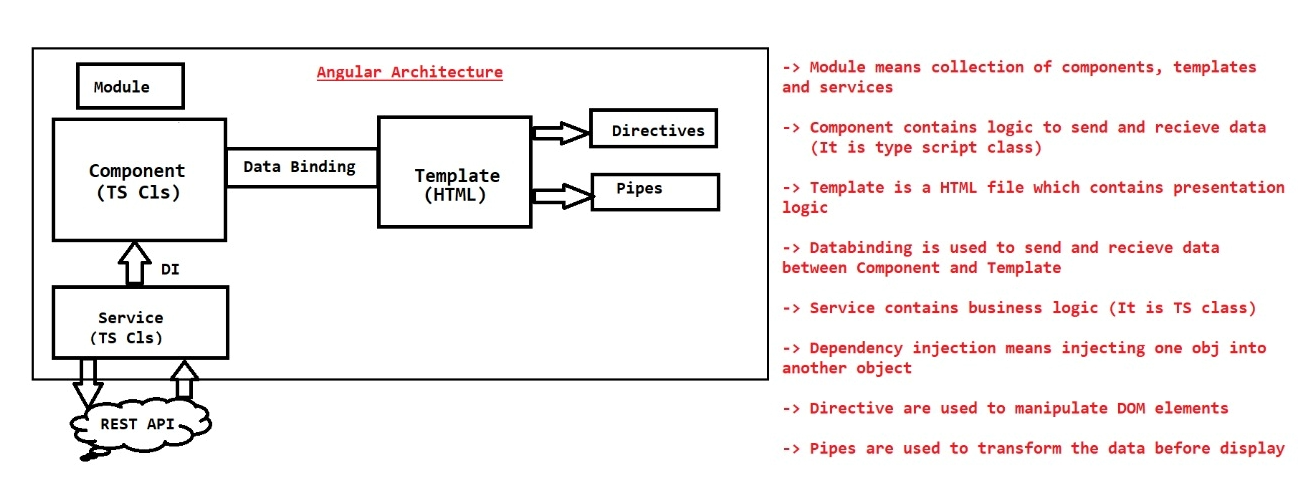
Angular Short Notes



* Angular application is collection of components. In components we will write logic to send data to template and capture data from template. Components are TypeScript classes.
* Metadata nothing but data about the data. It provides information about components and templates.
* Template is a view where we will write our presentation logic. In Angular application template is a HTML file. Every Component contains its own Template.
* Data Binding is the process of binding data between component property and view element in template file.
* Module is a collection of components, directives and pipes
* Service means it contains re-usable business logic. Service classes we will inject into Components using Depdency Injection.
* Dependency Injection is the process of injecting dependent object into target object. In Angular applications services will be injected into components using DI.
* Directives are used to manipulate DOM elements in the Template.

(We can execute presentation logic based on conditions like if-else , loops etc... using directives)

* Pipes are used to transform the data before displaying

(lower case to upper case, INR to USD, dd/mm/yyyy to DD-MMM-YYYY)

* In angular application we can have any no.of modules
* When we run angular application, it starts execution from startup module i.e app-module
* Angular application boot strapping will begin from app module
* AppModule will bootstrap AppComponent
* AppComponent is the default component in Angular application
* In Angular application "index.html" file will act as wecome file
* When we access Angular application URL in Browser it will load index.html file
* In index.html file we are using AppComponent selector to invoke AppComponent.

<app-root></app-root>

* Components:
* The component class includes "properties" to store the data, "methods" to manipulate the data.

import {Component} from "@angular/core"

@Component (meta-data)

class ClassName{

property:dataType = value;

method(args) : returnType {

//logic

}

}

* selector : represents tag which is used to invoke the component
* templateUrl : represents the html file that has to be rendered when the component is invoked
* template represents content of content
* styleUrls : Represents the list of styles (css) that have to be loaded for the component.
* providers : Represents list of services to be imported into the component
* animations : Represents list of animations to be performed in the component.

Data Bindings

1. Interpolation :

* It is used to display the value of property in template
* If the property value is changed then automatically it will be updated in template
* syntax : {{propertyName}}

1. Property Binding

* •Property Binding is used to send the data from component to template and assign the same into an attribute of tag.
* •If the property value is changed then automatically it will be updated in template

Syntax: [attribute]=\*property

1. Event Binding :

* It is used to pass event notifications from template to component

Syntax : <tag (event) = "method()" > </tag>

1. Two Way Binding:

* **"ngModel"** is the pre-defined directive which is used to achieve two-way data binding.
* Two data binding is applicable **only for <input/> and <select/> tags.**
* **FormsModule must be imported** in order to use Two Way Data Binding

Directives

**style**

* It is used to set CSS property value dynamically at runtime.
* When Component property value changed then CSS property value will be changed automatically.

Syntax : <tagname [style.cssproperty]="component-property">

**ngClass**

* IT is used to CSS classname dynamically at run time
* Use this directive to set styles with multiple properties conditionally.

Css file :

.class1{

color:green;

font-size:30px;

html file:

<div [ngClass]="myclass">{{marks}}</div>

Ts file :

myclass:string="";

this.myclass="class1";

**ngIf**

* The ngIf displays the element if condition is true otherwise it will remove element from DOM.

<tag \*ngIf="condition"> </tag>

Note: The ngIf directive must prefix with \*

Ts file :

b:boolean;

html file :

<div \*ngIf="b" style="background-color:blue;">Congratulations...!!</div>

<div \*ngIf="!b" style="background-color: red;">Better luck next time..!!</div>

ngIf and else

* The "ngIf and else" displays one element if it is "true" otherwise it displays other element.

syntax: <tag \*ngIf="condition; then template1;else template2"> </tag>

<ng-template #template1>

...

</ng-template>

<ng-template #template2>

...

</ng-template>

**ngSwitch**

* The "ngSwitch" checks the value of a variable, weather it matches with any one of the cases and displays element when it matches with anyone.
* Use "ngSwitch" if you want to display some content for every possible value in a variable.

Syntax:

<tag [ngSwitch]="property">

<tag \*ngSwitchCase="'value'"></tag>

<tag \*ngSwitchCase="'value'"></tag>

<tag \*ngSwitchCase="'value'"></tag>

<tag \*ngSwitchDefault></tag>

</tag>

**ngFor**

* It is used to repeat the tag once for each element in the array. It generates (repeats) the given content once for one element of the array.
* We have to use use prefix '\*' before "ngFor"

**Usecase**: Displaying all products available in shopping cart.

Syntax: <tag \*ngFor="let variable of arrayname"> </tag>

**ngFor with Object Array**

* Using this technique we can print array of object values in web page.
* First we have to store set of objects inside array then read objects one-by-one using "ngFor" and display the data in table format.

Usecase: Reading Product details (name & price) and displaying them.

syntax to create object array

------------------------------

**arrayRefVariable**:classname[] = [

new ClassName(), // new Employee(101, "John", 5000),

new ClassName(), // new Employee(102, "Smith", 5000),

new ClassName() // new Employee(103, "Nick", 6000)

];

syntax to use object array using ngFor

---------------------------------------

<tag \*ngFor="**let variable of arrayRefVariable**">

variable.property1

variable.property2

</tag>

**ngFor directive with Add and Remove functionality**

* We can allow the user to add new records (objects) to existing array. User can also delete existing records.

Adding element to array

arrayVariable.push(value);

Removing element from array

arrayVariable.splice(index, count);

**Services**

* The service is a class which contains re-usable business logic (encryption, de-cryption, validations, calculations etc.)
* Service class logics we can access in one or more components
* **If we keep re-usable set of properties and methods as part of service class, then we can access them from any component and from any other service available in the application.**
* We must declare **service class with "@Injectable()" decorator**, to make the service can be accessed from any component.
* We need to import "@Injectable()" decorator from "@angular/core" package.
* We must use **"@Inject()" decorator**, **to request angular to create an object for the service class. Then angular framework will automatically creates an object for the**

**service class and passes the object as an argument for our Component class Constructor.**

**Note: Realtime applications contains logic to access Backend Rest APIs in Service classes.**

Syntax -------

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

**@Injectable()**

class ServiceClassName {

//methods here

}

Add Service as a Provider in the Component

@Component({..., **providers : [ServieClassName]** })

class ComponentClassName {

}

Get the Instance of Service using Dependency Injection

import {Inject} from "@angular/core";

@Component( { } )

class ComponentClassName{

constructor**(@Inject(ServiceClassName) variable:ServiceClsName** ){

}

}

**Pipes**

Pipes are used to transform the value into user-expected-format Pipes are inovked in expresssion (**interpolation binding**), **through pipe (|) symbol.**

List of built-in pipes in Angular 2+

------------------------------------

1. uppercase 2. Lowercase 3. Slice 4. number 5. currency 6. percent

7. date 8. json etc.

Eg

City In Uppercase:: {{city | uppercase}} <br/>

City In Lowercase : {{city | lowercase}} <br/>

Slice : {{city | slice : 2:6}}<br/>

Currency in USD: {{salary | currency:"USD"}} <br/>

Currency in INR : {{salary | currency:"INR"}} <br/>

Short Date : {{dt | date : "shortDate"}} <br/>

Medium Date: {{dt | date: "mediumDate"}} <br/>

Medium: {{dt | date: "medium"}} > <br/>

Formatted Date : {{dt | date:"d/M/y"}}

**Forms and Validations**

In Angular we can develop forms in 2 ways

1) Reactive Forms

2) Template-Driven Forms

Template Driven Forms

* Template driven forms are suitable for development of Simple forms with limited no.of fields and simple validations.
* In these forms, each field is represented as a property in the component class.
* Validation rules are defined in the template using "html5" attributes. Validation messages are displayed using "validation properties" of angular.
* "FormsModule" should be imported from "@angular/forms" package.

**HTML5 attributes for Validation**

**required**="required" : This field is mandatory

**minlength**="n" : Minimum no.of characters

**pattern**="regexp" : Regular expression

**Reactive Forms**

* Reactive Forms are also called as Model Driven Forms
* Reactive forms are new types of forms in angular which are suitable for creating large forms with many fields and complex validations
* **In these forms, each field is represented as "FormControl" and group of controls is represented as "FormGroup"**
* "**ReactiveFormsModule**" should be imported from "@angular/forms" package.
* Validation rules are defined in the component using "**Validators**" object of angular and **validation messages are displayed in the template using "validation properties" of angular.**

**Validations in Reactive Forms**

* Validators.required
* Validators.minLength
* Validators.maxLength
* Validators.pattern

Validation Properties

* untouched
* touched
* pristine
* dirty
* valid
* invalid
* errors

**Routing**

* The routing concept is used to create page navigations in angular2+ applications
* "Routing" is the process of mapping between the "route(url)" and corresponding component

Angular2+ supports two types of routing

1. Hash-less routing Ex : /home
2. Hash routing Ex : #/home

Steps for working with Routing

Import "@angular/router" package in "package.json" file

"dependencies":{

"@angular/router": "latest"

}

* Set the base location of the application on server:

<base href="/">

Import "Router" from "@angular/router" package

import {Router} from "@angular/router";

* Create Routes

var variable1: Routes = [

**{path:"path-here", component:ComponentClsName},**

{path:"path-here", component:ComponentClsName}

]

Import "RouterModule" from "@angular/router" package

import {RoutesModule} from "@angular/router"

Combine Your Routes & RouterModule

var variable2 =

RouterModule.forRoot(variable1,useHash:true/false});

Import both "routes" and "RouterModule" in AppModule

@NgModule({..., imports : [..., variable2]})

class AppModule{

}

Create Hyperlink to route

<a routerLink="/path">Link Here</a>

Create Place Holder to display route content

<router-outlet> </router-outer>

AJAX

* AJAX stands for Asynchronus Java Script and XML
* AJAX is not a language but it is a concept which is used to send request from browser to server and also get response from server to browser without refreshing(reloading) the web page in browser.
* AJAX allows us to interact with the server and get some data from server without refreshing full web page.

**Types of Ajax Requests**

GET : Used to retrieve/search data from server

POST : Used to insert data to server

PUT : Used to update data on server

DELETE : Used to delete data from server

* '@angular/common/http' package provided necessary services to send AJAX request to server and get AJAX response from server
* If we want to send AJAX request we will import and inject HttpClient. Using this HttpClient we can send AJAX request to server.

Sending GET request to server

this.http.get<ModelClsname>(

"url",

{responseType: "json|text"})

.subscribe(this.successCallbackFunction, this.errorCallBackFunction);

Sending POST request to server

this.http.post(

"url",

{data},

{responseType:"json|text"}).

subscribe(this.successCallBack, this.errorCallBack);

Sending PUT request to server

this.http.put(

"url",

{data},

{responseType:"json|text"}).

subscribe(this.successCallBack, this.errorCallBack);

Sending DELETE request to server

this.http.delete(

"url",

{responseType:"json|text"}).

subscribe(this.successCallBack, this.errorCallBack);

Define Success Callback function

successcallback = (response) =>

{

//do action with response

}

Define error callback function

errorcallback = (error) =>

{

//do action with error

}