Angular

- front - end technology used to create the UI
- first version of angular - Angular JS
- from Angular 2 onwards, Angular TS
- TS - typescript
- TS is called super set of javascript, it includes all the methods and properties of JS along with
- data type declaration
- used to create SPA (only index.html file)
- It is a framework
- In angular we have seperate file for HTML, CSS, and TS (along with .spec file used for unit testing)
- Angular developed and maintained by Google
- Angular application and all its components can be created uisng angular-cli (command line interface)
- npm i -g @angular/cli@16.2.16 : to install angular cli
- ng new application_name   : to create an angular application

Create an angular application

- eg: ng new demo (to create an application)
- cd demo (to go inside that project folder)
- npm start / ng s -o (to run angular application)

Files and Folder structure

 .gitignore : files and folders, no need to push into github
 .editConfig : vs code configuration
 angular.json : angular project configuration files
 README : Description about the project
 tsconfig: typescript configuration
 node_modules: all installed dependencies
package.json : it includes scripts, packages installed, dev dependecies
package-lock.json: more detailed version of package.json
src folder: it includes components, assets, main.ts, index.html, style.css
index.html: the only html file that renders
app - base components
style.css: global style sheet
main.ts: main typescript file. entry point to the application
while creating a component, 4 file will create
    1) .component.html
    2) .component.css
    3) .component.ts
    4) .component.spec.ts (unit testing)

app-routing.module.ts : used to provide routes

Data Binding

```
----------------
1) one- way data Binding
    a) ts file to view (html)
     {{}} - string interpolation , to display data from .ts file to html file
    b) property binding
     syntax:    [attribute] = "value"
        eg: <img [src]="imagPathName">

    from HTML to TS
    ---------------------
    c) event binding
    syntax:   (event)="method"
    eg:(keyup) ="egtUsername($event)"
                event.target.value :  user entered value
    d) using template literal
       assign a nmae to input html elemnet
       syntax : #templatename
       next pass this template name in event to component.ts file, there access it
using data.value

2) two-way data binding
 - if we change value in component.ts, automaatically
 need to change the value in html and vice versa
 - data sharing in both the direction
 - [(ngModel)]
- if we want use ngModel, we need to import FormsModule in app.module.ts
- if we use ngModel in any html element, we need to declare name attribute also for
that html element
- no need to provide same value for name and ngModel

Dependency Injection
--------------------
- DI is used to share data between classes or services,
- the data can be methods, properties, etc
- dependency injection is done inside constructor
syntax:
 constructor(access-specifier variable_name:DependencyClass){

 }
 eg: constructor(private authRouter:Router)
 3 types of access specifiers : private, public, protected

 Routing
==============
Routing is used to navigate from one page to another
it is defined in app-routing.module.ts
 syntax:
 const routes: Routes = [
  {
    path:'register',component:RegisterComponent
```

```
  },{
    path:'login',component:LoginComponent
  },
  {
    path:'', component:LoginComponent
  }
];

in app.componet.html (parent component); we have to defined below tag
<router-outlet></router-outlet>

usage
-------
in component.html
--------------------
<a routerLink="/path"></a>

in component.ts
-------------------
inject Router dependency in constructor
dependencyName.naviagetByUrl('path')

services
---------------
- services are also classes, that contains methods and properties
-  used to share properties or methods between components
- ng g s service_name

Directives
------------
- used to give conditional or add extra behaviour to HTMl Dom
- 3 types of Directives
    1) component Directives
     -  component itself is called a directive. by using its selector we can bind
that componet anywher
    2) structural Directives
     - it changes the struture of HTML dom, by removing or adding new elements to
the dom
     - used to provide conditional rendering
     eg: *ngIf
         *ngFor
    3) attribute Directives:
        -  to change the appearence of an HTMl element
        eg:[ngClass]

Dealing with asynchronous operations in Angular
-------------------------------------------------
  Observables and Promises:  both can used to deal with async operations
  - But in angular we are commonly using Observable
```

1) Observable:
 - Observable can resolve more than one request at a time,
  - it continously checks, whether data is coming
 - but promise can handle only one request a time
- In promise, we got the result in .then()
- but in Observables, we get result in .subscribe()
- for an observable, there are two states
    1) next
    2) error

  syntax:
  this.adminService.adminAuthorization.subscribe(()=>{
    next:(res:any)={
      // getting success result here
    },
    error:(err:any)=>{
      // errors are getting here
    }
  })

  Angular Life cycle Hooks

  From the creation of a component to its destruction, there are different stages
 this stages are called life cycle
 inorder to execute any methods in any of the life cycle stages we use life cycle hooks

 1) ngOnchages
 2) ngOnInit
 3) ngDoCheck
 4) ngAfterContentInit
 5) ngAfterContentChecked
 6) ngAfterViewInit
 7) ngAfterViewChecked
 8) ngOnDestroy

 - ngOnInit() hook is called when component is initialized;
 - First method called when a component loaded is constructor()

Angular Pipes

Pipes are small fumctions/ methods that can be used in temlate(.html). It accept some input value and
tranform into new value
pipes are used to tranform input string, currency amounts, dates,etc

Two types
----
custom pipes: user created
in-built pipes : already available with angular

Pipes are applied by | symbol (pipe symbol)
eg: uppercase, lowercase, date

Creating a custom pipe
---
ng g p pipe_name
eg: ng g p search

Subject and Behaviour Subject
==========
this both are used to send data between components
- for Behaviour Subject, initially there must be a value
- for Subject, Initiaaly there is no value
- we are creating these in service.ts file
- then we can subscribe to this subject/behaviour subject in commponent where ever we need

 Guards
 ==========
Angular guards are commonly used to restrict path,
that means, to check whtether user is logged in or not, or logged in user has the permission to access the path, etc
in gurad, there are different methods are there
1) canActivate
2) canActivateChild
3) canDeactivate
4) resolve
5) canload
command to create guard: ng g g guard_name

Two Types of forms in angular
---
1) Tempalte driven forms
2) Reactive forms
 - The above two types of forms are different aproaches for handling user inputs and valdiation

 1) Tempalte driven forms:
  - Are simple and more relays on Angular 2 way data binding
  - form structure and validations are written in temaplete file(component.html)
  - it uses [(ngModel)] to get the data from input field

```
<form (submit)="login()">
  <input type="text" name="username" [(ngModel)]="username">
  <button type="submit" />
</form>
```

2) Reactive forms

 - also called Model driven forms
 - Form is created and manged directly in the component .ts file-

- it uses angular's FormControl and Form builder for creation of form
- we have to import ReactiveForms in app.module.ts
- key named formControlName is used to access the vslue of input field

Data sharing between components
---
1) parent component to child component
  - @Input() decorators used
  - in parent.component.html:
    <app-child [dataFromParent]="data to send"><app-child>
   - In child.component.ts
      inside class definition access this data using Input decorator
  @Input dataFromParent:any

2) Child to parent component communication
  - @Output decorator and eventEmitters
  In this method, child has to emit the data that need to send to parent, for that
  we have to define event emitter in child component as Output decorator
   @Output() dataEmitter = new EventEmitter<string>();
   this.dataEmitter.emit(data_need_to_send)
   In parent componet.html
   <app-child (dataEmitter)="getDataFromChild($event)"><app-child>
   we have to catch the emitted data as like above
   so in parent.component.ts
   getDataFromChild(data:any){
    console.log(data)
   }
3) Between un-related components
  - Subject and Behaviour Subject