to work with ngrx store

npm install --save @ngrx/store

and then we have to create interface

```
export interface Product {  
  name: string;  
  price: number;  
}
```

And then we need to create reducer by importing Action from ngrx store and it will have state and action.type - and also by default we can maintain state

```

import { Product } from '../product/product.model';
import { Action } from '@ngrx/store';

export const ADD_PRODUCT = 'ADD_PRODUCT';

export function addProductReducer(state: Product[] = [], action) {
  switch (action.type) {
    case ADD_PRODUCT:
      return [...state, action.payload];
    default:
      return state;
  }
}

```

And then in root module we have to register StoreModule by passing reducer to forRoot method

```

import { StoreModule } from '@ngrx/store';
import { addProductReducer } from '../reducers/product.reducer';

imports: [
  BrowserModule,
  StoreModule.forRoot({product: addProductReducer})
],

```

And in component we can dispatch the data by injecting store service from @ngrx/store

```

export class ProductComponent implements OnInit {
  products: Observable<Product[]>;
  constructor(private store: Store<AppState>) {
    this.products = this.store.select(state => state.product);
    console.log(" this.products", this.products)
  }
  addProduct(name:any, price:any) {
    this.store.dispatch({
      type: 'ADD_PRODUCT',
      payload: <Product> {
        name: name,
        price: price
      }
    });
  }
  ngOnInit() {

```

install bootstrap in angular?

Bootstrap is a framework and It supports responsive design by adjusting the styles based on the different types of device sizes.

Inorder to install bootstrap in angular application

npm install bootstrap

Then in **angular.json** file in **styles** property we need configure that

```
"styles": [
  "node_modules/bootstrap/scss/bootstrap.scss",
  "node_modules/bootstrap-icons/font/bootstrap-icons.css",
  "src/styles.scss"
],
"scripts": [
  "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"
]
```

types of layout available in Bootstrap?

- **Fluid Layout:** Fluid layout is used when you want to create a app that is 100% wide and use up all the width of the screen
- **Fixed Layout:** For a standard screen you will use fixed layout (940 px) option
-

Few Important Classes from bootstrap :

container and container-fluid

The only difference between container-fluid and container is that the container is not full-width on larger screens. The container is a fixed width that's centered with large margins on the sides. container-fluid doesn't resize, it's always 100% width.

row :

Bootstrap's grid system allows up to 12 columns across the page by using **row** class.

```

<div class="row">
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
  <div class="col-md-1"></div>
</div>

```

Class prefix	.col-xs	.col-sm	.col-md	.col-lg
Screen width	<768px	>=768px	>=992px	>=1200px

Button classes

.btn

.btn-primary

.btn-secondary

Alert Classes

.alert

.alert-success

.alert-danger

Form Classes:

form-group

form-control

Angular Material:

In order to work with angular material we have install by using the cmd
npm install @angular/material

Button : `<button mat-button>Click me!</button>`

Forms : need to import `MatInputModule` in root module and need use
`mat-form-field` , `matInput` , `mat-datepicker`

```
import {MatInputModule} from '@angular/material/input';
<form class="example-form">
  <mat-form-field class="example-full-width">
    <input matInput placeholder="Favorite food"
value="Sushi">
  </mat-form-field>

  <mat-form-field class="example-full-width">
    <textarea matInput placeholder="Leave a
comment"></textarea>
  </mat-form-field>
</form>
```

Date Picker : `import {MatDatepickerModule} from
'@angular/material/datepicker';`

```
<mat-form-field>
  <mat-datepicker #picker></mat-datepicker>
```

</mat-form-field>

<ng-template></ng-template>

Ng-template is a container to group the elements. It will not render it in the dom or do not disturb the styles . It helps to create dynamic templates for example in order to load the elseBlock of **ngIf** .

```
<div>
  Ng-template Content
  <div *ngIf="false else showNgTemplateContent">
    Shouldn't be displayed
  </div>
</div>

<ng-template #showNgTemplateContent> Should be displayed
</ng-template>
```

<ng-container></ng-container>

Ng-container is just a container to group the elements- but it will not render in the DOM - we will use this for structural Directives like ***ngFor** and ***ngIf**

<ng-content></ng-content>

Ng-content it is placeholder for projected content

UNit Testing :

Yes i have written the test cases also - but by taking the reference of other files which are written by someone or going through documents

In order to run we can use this cmd : ng test

fewFunctions in unit testing . like expect , toEqual , toBeTruthy , toBeFalsy

)

```
clickHandler(value: string) {  
  if (value === 'foo') {  
    return true;  
  }  
  return false;  
}
```

```
it('should return value', () => {  
  expect(component.clickHandler('foo')).toEqual(true);  
  expect(component.clickHandler('bar')).not.toEqual(true);  
})
```


Interface :

Interface is a specification that identifies a related set of properties and methods to be implemented by a class. So basically using interface you can set some basic rules for your properties and methods using class.

Sometime we are creating an object array with a specific data type field like id has to be integer or number, name has to be string value, birth of date have date data type data. but sometimes we might make a mistake and set the string value instead of integer or number then it cought problem and your app will show you an error. But if you use the interface then it will solve problems when you write code on your IDE. IDE will show you errors quickly where there is a problem.

To define interface :

```
export interface Student {  
    id: number;  
    name: string;  
}
```



```
import { Component } from '@angular/core';  
  
interface Student {  
    id: number;  
    name: string;  
}  
  
@Component({  
    selector: 'my-app',  
    templateUrl: './app.component.html',  
    styleUrls: [ './app.component.css' ]  
})  
export class AppComponent {  
    name = 'Angular';  
  
    students: Student[] = [  
        {id: 1, name: "Hardik"},  
        {id: 2, name: "Paresh"},  
        {id: 3, name: "Rakesh"},  
    ]  
}
```

What are the features in typescript?

Spread Operator and Rest Operator

Spread operators allow us to expand an array or object into its individual elements, while rest operators allow us to condense multiple elements into a single array or object.

```
function foo(...data){ //rest operator
  console.log(data)
}

var arr= [1,2,5,6]
var arr1 = [3,4]
var newArr= [1,2,...arr1, 5,6] //spread operator
console.log(newArr)

foo(1,2,4,5);
```

Arrow Functions

```
var foo = ()=>{
  console.log("Arrow Function")
}

foo();
```

Ternary operator

```
var x = 1;
var y = 0;
var msg = check(x,y) ? 'X is greater than Y' : 'X is less than Y';
console.log(msg)

function check(x,y){
  if(x > y){
    return true;
  } else{
    return false;
  }
}
```

Optional Parameters

Any Data Type

Nullish Coalescing Operator

Interface

Template strings

```
var age = 12;
var name = "chandu";

console.log("My Name is "+name+ " Age is : "+age)

console.log(`My Name is ${name} Age is ${age}`)
```

What is a first class function ?

In Javascript - functions are the first class objects -First class functions means that language functions are treated like a variable .

For example, in such a language, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable.

Debouncing: **Debouncing** is a technique where you delay the execution of a function until after a certain amount of time has passed. This is useful if you have a frequently used function—say, a scroll or resize event listener—and don't want to trigger it too frequently because that might slow down the browser.

Throttling: Throttling is a similar technique to debouncing, but instead of delaying the execution of a function, it limits the rate at which a function. This is useful when a function, such as a mousemove or keydown event listener, may be called repeatedly but need not be run each time.

Arrays built in methods :

push() - push method will add one or values in array at **the end**

```
var arr = [1,2,3,4]
arr.push(5)
console.log(arr): //[ 1, 2, 3, 4 ,5]
```

unshift() - unshift method will add one or more values in array **at the beginning**

```
var arr = [1,2,3,4]
arr.unshift(5)
console.log(arr): //[ 5, 1, 2, 3, 4 ]
```

pop() - The pop() method removes (pops) the last element of an array.

```
var arr = [1,2,3,4]
arr.pop()
console.log(arr): //[ 1, 2, 3,]
```

shift()- The shift() method removes (pops) the first element of an array.

```
var arr = [1,2,3,4]
```

```
arr.shift()
```

```
console.log(arr): //[ 2, 3,4]
```

indexOf() - the indexOf() method is used find the index of specified value

if value is there it will return index of that value

If value is not there it will return -1

```
var arr = [1,2,3,4]
var index = arr.indexOf(3)
console.log(index) // 2
```

```
var index2= arr.indexOf(6)
console.log(index2) //-1
```

includes()

The includes() method returns true if an array contains a specified value.

```
var arr = [1,2,3,4]
var val = arr.includes(3)
console.log(val) // true
```

```
var val2= arr.includes(6)
console.log(val2) //false
```

join()

The join() method returns an array as a string by concatenating the entire array.

```
const date = ["12", "18", "2023"];
let formattedDate = date.join('-');
console.log(formattedDate)
```

split() : the split method will return as array

Var str = "how are you";

Var data = str.split(" "); //splitting with space

```
console.log(data); // ["how", "are", "you"]
```

```

var date = new Date();
console.log(date)//2023-12-17T09:49:06.740Z
console.log(date.toLocaleDateString())//2023-12-17T09:49:06.740Z

var formatterDate= date.toLocaleDateString();
//var formatterDate= 12/17/2023;

var splitted = formatterDate.split('/');
console.log(splitted) //[ '12', '17', '2023' ]

```

slice()

splice()

forEach()

map()

filter()

reduce()

some(): some() method is used to check the array if array has at least one element that will meet specific condition

```

const numbers = [1, 2, 3, 4, 5];

const testResult = numbers.some(function (element) {
  return element > 4;
});

console.log(testResult); // true

```

every() : In JavaScript, the array every() method checks whether all the given elements in an array are satisfying the provided condition.

```

1  const numbers = [1, 2, 3, 4, 5];
2  const testResult = numbers.every(function (element) {
3    return element > 4;
4  });
5  console.log(testResult); // false
6
7  const numbers2 = [11, 21, 34, 41, 52];
8  const testResult2 = numbers2.every(function (element) {
9    return element > 4;
10 });
11 console.log(testResult2); // true
12

```

of and **in** – **of** used to loop through the **values of an array** and **in** used to loop through the **keys of an object**

//of example

main.js	Output
<pre>1 const cars = ["BMW", "Volvo", "Mini"]; 2 3 let text = ""; 4 for (let x of cars) { 5 console.log(x) 6 }</pre>	<pre>node /tmp/uP40XpAn6J.js BMW Volvo Mini</pre>

//in example

main.js	Output
<pre>1 const obj = { 2 name : 'Naveen', 3 age: 12, 4 number: 43324324 5 } 6 for (let x in obj) { 7 console.log(x) 8 }</pre>	<pre>node /tmp/uP40XpAn6J.js name age number</pre>

Data Types in Javascript :

String

Number,

Null - typeof null is object

Undefined - typeof undefined is undefined

Object

Symbol

Boolean

Primitive Data Types

String

Number

Null

Undefined

Symbol

Non Primitive Data types

Object

Array

Symbol :

A JavaScript Symbol is a primitive datatype just like Number, String, or Boolean.

It's used to hold the unique hidden property in any object

`typeof Symbol()` is symbol

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
let id = Symbol('id');
person[id] = 140353;
// Now person[id] = 140353
// but person.id is still undefined
```

Set and Map:

Set is a collection of values - in set each value occurs only at once - set can hold any data type value

```
var collection = new Set();
collection.add(1) //to add values to set
collection.add(2)
collection.add(3)
collection.add(3)

console.log([...collection])
console.log(collection.size) //3 : to get length of set
collection.delete(1)
console.log(collection.size) //2 : to get length of set
collection.clear() - //to clear the set
console.log(collection)
```

node /tmp/t0ksvtYRnL.js
[1, 2, 3]
3
2
Set(0) {}


```

var map = new Map();
map.set('a', 1);
map.set('b', 2);
map.set('c', 3);
map.set('C', 3);
map.set('a', 4); // Has: a, 4; b, 2; c: 3, C: 3
assert(map.size, 4);

```

Map	Set
It is a collection of key-value	It is a collection of unique elements
Map is two-dimensional	The set is one dimensional
Values are accessed using keys	In-built methods are used to access values

```

// var mainObj= {};
// var x = {};
// var y = {
//     num : 1
// };
// mainObj[x]= "Sample Description";
// mainObj[y]="Other Description";
// console.log(mainObj);

// var map = new Map();
// map.set('name' , 'Naveen');
// map.set(x , 'obj1');
// map.set(y, 'obj2')

// for(let [key , obj] of map.values()){
//     console.log(key , obj)
// }
// console.log(map.entries())
// for(let [key , obj] of map.entries()){
//     console.log(key , obj)
// }
// console.log([...map])

```

Maps key will hold the value every originally its removed - it can't be garbage collected for that we should use WeakMap

Copy / Shallow Copy Vs Deep Copy:

```
var obj = {
  name : 'naveen',
  age : 18,
  skills: {
    primary: 'Angular',
    secondary: 'PHP'
  },
  foo: function(){
    console.log('asdf')
  }
}

var obj2 = obj; //Shallow Copy or Copy - both objs will effect obj.name also will change
obj2.name = 'adf'
var obj2 = Object.assign({}, obj)
//its partial deep copy - obj.name will not effect but obj.skills.primary will effect with the below line
obj2.skills.primary= 'addaf'
var obj2 = JSON.parse(JSON.stringify(obj))
// its more than deep copy - with this it will not effect any thing - but can not copy the function from obj to obj2 for that we have to install lodash (npm install lodash)
var _ = require('lodash');
var obj2 = _.cloneDeep(obj) // this is perfect deep copy
```

- Copying means initiating a new variable with the same value(s).
- Deep copying means that all of the values of the new variable are copied and disconnected from the original variable.
- In deep copy, a new memory allocation happens for the other entities, and reference is not copied to the other entities.

Palindrome Code:-

```
function checkPalindrome(str){
  var rev='';
  for(i=str.length-1;i>=0;i--){
    rev+=str[i]
  }
  if(rev==str)
  {
    console.log("palindrome")
  }
  else
  {
    console.log("it's not a palindrome")
  }
}
checkPalindrome('madam')
```

node /tmp/hqcZXG0k2A.js
palindrome

Remove Duplicate elements from an array:

```
main.js  [ ] [ ] Run Output
1 function removeDuplicates(arr){
2   var out=[];
3
4   for(var i=0;i<arr.length;i++)
5   {
6     if(!out.includes(arr[i]))
7     {
8       out.push(arr[i])
9     }
10  }
11  return out;
12
13 }
14 var result=removeDuplicates([1,2,3,4,5,6,6,3,1,2])
15 console.log(result)
```

node /tmp/hqcZXG0k2A.js
[1, 2, 3, 4, 5, 6]

Group by Age:

```

function groupByAge(students){
    var out={};
    for(var i=0;i<students.length;i++)
    {
        if(!out[students[i].age])
        {
            out[students[i].age]=[]
        }
        out[students[i].age].push(students[i])
    }
    return out;
}

var data=[{name:'chandana',age:25},
{name:'naveen',age:26},
{name:'chy',age:25},
{name:'chs',age:25}]
var result=groupByAge(data)
console.log(result)

```

```

node /tmp/hqcZXG0k2A.js
{
  '25': [
    { name: 'chandana', age: 25 },
    { name: 'chy', age: 25 },
    { name: 'chs', age: 25 }
  ],
  '26': [ { name: 'naveen', age: 26 } ]
}

```

```

function groupByAge(students,groupByType){
    var out={};
    for(var i=0;i<students.length;i++)
    {
        if(!out[students[i][groupByType]])
        {
            out[students[i][groupByType]]=[]
        }
        out[students[i][groupByType]].push(students[i])
    }
    return out;
}

var data=[{name:'chandana',age:25},
{name:'naveen',age:26},
{name:'chy',age:25},
{name:'chy',age:25}]
var result=groupByAge(data,'name')
console.log(result)

```

```

node /tmp/hqcZXG0k2A.js
{
  chandana: [ { name: 'chandana', age: 25 } ],
  naveen: [ { name: 'naveen', age: 26 } ],
  chy: [ { name: 'chy', age: 25 }, { name: 'chy', age: 25 } ]
}

```

Sum of pairs:

Missing Numbers:

main.js	Output
<pre>1 var arr=[1,2,4,9,30,24,25] 2 var max=Math.max(...arr) 3 console.log(max) 4 var min=Math.min(...arr) 5 console.log(min) 6 7 for(var i=min;i<max;i++) 8 { 9 if(!arr.includes(i)) 10 { 11 arr.push(i) 12 } 13 } 14 console.log(arr) 15</pre>	<pre>node /tmp/KUG2H75aGc.js 30 1 [1, 2, 4, 9, 30, 24, 25, 3, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 26, 27, 28, 29]</pre>

Largest Number From An Array:

<pre>var arr=[1,3,4,678,8,990,0076,65] var largest=arr[0] for(var i=0;i<arr.length;i++) { if(arr[i]>largest) { largest=arr[i] } } console.log(largest)</pre>	<pre>node /tmp/w0aw75aFQF.js 990</pre>
---	--

Smallest Number From An Array:

<pre>1 var arr=[1,3,4,678,8,990,0076,65] 2 var Smallest=arr[0] 3 4 for(var i=0;i<arr.length;i++) 5 { 6 if(arr[i]<Smallest) 7 { 8 Smallest=arr[i] 9 } 10 } 11 12 console.log(Smallest)</pre>	<pre>node /tmp/w0aw75aFQF.js 1</pre>
---	--------------------------------------




Find the frequency of elements in array


main.js	Output
<pre>1 2 const myArray =["hello","java","hello","world","world"]; 3 4 const frequency = {}; 5 6 for (let i = 0; i < myArray.length; i++) { 7 const element = myArray[i]; 8 9 if (frequency[element]) 10 { 11 frequency[element] += 1; 12 } 13 else 14 { 15 frequency[element] = 1; 16 } 17 } 18 console.log(frequency)</pre>	<pre>node /tmp/egRVKxNNzv.js { hello: 2, java: 1, world: 2 }</pre>

Collect books from array of objects and return collection of books as an array

main.js	Output
<pre>1 const booksArray = [2 { title: ["Book1","bhvbn"], author: "Author1" }, 3 { title: ["Book2","dfgh"], author: "Author2" }, 4 { title: ["Book3","fgh"], author: "Author3" }, 5]; 6 var ot=[] 7 8 for(var i=0;i<booksArray.length;i++) 9 { 10 ot.push(...booksArray[i].title) 11 } 12 console.log(ot) 13</pre>	<pre>node /tmp/ST0otSwzLE.js ['Book1', 'bhvbn', 'Book2', 'dfgh', 'Book3', 'fgh']</pre>

Print all duplicate elements of an array

main.js	  	Output
<pre>1 function removeDuplicates(arr){ 2 var out=[] 3 for (let i = 0; i < arr.length; i++) { 4 for (let j = i+1; j < arr.length; j++) { 5 if (arr[i] == arr[j]) { 6 7 if (out.includes(arr[i])) { 8 break; 9 } 10 11 else { 12 out.push(arr[i]); 13 } 14 } 15 } 16 } 17 } 18 return out 19 } 20 let result=removeDuplicates([1,1,2,2,3,3,4,5,6,7,8,8]) 21 console.log(result)</pre>		<pre>node /tmp/VksH5JP6Br.js [1, 2, 3, 8]</pre>



Coding Course,
Learn javascript the
power of AI to aid yo

Fibonacci Series:

flattenArray

[JavaScript Online Compiler](#)

Sponsored by:

Up to 70% off - Reliab...

Zeelool | Stylish Prescription Glasses, Affordable Eyeglasses online. 2023 trendy new glasses, up...

main.js

Run

```
1 const array = [[1, 2, 3], [4, 5], [6, 7, [8, 233,[67,8909,898]]]];
2 function flattenArray(arrayOfArrays){
3   let out=[];
4
5   for(var i=0;i<arrayOfArrays.length;i++)
6   {
7     let element=arrayOfArrays[i];
8     if(Array.isArray(element))
9     {
10      out.push(...flattenArray(element));
11    }
12    else
13    {
14      out.push(element)
15    }
16  }
17  return out
18 }
19 let result=flattenArray(array)
20 console.log(result)
21
```

Output

node /tmp/XIwkQ9VKZB.js

[
 1, 2, 3, 4, 5,
 6, 7, 8, 233, 67,
 8909, 898
]