**-> ANGULAR JS 2**

* Angular 1 was not built for mobile devices, its possible to run angular 1 on mobile but we will be requiring 3rd party frameworks for that
* The advantage of component based development makes it more reusable and improves the testing part and makes angular more testable
* **Installation:**
  + Install node and npm
  + Install visual code and configure the environment
  + Configure the settings such that they search for node and npm from the path from tools - > options -> projects and solutions -> external web tools
  + Install typescript in visual studio
  + Create an empty asp.net web application
  + Download quick start files from angular
  + Restore the required packages
* To create an angular application from command prompt
  + ng new my-app
  + cd my-app
  + ng serve --open will open the browser
  + code .
* the web browser does not understand the typescript file but only understands javascript file so we need to convert the typescript file to javascript file
* This is done when the application is build and then the typescript compiler will execute and will convert the typescript file to the javascript file
* Everything in angular 2 is a component so it is the basic building box of angular 2
* Components in angular is a class which contains template , class and the decorator provided by angular
* Template provides the ui of the web app
* Class contains the code for the template
* Its properties specify the data for the template
* And methods specify the logic for the template
* We are going to code everything in typescript
* By convention the root component in angular is the “app component”
* App.module bootsrtraps and launches the angular application
* BrowserModule is required for connecting to the browser
* NgModule is required to add metadata to angular module
* If we decorate a class with NgModule then it becomes a component
* tsc = transcript compiler
* Export keyword is used before “class” keyword so that other components in our app can import it and use it as required
* To add a decorator to a class we use “@’name of the component’”
* “selector” property in the component meta data specifies the place where the html page is to be displayed
* “template” property in the component meta data specifies the html to be displayed req for the view
* The “template” html should be in back tick characters and not single quotes
* We can use “single” or “double” quotes in view template but only in case when view template can be specified in single line
* We use pair of back tick characters when the view template is in more than 1 line
* To specify the url of the template to be used we use another meta data property “templateUrl”
* Angular says if the template view is having more than 3 lines then it should be not inline
* templateUrl is relative to index.html
* Applying styles to angular components
* Disadv of using external stylesheet is that we need to refer this file in the index.html
* Another disadv of using external stylesheet is that it leads to unpredictable results and to prevent them we need to overwrite them in in-styling technique
* Styles defined in external stylesheet are global and may affect the documents having in-styling
* Adv of using external stylesheet is that we can do the changes only in 1 place
* Adv of using inline styles is that since the styles defined are local to the component this file can be easily reused
* Also another adv of this is that these styles wont collide with other styles specified for the application
* Disadv of inline styling is that whenever we need to make changes everywhere where this change should be done, this is tedious and time consuming
* This is also not a good practice to combine the html and the styles like this
* We can define styles in the @Component decorator using styles property and specifying an array of the styles to be applied
* Interpolation is all about data binding
* {{}} in view template is called template expression and the value in it is converted into a string
* We can call methods of the class from template expression
* Property binding and interpolation are same in functionality but differs in syntax
* Interpolation is a special syntax that angular converts into property binding
* Canonical variation of property binding
* Property binding and interpolation automatically sanitizes the malicious code before rendering it on the browser
* DOM(Document Object Model) can be considered as an API which helps to play with the html elements in an HTML doc using its formed DOM
* Property binding binds the DOM object property to a value and not an HTML element’s attribute to a value
* There are cases when the html element’s attribute does not have corresponding property so in this situation we tend to bind the component’s property with element’s attribute directly, this is called Attribute Binding
* Angular recommends to do property binding and perform attribute binding when there is no corresponding property for an HTML element
* Eg collspan attribute of th element does not have a property
* With property binding we use a pair of square brackets and with event binding we use a pair of parentheses and include the event which we want to observe
* \* before a directive says its a structural property
* \*ngIf is a structural directive provided by angular which evaluates a predicate and depending on the result it add the html element to the dom or not
* 2 way binding in angular can be thought as a combination of event binding and property binding
* trackBy property is used to prevent performance issues by instead of using the object reference we can refer to the data using its property
* Pipes transform data before displaying it
* There are several in-built pipes provided by angular
* Angular takes date in “mm/dd/year”
* In order to make a class a pipe we need to decorate the class with @Pipe decorator
* When we add a component inside another component it is called Nested Component or the Child Component
* The component which contains this component is called the Container Component or the Parent Component
* To make properties in a class as input properties we need to decorate them with Input Decorator
* Single = is used to assign values
* Double == is used to compare 2 var values
* Triple === is used to compare 2 values and their data types
* Event data is commonly called as “Even Payload”
* Using 2 angular structural directives together is not allowed in angular with 1 element
* Typescript is a strongly typed language that every variable in it has a data type
* For defining type of an object we are going to create interface in another file
* It is common convention to prefix Interface name with ‘I’
* Interface contains properties and methods
* We can make a property in interface optional that is it is not necessary to include that property in the class that implements the interface
* We make the property in an interface optional by adding a question mark in front of it(department?)
* The order in which life cycle hooks are specified is the order in which they are implemented
* We don’t need to write “implements interface\_name” while implementing life cycle hook but it is preferred and a makes implementation easy when we add this line
* To make a class a service we decorate that class with “Injectable” decorator
* We use this decorator to inject any dependencies in our service class
* “providers” property in @Component helps to register a class with dependency Injection System as a provider
* Service used in a component is available to its children
* When a service is registered at module level it is registered in root angular injector, then it will be available for all the components
* Service calls are usually done with http calls
* ngOnInit is the best lifecycle hook to make service calls
* ngOnInit is called after constructor and is used to perform tasks of angular binding like calling service method, the tasks that are time consuming should be called using ngOnInit and not constructor
* When we subscribe to an observable from an observer we specify the callback function by specifying parameters in the subscribe function
* Usually services does not have ui associated with them
* Routes is an array of routing objects
* More specific routes should be at top and more general routes should be at bottom
* When to bind to link parameters array we need to enclose the “routerLink” in square brackets as [routerLink]
* Angular injector is the one which passes the instance of the service to a component’s constructor by creating an object of the component
* To tell about our service to the angular injector we need to register our service with the angular injector
* This registering of service is done using providers property of NgModule or Component decorator
* Component specifies the dependency on the service by the code in its constructor for service’s private instance
* In dependency injection we have an external source(I.e the Angular injector ) which is creating the dependency instance, so our dependent class need not change when the the class on which our class is dependent changes
* In angular 1 we had only 1 injector
* In angular 2 we have a complete injector hierarchical tree
* Thus we have 1 root injector and 1 injector at every component level
* Navigate method of Router is used to programmatically route to a particular page
* In angular by default http service returns an observable
* Throw method returns an Observable
* To deal with asynchronus data we can use observables or promises
* Lazy components are those for which service call is not given untill they are subscribed to the service
* Non Lazy makes a service call even if they are not subscribed to the service
* Problem with “retry ” operator is it immediately retries
* retryWhen operator is used when we need to retry after some delay
* Angular1 is controller based
* Angular2 is component based
* Angular2 leads to more code reuse, better testing , 5times faster, mobile support ,
* **ng g c “component name” --spec false --flat true:** here flat says that the folder we are specifying for the component it should be created in that component only and not in any other folder of its own
* Base href in inidex.html specifies the path for the navigation
* Template driven forms are generally used to create simple forms
* Reactive forms are used to create more complex forms
* Disabled form controls are not included in angular generated form model
* Label is assigned to an input box using “for”
* ngModel data binding is done using “name value”
* Structural directives add or remove items from DOM
* If all the fields of valid are valid the form is valid and vice versa
* We can validate email using 2 methods : pattern validator and email validator
* In pattern validator we use regular expression to validate our fields
* If our option value is a string then use normal “Value” but if it is null then use “ngValue”
* Required validation wont work when the default value of the item is other than null , in this case we need to create our own custom validator
* to create a custom validator we need to imlement a Validator interface
* To make a class a directive we need to decorate it with @Directive decorator
* Abstract control class is the parent class of both form control and form group class
* Form controls are the items in form say textbox, checkbox, radio buttons etc
* Form group is the form element
* We need to register our custom validator using providers variable
* We need to add it to the module of our project
* The input parameter to the custom validator should have the same name as that of the custom validator selector
* But to use another name for the input parameter we can use alias
* Validating password and confirm password is cross platform validation and we cant perform this cross platform validation using built in angular validators , we need to do this using custom validators
* Service is used to get data without hard coding it
* ngSwitchCase is a structural directive
* If there are multiple case values all of them will be displayed
* To detect changes in an input property we can follow 2 ways:
  + onChanges life cycle hook
  + Property Setter
* A child component uses an event to pass data to its parent component
* Output properties use events to send data from child to parent component
* We create a template reference by prefixing the variable with ‘#’
* Using this template reference we can call and use the public methods and public properties present in the respective component of the template
* There are 2 ways to send data from child component to parent component
  + Output properties : there are several moving paths
  + Template reference variable : there are relatively less moving paths
* Route guard is implemented as a service
* CanDeactivate guard wont work if we directly route by changing the url
* Also it wont work if we close our browser window
* It wont work while navigating to external website like google.com or any other external website
* Link parameter array in routing navigate method includes 2 parameters
  + The routing destination path
  + The parameter for the routing
* To read parameters from the rout link angular provides a service called activated route service
* We convert a string value to a number using ‘+’ symbol
* We use square brackets around ‘routerLink’ only when we are assigning its value to a link parameter or a property of the component otherwise we don’t use it for simple strings
* Using the ActivatedRoute class we can get the parameters from the route link
* Using the Router class we can navigate to a particular route link
* When we use snapshot to read the route parameters it gets executed only when the component is refreshed and wont be executing when called from the same component
* To react with each change in the parameter in the route we need to subscribe to the change which is done using observable method
* Whenever we subscribe to an observable then whenever there is a change we need to perform some operation
* It is necessary that when we subscribe to an observable we need to unsubscribe it
* But there are some exceptions to this like ActivatedRoute where the Angular Router takes care of this and automatically unsubscribes it
* For passing optional parameters we pass an object in the form of key value pairs
* To read optional parameters we again use the Activated Route class
* We can reset the form in 2 ways
  + Code
  + Using HTML
* Reset method also resets the form flags like dirty , prestine etc
* Object.assign() method is used to create a copy of an object , it accepts 2 parameters first - an empty object to which the object to be copied is going to be copied second - the object which is to be copied
* When reset method is called on the form the reference to the object is also reset so its values are reset to null
* Angular recommends not to use filter pipe for sorting data or for filtering as it can effect the performance drastically if not implemented correctly
* For creating a custom pipe the class needs to implement an interface PipeTransform
* Whenever we create a new pipe its by default pure
* A pure pipe will be executed whenever a pure change occurs
* There are performance implications for an impure pipe
* Impure pipes run unnecessarily when not even source data is not changed
* Impure pipes run for each change even for mouse move
* By default query params are not retained across multiple routes
* ActivatedRoute service is also used to read query string parameters
* To work with query parameters we use queryParamMap property
* To work with optional or required parameters we use paramMap property
* When we get data from a database it returns an observable of that data so to use that data we need to use that observable to display that data
* We can create an observable using a number of methods but the simplest way is to use the “of ” operator
* Route resolver can be used as a function or a service
* To make a class work as a resolver service it should implement an interface Resolver
* When we add resolver guard to the route path where we want it to get added it makes angular to wait for the resolver service to get executed first and then route to that path
* We can use the navigation events to monitor routes , also we can use them to troubleshoot when routing does not work as expected, also used to loading or processing messages when there is a delay while routing from 1 route to another
* The easiest way to remove an element from the array is using the array splice method
* Since ngIf is a structural directive when it is used to display the control on based of a condition then when condition fails then instead of hiding the control it removes it from DOM and then when condition satisfies it adds it back to DOM
* Instead of using this we can use hidden property of the control which hides the control by toggling the display to None
* This helps in better performance of the application
* When we know the control is not to be showed then we should use ngIf directive
* But if we frequently toggle between the visibility then we use hidden property
* Content projection helps to create reusable components
* ng-content in accordion is used to contain the content of say panel body and panel footer
* Depending upon the server side tech there are many ways to build REST API
* Data access logic is usually encapsulated in a service as it becomes reusable and can be used across many components
* We use GET to get data from server
* We use POST to post new data to server thus write new data to server
* PUT is idempotent but POST is not that is calling PUT with the same object number of times wont make any change
* POST creates a new item when it is called so its not idempotent
* When a resolver calls the observable service without subscribing to the observable service it does not lead to an error because resolver automatically subscribes to the observable and does not create an error
* Catch() to get the errors while making an http request to the server is a patch operator
* To call a method which returns an observable object we need to subscribe to the method
* Spec files are used for unit testing
* So in order to skip creation of these files in the project use “--skip tests”
* There are 2 ways to create forms in angular
  + Template driven form : these are heavy on templates that is we create the form completely in HTML . they are easy to create and understand . they are used to create simple forms. Creating complex forms using this technique is not recommended as HTML can become really complex. Its not easy to unit test these as most logic is in HTML
  + Reactive Forms(Model Driven Forms): these are created completely in code. These are more flexible . its easy to add elements dynamically and authenticate depending upon the decisions take during runtime in code. Its easy to unit test as most code is in component class. Only downside is they require more code than template forms
* We can think of form group as a collection of form controls
* A form group can have another form group nested in it
* Both Form Group class and Form Control class are derived from AbstractControl class
* An instance of form control class helps to get the data of the html input elements to which the form control is associated
* FormGroup instance is able to track the values of all the form controls present in the FormGroup instance
* Both formGroup and formControl directive are provided by RactiveForms Module
* Form model contains all the properties of a form
* Form group has controls property which tracks the properties of all the controls present in the form
* setValue() is used to update all the form controls and cant be used to update a subset of the form controls
* patchValue() is used to update subset form controls and can also be used to update all the form controls in the form
* There are 2 ways to create Reactive forms in angular
  + By creating independent new instances of form group and form control
  + By using the FormBuilder Class , it reduces the code required to create complex forms,
* FormBuilder class is imported as a service in the component thus we need to inject it into the constructor of the component in which it is being used
* When implementing validation in reactive forms we first import the Validators Class which provides various validating functions
* All the methods in the validators class are static functions so we can access them using the name of the class and not the instance
* valueChanges property is available in both FormControl and FormGroup class and raises an event whenever form group or form control value changes and it is an Observable so we need to subscribe to it
* When we observe the valueChanges on a form group the entire form details are shown
* Using valueChanges property we can do all the validation in code
* For getting keys of the nested form group we will get this by recursively calling the function which is identifying the keys of a form group
* Blur event is raised when a form control loses focus
* When we pass a formGroup as a parameter to the custom validator then we need to specify the validator with the group instead of a single control
* While creating angular reactive forms there are 3 building blocks : FormControl, FormGroup, FormArray
* FormArray can contain unlike items
* It can contain form controls, form groups and even nested form arrays
* There are 2 methods to create a formArray - 1. using new FormArray([]) , 2. using FormBuilder class
* Although we can store unlike items in a FormArray we usually use it to store like items
* formGroup is serialized as an object
* formArray is serialized as an array, it is useful to generate form groups and form controls dynamically
* setControl() method is used to replace an existing control, it accepts 2 parameters 1st -> the control to be replaced, 2nd -> the new control with which the existing control is to be replaced
* When we change the values of form group or form control or form array programmatically then properties like dirty , touched do not get updated
* Core Module is used to include services of HTTP Module
* Services in angular are Singleton so we provide the single instance of the service to all the components of the application , and so we include it in the Core Module
* Core Module is a pure services module with no declarations
* The core module is then imported in the root module only and we don’t import it to any other module
* Shared Module contains the reusable components , directives and pipes that we want to use across our entire application
* Shared Module is imported into specific feature module as needed
* Shared Module also may export commonly used modules like common module , forms module so that they can be easily used from any component without importing it in a component
* BrowserModule is required to provide necessary services to run an application in Browser
* BrowserModule imports and re-exports the Common Module
* BrowserModule is imported only once and only in the root Module
* In a module if we want its component to be used or to get imported by other module by just importing the module then we need to export the component
* forRoot() method for registering routes should be called only once and that too in the app module
* forChild() method for registering routes specific to a particular feature should be used in the module of that specific feature
* All feature modules should be imported before App routing Module
* We can export angular module without first importing it
* All the routes in the Angular Module that u want to Lazy Load should have the same route prefix
* There are 3 ways of module loading in angular
  + Eager : all modules must be downloaded onto the client machine before the application starts, module is referenced in the application
  + Lazy : modules are loaded when the user navigates to the routes in those respective modules
  + Pre : when modules are downloaded in client machine in the background after the necessary modules for the application are loaded and downloaded so it improves the performance of the application