Sr. Software Engineer (SSE)

ABSTRACT

This is one of the subject from my personal notes series named “Coding-With-Arqam” that I am developing from the start of my professional development career.

Subject

Angular

ANGULAR

Angular (6-9), Material, TypeScript

--> Commands:

-> npm install -g @angular/cli@latest

-> ng new App-Name

-> cd App-Name

-> ng serve // to run angular project like npm run

-> ng serve --open // to run angular project like npm run

-> ng generate component component_name // to generate component

-> ng g c component_name // to generate component

-> ng generate service service_name // do not need to mention the "service" word. It creates like service_name.service.ts

-> ng build --prod / ng build // for taking the build to deploy

--> Notes:

-> TypeScript is a superset of javascript. Addition to jS

-> Flow: main.ts -> app.module.ts -> AppComponent (app.component.ts + html)

-> Decorators are always added with @ sign

-> Decorators are a design pattern that is used to separate modification or decoration of a class without modifying the original source code.

-> In AngularJS, decorators are functions that allow a service, directive or filter to be modified prior to its usage.

-> npm cache clean --force

-> Angular (start-9) is a javascript framework for client-side work.

-> Angular also supports workspaces with multiple projects.

-> BootstrapCDN = live bootstrap by including link.

-> If we have installed it in the local project, we have to include it in angular.json file.

-> The height can be in px, em, or rem. If no units are specified, px units are assumed.

*-> [ngIf] = *ngIf*

-> Material Angular uses reactive forms when we use "FormControls", "FormBuilder", "FormGroup", "Validators".

-> Visual Studio Code Formators: Beautify, Prettier, etc.

-> We use "change" event in select and "Click" event in buttons, etc.

--> Angular JS VS Angular:

-> Architecture:

-> MVC design => Components and directives.

-> Language:

-> Written in JavaScript => Microsoft's Typescript which is superset of ECMAScript 6 (ES6)

-> Mobile support:

-> Does not support => Supports.

-> Routing:

-> $routerprovider.when() => @RouteConfig{()}

-> Dependency Injection:

-> Does not use => Uses hierarchical dependency injection system with unidirectional tree-based change detection.

-> Structure:

-> Less manageable => Better structure, easy to manage and maintain

-> Speed:

-> Reduced the development effort and time due to two-way binding => Angular 4 is the fastest version yet

--> Abbreviations:

-> e2e: end to end testing

--> Routes And Paths:

-> Routes are definitions (objects) comprised from at least a path and a component (or a redirectTo path) attributes.

-> Path = Part of the URL that determines a unique view that should be displayed.

-> Component = Angular component that needs to be associated with a path.

--> Component:

-> Just a class that serves as a controller for the user interface.

-> It consists of three parts (files): some TypeScript code, an HTML template, and CSS/scss styles.

-> Providers:

-> A component can contain a list of providers the component and its children may inject.

-> An object declared to Angular so that it can be injected in the constructor of your components, directives and other classes instantiated by Angular

-> A provider is an instruction to the Dependency Injection system on how to obtain a value for a dependency.

-> Most of the time, these dependencies are services that you create and provide.

-> E.g: ng generate service User.

--> Services:

-> Singleton objects.

-> Get instantiated only once during the lifetime of an application.

-> Organize and share business logic, models, or data and functions with different components of an Angular application.

-> A class with a narrow, well-defined purpose.

-> Angular distinguishes components from services to increase modularity and reusability.

-> Code to be used everywhere on the page.

-> It can be for data connection that needs to be shared across components, etc.

-> With services, we can access methods and properties across other components in the entire project.

-> Like myservice.service.ts

-> ng g service myservice

-> The ngOnInit function gets called by default in any component created.

-> Like: this.todaydate = this.myservice.showTodayDate(); // getting date from service

--> Http Service:

-> Help us fetch external data, post to it, etc.

-> We need to import the http module to make use of the http service.

-> E.g: (in app.component.ts file)

-> import { Http } from '@angular/http';

-> this.http.get("http://jsonplaceholder.typicode.com/users").map((response) ⇒ response.json()).subscribe((data) ⇒ console.log(data))

--> Http Client:

-> Introduced in Angular 6.

-> Help us fetch external data, post to it, etc.

-> We need to import the http module to make use of the http service.

--> Http Service vs Http Client:

->

--> Versions:

-> AngularJS = 2010

-> Angular2 = 2016

-> Angular4 = 2017

-> Angular2 is completely rewritten from the ground up

-> Angular2 is five times faster than the angualrJS

-> AngularJS was not written for mobile devices

-> We have more languages with Angular2 like JS, TS, Dart, PureScript, Elm, etc

-> Updated verisions shows compatibility with latest versions of the typescript

*-> new if-else style -> *ngIf*

--> Toasts:

-> A simple Pop-up component

-> Notifications.

-> Animated notification pop-up that is small and nonblocking.

--> Bower.JSON:

-> The simple example above shows a bower. json file which defines some information about the projects as well as a list of dependencies. The bower. json file is actually used to define a Bower package, so in effect you're creating your own package that contains all of the dependencies for your application.

--> CSS:

-> SCSS = https://sass-lang.com/documentation/syntax#scss

-> Sass = https://sass-lang.com/documentation/syntax#the-indented-syntax

-> Less = http://lesscss.org

-> Stylus = http://stylus-lang.com

--> Router-outlet:

-> Works as a placeholder which is used to load the different components dynamically based on the activated component or current route state.

-> Navigation can be done using router-outlet directive and the activated component will take place inside the router-outlet to load its content.

--> Forms Types:

-> Template Driven form:

-> use the FormsModule, asynchronous

-> most of the logic is driven from the template

-> most of the work is done in the template

-> import { FormsModule } from '@angular/forms'; which is done in app.module.ts

-> In template driven forms, we need to create the model form controls by adding the ngModel directive and the name attribute

-> validations like "required", "pattern" is in html

-> Reactive forms/Model Driven Form:

-> Use the "ReactiveFormsModule", synchronous

-> The logic resides mainly in the component or typescript code.

-> Rctive forms are more suitable because we can define validations and model from component

-> Primarily it is also called "Model Driven Forms"

-> import { FormBuilder, FormGroup, Validators ,FormsModule,NgForm } from '@angular/forms'; // to use reactive form

--> Commands:

-> ng g <schematic> [options]

-> Schematic = appShell, application, class, component, directive, enum, guard, interface, library, module, pipe, service, serviceWorker, universal, webWorker

-> ng build --prod / npm run ng build --prod

--> Steps to delete a component in Angular:

-> Remove the import line reference from Angular app.module.ts file.

-> Remove the component declaration from @NgModule declaration array in app.module.ts file

-> And then manually delete the component folder from Angular project.

-> Finally Delete all the references of component manually from the Angular project.

--> Directive:

-> JS class

-> Declared as @directive

-> Allows you to attach a behavior to DOM elements

*-> E.g: *ngFor and *ngIf (built-in)*

-> Types:

-> Component Directives:

-> These form the main class having details of how the component should be processed, instantiated and used at runtime

-> Structural Directives:

-> A structure directive basically deals with manipulating the dom elements

*-> Structural directives have a * sign before the directive*

*-> For example, *ngIf and *ngFor.*

-> Attribute Directives:

-> Attribute directives deal with changing the look and behavior of the dom element

-> You can create your own directives like "ng g directive changeText"

--> Pipes:

-> Do formating of the data before displaying in the View

-> Pipe is used by using |

-> This symbol is called a Pipe Operator

-> It takes integers, strings, arrays, and date

-> E.g:

-> {{ Welcome to Angular 6 | lowercase}}

-> {{title | lowercase}}

-> {{6589.23 | currency:"USD"}}

-> {{todaydate | date:'d/M/y'}}

--> Routing:

-> Navigating between pages.

-> E.g: <a routerLink = "new-cmp">New component</a>

--> Module:

-> Refers to a place where you can group the components, directives, pipes, and services, which are related to the application.

-> To define module, we can use the NgModule.

-> When you create a new project using the Angular -cli command, the ngmodule is created in the app.module.ts file by default

-> Declaration:

-> Array of components created.

-> If any new component gets created, it will be imported first and the reference will be included in declarations.

-> Import:

-> Array of modules required to be used in the application.

-> It can also be used by the components in the Declaration array like imported "BrowserModule" included in declaration array.

--> Data Binding:

-> communication b/w typescript code (bussiness logics) and html

-> One-way Data Binding:

-> Uni directional.

-> From Component to View (Data Binding):

-> Typesript code -> string interpolation/ Property binding -> Template(HTML)

-> Use curly braces for data binding - {{}};

-> this process is called interpolation.

-> The variable in the app.component.html file is referred as {{title}} and the value of title is initialized in the app.component.ts file and the value is displayed in app.component.html.

-> From View to Component (Event Binding):

-> Template(HTML) -> event binding -> Typesript code

-> When a user interacts with an application in the form of a keyboard movement, a mouse click, or a mouseover, it generates an event.

-> E.g: (click)="myClickFunction($event)"

-> Two-way Data Binding:

-> Bi-direction

-> Can be achieved using a ngModel directive.

-> Import the FormsModule from @angular/forms in parentModule.module.ts

-> If you do not import the FormsModule, then you will get Template parse errors

-> Example:

-> <input type="text" [(ngModel)] = 'val' /> // where val is the variable in ts file.

--> ng-template:

-> ng-template is an Angular element used to render HTML templates.

*-> We use ng-template with angular *ngIf directive to display else template.*

-> ng-template is a virtual element and its contents are displayed only when needed (based on conditions).

-> ng-template should be used along with structural directives like [ngIf],[ngFor],[NgSwitch] or custom structural directives.That is why in the above example the contents of ng-template are not displayed.

->

-> E.g:

*-> <span *ngIf = "isavailable;then condition1 else condition2">Condition is valid.</span>*

<ng-template #condition1>Condition is valid from template</ng-template>

<ng-template #condition2>Condition is invalid from template</ng-template>

--> Snack Bar:

-> just a pop up that shows from below of the screen

--> Promise VS RxJS Observable:

-> A Promise handles a single event when an async operation completes or fails.

-> Observable provides operators like map, forEach, reduce, similar to an array.

-> Obervables are preferrable over promise because it provides multiple functions like subscribe(), map(), filter(), etc

-> Cancellation is very difficult in promise and ES6 doesn't supports cancellation in promise.

--> Data sharing b/w the components:

-> Passing the reference of one component to another

-> Communication through parent component

-> Communication through Service

-> @Input, @Output and EventEmitter:

--> Decorators in angular:

-> Decorators are a design pattern that is used to separate modification or decoration of a class without modifying the original source code. In AngularJS, decorators are functions that allow a service, directive or filter to be modified prior to its usage.

-> @injectable etc are decarators.

--> Dependency Injection(DI):

-> A way to create objects that depend on the other objects

-> A Dependency Injection system supplies the dependent objects (called the dependencies) when it creates an instance of an object

--> Redux:

-> To use Redux in the Angular framework, we can use the NgRx library.

-> This is a reactive state management library. With NgRx, we can get all events (data) from the Angular app and put them all in the same place (Store).

-> why should we use Redux in an Angular application instead of a shared service for example?

-> Angular:

-> We can use a service to share data between components

-> We can use the Input/Output

-> We can also use ViewChild for nested components

-> All above angular data sharing services increase the complexity in large projects

-> If we have a large number of components,

we risk losing control over the data flow within a component (where did this data come from and what is its intended destination?)

-> The store and the unidirectional(redux) data flow reduce the complexity of the application.

-> The flow is more clear and easy to understand for new team members

--> Angular 6 VS Angular 7 VS Angular 8 VS Angular 9:

-> Amgular 6:

-> TypeScript 2.7 support.

-> ng-add.

-> ng-update

-> Angular 7:

-> TypeScript 3.1 support.

-> Virtual scrolling.

-> Drag & Drop.

-> Angular 8: (Most widely used)

-> TypeScript 3.4 support.

-> Lazy loading.

-> Angular Firebase (Support for official firebase).

-> Differential loading.

-> Angular 9:

-> TypeScript 3.7 support.

-> More reliable ng-update.

-> The AOT builds will be noticeably faster ensuring a significant change in the compiler's performance.

--> Angular VS Vue VS React:

-> Can be used with any backend programming language like PHP, Java etc.

-> Vue JS is strictly front-end based and uses HTML, CSS and JS separately.

-> Angular is Structured while vue.js modular and flexible

-> vue is more lightweight

-> Angular is more enterprise ready for developing complex applications

-> Angular's main drawback is its size, startup time, and memory allocation capacity compared to Vue.

-> model binding: ng-model (angular) and v-mdoel (vue.js)

-> Vue.js is a JavaScript library for building web interfaces

-> Google and Wix (Angular).

-> Whatsapp, Instagram Paypal, Glassdoor, BBC, Facebook (React)

-> GitLab and Alibaba (Vue).

-> React is mostly used in high traffic websites.

-> React was developed when Facebook ads started gaining traffic and faced problems in their coding and maintenance which depicted certain issues

-> Vue.js was created by an ex-engineer of Google, Evan You.

-> Angular has a lot to offer to its developers from templates to testing utilities which increases the size.

-> Angular is unsuitable for the light-weight applications.

-> React is suitable for light-weight apps becoz it is not a framework like angular.

-> This is why it needs support from other libraries for tasks like routing and all.

-> Vue is the smallest of other frameworks and libraries hence extremely suitable for the development of lightweight applications

-> Summary:

-> Angular and React has a strong community base with backing from top companies like Facebook and Google.

But still, vue is popular in the open-source community.

--> TSLint:

-> An extensible linter (machine which filters) for the TypeScript language.

-> Warning: It has been deprecated as of 2019.

-> Extensible static analysis tool that checks TypeScript code for readability, maintainability, and functionality errors.

--> Sharing Data Between the Components:

-> *Parent to Child - Sharing Data via Input:*

-> *Child to Parent - Sharing Data via ViewChild:*

-> *Child to Parent - Sharing Data via Output() and EventEmitter:*

-> *Unrelated Components - Sharing Data with a Service:*

--> *Route Guards:*

-> *Note: Never use [AuthGuardService] with app route (basic or default route when no path is given).*

-> *NOTE: User providers in injectable as "@Injectable({ providedIn: 'root' })".*

-> *There are five types of interfaces in route guard:*

-> *CanActivate*

-> *CanActivateChild*

-> *CanDeactivate*

-> *Resolve*

-> *CanLoad*

-> *CanActivate:*

-> *Scenario:*

-> *Let we want to check that if a user is admin, he has allowed to get second component. And if not and admin, he will not be able to get second component. And despite showing empty screen, we will move that user to third component.*

-> *We will get user from AuthService (having login, logout, forgot pass, etc services).*

-> *Solution:*

-> *We will import CanActivate and other route guards(if needed) from angular/core.*

-> *We will then implements (inherit) our AuthGuardService with these auth guards.*

-> *in CanActivate function, we will impolement the logic of checking the user and perform the action accordingly.*

-> *Code:*

```
->   canActivate(
        next: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean {
        // Logic ...................
        }
```

--> *Errors:*

-> *Experimental support for decorators is a feature that is subject to change in a future release. Set the 'experimentalDecorators' option in your 'tsconfig' or 'jsconfig' to remove this warning:*

-> *This warning occurs when we create the service. This is due to VS-Code.*

-> *Add this line in seeting.json file of VS-Code. Path is File->preferences->setting->setting.json*

-> *Line: "javascript.implicitProjectConfig.experimentalDecorators": true*

--> *Material Angular:*

-> *Notes:*

-> *ng add @angular/material*

-> *Material angular doesn't siupports input fields of type "file". We need to use external libraries like "npm i angular-material-fileupload"*

--> *Modal Dialogue:*

-> *Modal dialogs are great for when you need to restrict a user to a particular action before they can return to the normal use of the application. ... Say, the user clicks the logout button in your* *application but, instead of logging them out right away, you want to confirm that's what the user is trying to do.*

TypeScript

--> General Points:

-> TypeScript is an open-source programming language developed and maintained by Microsoft.

-> TypeScript is a primary language for Angular application development.

-> It is a superset of JavaScript with design-time support for type safety and tooling.

-> Browsers can't execute TypeScript directly.

-> Typescript must be "transpiled" into JavaScript using the tsc compiler, which requires some configuration.

-> tsconfig.json:

-> TypeScript compiler configuration.

-> Guides the compiler as it generates JavaScript files for a project.

-> This file contains options and flags like baseUrl, etc that are essential for Angular applications.

-> Typically, the file is found at the root level of the workspace.

-> It can be run on Node js or any browser which supports ECMAScript 3 or newer versions.

-> Typescript provides optional static typing, classes, and interface.

-> New way of writing Js.

-> Open-source pure object-oriented programing language.

--> Features:

-> OOP language

-> Supports JS libraries

-> JS is Typescript

-> Portable

-> DOM Manipulation

-> TS is just a Js

--> Execution Cycle:

-> app.ts => tsc app.ts => app.js

--> Text Editors:

-> VS Code, Sublime Text, WebStorm, Vim, Atom, etc.

--> Versions:

-> 0.8 - 3.5

--> JavaScript VS TypeScript:

-> Doesn't support strongly typed or static typing => Supports

-> .js => .ts

-> It is directly run on the browser => Not

-> Just a scripting language => Supports object-oriented programming concept like classes, interfaces, inheritance, generics, etc.

-> Doesn't support modules => Supports

-> Number, string are the objects => Number, string are the Interface

--> Types:

- *-> Static: "at compile time" or "without running a program."*
  - *-> Built-in / Primitive Type:*
    - *-> Numbers, Void, String, Null, Boolean*
    - *-> Example: let first: number = 12.0;*
  - *-> User-defined:*
    - *-> Array, Class, Tuple, Enum, Interface, Fucntions*
    - *-> Enums:*
      - *-> TypeScript gets support for enums from ES6.*
      - *-> Enums define a set of named constant.*
      - *-> Provides both string-based and numeric-based enums.*
      - *-> Example:*

```
-> enum Color {
                    Red, Green, Blue
   };
   let c: Color;
   ColorColor = Color.Green;
```

- *-> Generics:*
  - *-> A tool which provides a way to create reusable components.*
  - *-> It creates a component that can work with a variety of data types rather than a single data type.*
  - *-> Generics use a special kind of type variable <T> that denotes types.*
  - *-> The generics collections contain only similar types of objects.*
- *-> Decorators:*
  - *-> Special kind of declaration that can be applied to classes, methods, accessor, property, or parameter.*
  - *-> TypeScript Decorators serves the purpose of adding both annotations and metadata to the existing code in a declarative way.*
  - *-> Example:*
    - *-> @NgModule*
    - *-> @Component*
    - *-> @Injectable*
    - *-> @Directive*
    - *-> @Pipe*
    - *-> @Input*
    - *-> @Output*
    - *-> @HostBinding*
    - *-> @HostListener*
    - *-> @ContentChild*
    - *-> @ContentChildren*
    - *-> @ViewChild*
    - *-> @ViewChildren*

  - *-> Types:*
    - *-> Class Decorators:*
      - *-> Just before the class declaration, and it tells about the class behaviors.*
      - *-> A class decorator is applied to the constructor of the class.*
      - *-> A class decorator can be used to observe, modify, or replace a class definition.*
      - *-> Example:*
        - *-> @sealed*

class Person { .. code here ..}

-> In the above example, when @sealed decorator is executed, it will seal both the constructor and its prototype so that we cannot inherit the Person class.

-> Method Decorators:

-> Defined just before a method declaration.

-> It is applied to a property descriptor for the method.

-> It can be used to observe, modify, or replace a method definition.

-> We cannot use method decorator in a declaration file.

-> Example:

```
-> @log
    Add(item: string): void {
        this.itemArr.push(item);
    }
```

-> In the above example, the @log decorator will log the new item entry.

-> Accessor Decorators:

-> Defined just before an accessor declaration.

-> Note: An accessor is a getter and setter property of the class declaration.

-> Example:

```
-> @configurable(false)
    get salary() { return 'Rs. ${this._salary}'; }
    set salary(salary: any) { this._salary = +salary; }
```

-> Property Decorators:

-> Defined just before a property declaration.

-> It is similar to the method decorators.

-> The only difference between property decorators and method decorators is that they do not accept property descriptor as an argument and do not return anything.

-> Parameter Decorators:

-> Defined just before a parameter declaration.

-> It is applied to the function for a class constructor or method declaration.

-> Example:

```
@validate
show(@required name: string) {
    return "Hello " + name + ", " + this.msg;
}
```

--> Type Inference:

-> Store a collection of values of different data types in a single variable.

--> Tuples:

-> Arrays will not provide this feature, but TypeScript has a data type called Tuple to achieve this purpose.

-> Example: let arrTuple = [101, "JavaTpoint", 105, "Abhishek"];

--> String:

-> let var_name = new String(string); OR let var_name: "Ali";

-> Types:

-> Single quoted

-> Double quoted

-> Back-ticks quoted:

-> It is also known as Template string.

-> It is used to write an expression.

-> We can use it to embed the expressions inside the string.

-> Multi-line

--> Numbers:

-> properties:

-> MAX_VALUE, MIN_VALUE, NEGATIVE_INFINITY, POSITIVE_INFINITY, NaN, prototype.

--> Loops:

-> Indefinite:

-> While, do-while

-> Definite:

-> for, for-of, for-in

--> Enums:

-> Enums stands for Enumerations

-> Enums are a new data type supported in TypeScript.

-> It is used to define the set of named constants, i.e., a collection of related values.

-> Types:

-> Numeric:

-> Example:

```
-> enum Direction {
        Up = 1,
        Down,
        Left,
        Right,
}
```

we initialize Up with 1, and all of the following members are auto-incremented from that point.

-> String:

-> Hetrogeneous:

--> Map function:

-> Added in ES6 version of JavaScript.

-> It allows us to store data in a key-value pair

-> Methods:

-> set, get, has, delete, size, clear

--> Set:

-> Added in ES6 version of JavaScript.

-> It allows us to store distinct data (each value occur only once) into the List similar to other programming languages.

--> Accessor:

-> Getter

-> Setter

--> Interface:

-> It defines the syntax for classes to follow, means a class which implements an interface is bound to implement all its members.

-> We cannot instantiate the interface, but it can be referenced by the class object that implements it.

-> The interface contains only the declaration of the methods and fields, but not the implementation.

--> Namespaces:

-> Way which is used for logical grouping of functionalities.

-> It encapsulates the features and objects that share common relationships. It allows us to organize our code in a much cleaner way.

-> A namespace is also known as internal modules.

-> A namespace can also include interfaces, classes, functions, and variables

--> tsconfig.json:

-> The tsconfig.json file is a file which is in JSON format.

-> In the tsconfig.json file, we can specify various options which tell the compiler how to compile the current project.

->

--

Material Angular

--

--> General Points:

-> UI/UX components in Angular, are known as Angular Materials.

--> Modules:

-> AutoComplete:

-> Suggests relevant options as the user types.

-> Example:

```
<mat-autocomplete #auto="matAutocomplete">
    <mat-option *ngFor="let option of options" [value]="option">
        {{option}}
    </mat-option>
</mat-autocomplete>
```

-> Badge:

-> A small value indicator that can be overlaid on another object.

-> Example:

```
<span matBadge="4" matBadgeOverlap="false">Text with a badge</span>
```

-> Bottom Sheet:

-> These panels are intended primarily as an interaction on mobile devices where they can be used as an alternative to dialogs and menus

-> Botton:

-> mat-button

-> mat-raised-button

-> mat-flat-button

-> mat-stroked-button

-> mat-icon-button

-> mat-fab

-> mat-mini-fab

-> Button Toggle:

-> A groupable on/off toggle for enabling and disabling options.

-> Example:

```
<mat-button-toggle-group name="fontStyle" aria-label="Font Style">
        <mat-button-toggle value="bold">Bold</mat-button-toggle>
        <mat-button-toggle value="italic">Italic</mat-button-toggle>
        <mat-button-toggle value="underline">Underline</mat-button-toggle>
</mat-button-toggle-group>
```

-> Card:

-> A styled container for pieces of iutemized content.

-> Example:

```
<mat-card>Simple card</mat-card>
```

-> CheckBox:

-> Capture boolean input with an optional intermediate mode.

-> Example:

https://material.angular.io/components/checkbox/overview

-> Chips:

-> Presents a list of items as a set of small, tactile entities.

-> Example:

```
<mat-chip-list aria-label="Fish selection">
        <mat-chip>Static option 1</mat-chip>
        <mat-chip *ngFor="let fruit of fruits" [selectable]="selectable"
                        [removable]="removable" (removed)="remove(fruit)">
                {{fruit.name}}
                <mat-icon matChipRemove *ngIf="removable">cancel</mat-icon>
        </mat-chip>
        <input placeholder="New fruit..."
                [matChipInputFor]="chipList"
                [matChipInputSeparatorKeyCodes]="separatorKeysCodes"
                [matChipInputAddOnBlur]="addOnBlur"
                (matChipInputTokenEnd)="add($event)">
</mat-chip-list>
```

-> DatePicker:

-> Captures dates, agnostic about their internal presentation.

-> Example:

```
<mat-form-field appearance="fill">
        <mat-label>Choose a date</mat-label>
        <input matInput [matDatepicker]="picker">
        <mat-datepicker-toggle matSuffix [for]="picker"></mat-datepicker-toggle>
        <mat-datepicker #picker></mat-datepicker>
</mat-form-field>
```

-> Dialog:

-> A configurable modal that displays dynamic content.

-> Modal dialogs are great for when you need to restrict a user to a particular action before they can return to the normal use of the application

-> Like logout modal dialog.

-> import {MatDialog, MatDialogRef, MAT_DIALOG_DATA} from '@angular/material/dialog';

-> Example:

https://material.angular.io/components/dialog/overview

-> Divider:

-> A veritical or horizontal visual divider.

-> Example:

```
<mat-list>
    <mat-list-item>Item 1</mat-list-item>
    <mat-divider></mat-divider>
    <mat-list-item>Item 2</mat-list-item>
    <mat-divider></mat-divider>
    <mat-list-item>Item 3</mat-list-item>
</mat-list>
```

-> Expansion Panel:

-> A container which can be expanded to reveal more content.

```
-> <mat-accordion>
        <mat-expansion-panel hideToggle>
        .............
        </mat-expansion-panel hideToggle>
    </mat-accordion>
```

-> Example:

https://material.angular.io/components/expansion/overview

-> Form Field:

-> Wraps input fields so they are displyed consistently.

-> Example:

```
<mat-form-field appearance="outline">
    <mat-label>Outline form field</mat-label>
    <input matInput placeholder="Placeholder">
    <mat-icon matSuffix>sentiment_very_satisfied</mat-icon>
    <mat-hint>Hint</mat-hint>
</mat-form-field>
```

-> Example:

```
mat-form-field appearance="fill">
    <mat-label>Enter your password</mat-label>
    <input matInput [type]="hide ? 'password' : 'text'">
    <button mat-icon-button matSuffix (click)="hide = !hide" [attr.aria-label]="'Hide password'" [attr.aria-pressed]="hide">
    <mat-icon>{{hide ? 'visibility_off' : 'visibility'}}</mat-icon>
    </button>
</mat-form-field>
```

-> Drid List:

-> A flexible structure for presenting content items in a grid.

-> Example:

```
<mat-grid-list cols="2" rowHeight="2:1">
    <mat-grid-tile>1</mat-grid-tile>
    <mat-grid-tile>2</mat-grid-tile>
</mat-grid-list>
```

-> Icon:

-> Renders a specified icon.

-> Example:

```
<mat-icon aria-hidden="false" aria-label="Example home icon">home</mat-icon>
```

-> Input:

-> Enables native inputs to be used within a Form field.

-> Example:

```
<input matInput placeholder="Ex. Pizza" value="Sushi">
<textarea matInput placeholder="Ex. It makes me feel..."></textarea>
```

-> List:

-> Presents convetional list of items.

-> Example:

-> Simple lists:

```
<mat-list>
    <mat-list-item> Pepper </mat-list-item>
    <mat-list-item> Salt </mat-list-item>
</mat-list>
```

-> Navigation lists:

```
<mat-nav-list>
    <a mat-list-item href="..." *ngFor="let link of links"> {{ link }} </a>
</mat-nav-list>
```

-> Action lists:

```
<mat-action-list>
    <button mat-list-item (click)="save()"> Save </button>
    <button mat-list-item (click)="undo()"> Undo </button>
</mat-action-list>
```

-> Selection lists

-> Multi-line lists

-> Lists with icons

-> Lists with avatars

-> Dense lists

-> Lists with multiple sections

-> Menu:

-> A floating panle of nestable items.

-> Like three dots mostly in the mobile application.

-> Example:

<button mat-button [matMenuTriggerFor]="menu">Menu</button>

<mat-menu #menu="matMenu">

<button mat-menu-item>Item 1</button>

<button mat-menu-item>Item 2</button>

</mat-menu>

-> Paginator:

-> Controls for displaying paged data.

-> Example:

<mat-paginator [length]="100"

[pageSize]="10"

[pageSizeOptions]="[5, 10, 25, 100]">

</mat-paginator>

-> Progress bar:

-> A linear progress indicator.

-> Example:

-> Determinate:

<mat-progress-bar mode="determinate" value="40"></mat-progress-bar>

-> Indeterminate:

<mat-progress-bar mode="indeterminate"></mat-progress-bar>

-> Buffer:

-> Query:

-> Progress spinner(Loader):

-> A circular progress indicator.

-> Example:

<mat-spinner></mat-spinner>

-> Radio Button:

-> Allows the user to select one option from a group.

-> Example:

<mat-radio-group aria-label="Select an option">

<mat-radio-button value="1">Option 1</mat-radio-button>

<mat-radio-button value="2">Option 2</mat-radio-button>

</mat-radio-group>

-> Ripples:

-> Directive for adding Material Design ripple effects.

-> Example:

<div matRipple [matRippleColor]="myColor">

<ng-content></ng-content>

</div>

-> Select:

-> Allows the user to select one or more options using a dropdown.

-> Example:

<mat-select>

*<mat-option *ngFor="let food of foods" [value]="food.value">*

{{food.viewValue}}

</mat-option>

</mat-select>

-> Sidenav:

-> A container for content that is fixed to one side of the screen.

-> Example:

https://material.angular.io/components/sidenav/overview

-> Side toggle:

-> Captures boolean values as a clickable and draggable switch.

-> Example:

<mat-slide-toggle>Slide me!</mat-slide-toggle>

-> Slider:

-> Allows the user to input a value by dragging along a slider.

-> Example:

<mat-slider min="1" max="5" step="0.5" value="1.5"></mat-slider>

-> Snackbar:

-> Displays short actionable messages as an uninvasive alert.

-> Example:

<button mat-stroked-button (click)="openSnackBar(message.value, action.value)">Show snack-bar</button>

-> Sort header:

-> Allows the user to configure how tabular data is sorted.

-> Example:

https://material.angular.io/components/sort/overview

-> Stepper:

-> Presents content as steps through which to progress.

-> Example:

https://material.angular.io/components/stepper/overview

-> Table:

-> A configurable component for displaying tabular data.

-> Example:

https://material.angular.io/components/table/overview

-> Tabs:

-> Only presents one view at a time from a provided set of views.

-> Lazy Loading can also apply.

-> Example:

<mat-tab-group>

<mat-tab label="First"> Content 1 </mat-tab>

<mat-tab label="Second"> Content 2 </mat-tab>

<mat-tab label="Third"> Content 3 </mat-tab>

</mat-tab-group>

-> Toolbar:

-> A container for top-level titles and controls.

-> Example:

```
<mat-toolbar>
    <span>My Application</span>
</mat-toolbar>
```

-> Tooltip:

-> Displays floating content when an object is hovered.

-> Example:

```
<button mat-raised-button matTooltip="Info about the action" aria-label="Button that displays a tooltip when focused or hovered over">
    Action
</button>
```

-> Tree:

-> Presents hierarchical content as an expandable tree.

-> Example:

```
<mat-tree>
    <mat-tree-node> parent node </mat-tree-node>
    <mat-tree-node> -- child node1 </mat-tree-node>
    <mat-tree-node> -- child node2 </mat-tree-node>
</mat-tree>
```

-> Example:

https://material.angular.io/components/tree/overview

Reference Links

- *https://angular.io/start*
- *https://www.angularjswiki.com/angular/how-to-delete-a-component-in-angular/*
- *https://getbootstrap.com/*
- *https://bootsnipp.com/snippets/nNX3a*
- *https://stackblitz.com/edit/angular-o9wwhh?file=src/app/top-bar/top-bar.component.html*
- *http://127.0.0.1:3000?title=title&description=desc&status=1*
- *https://www.tutorialspoint.com/angular6/angular6_module.htm*
- *https://medium.com/@limitlesscoders/how-to-use-route-guards-to-protect-routes-in-angular-2703bce2e34c (AuthGuard/Route Protection)*
- *https://www.c-sharpcorner.com/article/angular-material-design-components-with-reactive-form-part-2/#:~:text=Reactive%20form%20in%20Angular%20is,have%20a%20complex%20form%20requirement. (Forms in material angular)*
- *https://www.javatpoint.com/typescript-tutorial (TypeScript)*