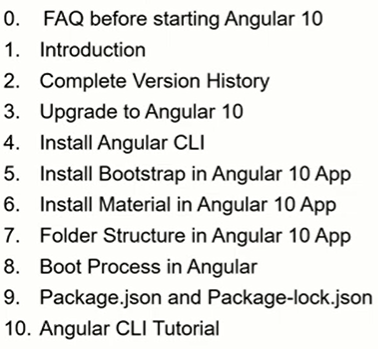
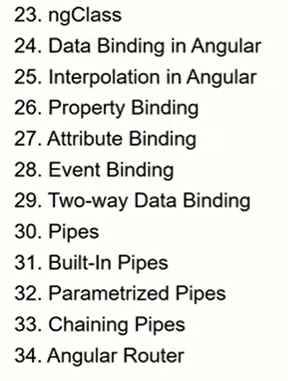
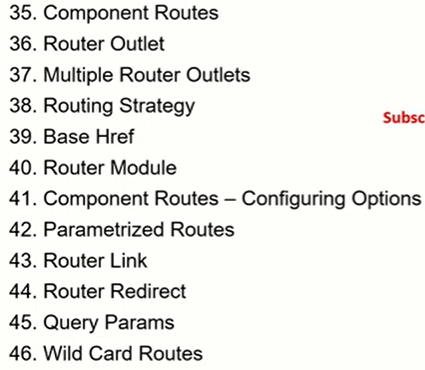
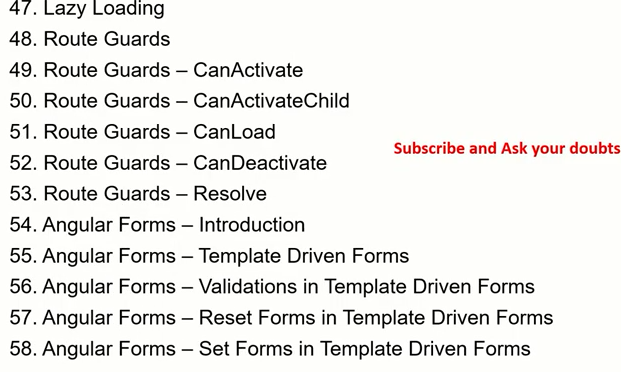
Angular classes

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

  ****

****

**Version History**

**Angular JS (or)Angular 1.x**

**-----------------------------------**

* Angular JS usually referred to as “angular.js or Angular 1.x
* It is manily maintained by google and challenges encountred in developing single page applications
* It aims to simply both the development and the testing of applications by providing a framework for client-side model-view-controller(MVC) and model-view-viewmodel(MVVM)architectures, along with components commonly used in rich intenet applications.
* It was initially relased in oct2010
* AngularJS code is written in javascript
* Angular JS I now long term support(LTS)mode.

**Angular 2**

**-----------------------------------**

* Angular 2 is complete rewirte from the same team that built angularJS
* It is written in entirely in typescript
* Angular1.x was not built for mobile support I mind, where 2 is mobile oriented
* Where angular 2 is mobile oriented
* It was realesed in sep 2016
* It provides choice for languages .you can use any of the other languages from ES6,ES5,typescript to write angular 2 code

**Angular 3 was skipped**

**---------------------**

Reason behind this is that version mismatch b/w @angular/core, @angular/compiler and @angular/router libraries

**Angular 4**

**---------------------**

* Angular 4 was reaease in mar2017
* This relaease is backwards compatible with 2.x.x from most applications
* There is no major change in angular 4 from angular 2
* Angular 4 is not a complete rewirte of angular 2
* Angular team has laid emphasis on making angular apps more faster, compact.
* Under the hood changes :New changes reduced the size of the generated code for your components by around 60% in most case
* Faster compilation
* Better bug fixes Alert.

**Some major changes**

* “ngif/else
* Renders 2 in place of rendered from same @angular/core
* No need to write a pattern from email validation in angular 4
* Typescript 2.1 and 2.2 compatibility
* Animations added

**Angular 5**

**---------------------**

* Angular was released in Nov 2017

**Some major changes**

* Build optimizer are added:production builds created with the angular CLI will now the build optimized by default.
* Angular universal state transfer API and DOM support
* Complier impoverments
* Internationalized number date,and currency pipes
* Http Client:@angular/http is depecreated in angular5.It is replaced with @angular/common/http library
* HttpModule is not more use in angular 5 .it is replaced by HttpClientModule of @angular/common/http library in each of your modules,inject the HttpClient service, and remove any(res=>res.son()) calls, which are no longer needed.
* Angular CLI v1.5 will generate v5 projects by default
* Angular froms added updted blur/submit
* RXJS 5.5 support

New router lifecycle events:guardcheckstart,childActvationStart,GuardsCheckEdnd,ResolveStart, ResolveEnd,ActivatedEnd, ChildActivationEnd.

Angular 6

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

Angular 6.0.0 was relased on may 2018.It was released angular CLI6 and material 6

**Some major changes**

* Two new angular CLI commands added: ng updated and ng add
* Angular elements
* Components dev kit(CDK)
* Angular material start components
* Cli workspaces
* Schematics
* Library support
* Tree shakable provoders
* Animations
* RXJSv6

**Angular 7**

**---------------------**

* Angular 7.0.0 was released on oct2018, synchronized with angular CLI7 and angular material 7.

**Some major changes**

* Cli prompots:The CLI will now promt users when running common commands like ng new or ng add @angular/material to help you discover built-in-features like routing or scss support.CLI prompt are also added in schematics
* Bundle budgets inCLI
* Angular material &CDK
* Virtual scrolling
* Drag and drop
* Content projection support in angular elements
* Depency updates:
* Typescript3.1
* Rxjs6.3
* Added support for node 10

**Angular 8**

**---------------------**

* Angular 8.0.0 was released on may 2019, synchronized with angular CLI8 and angular material 8.

**Some major changes**

* Differntial loading by default :It is a process by which the browser choosed b/w modern browsers and legacy browsers
* Javascript based on its own capabilities
* Dynamic imports for route configuration
* Builder aAPI in CLI:it is an exciting feature , using this we can customizes angular cli commands like ng build ,ng test and ng run
* Workspace API in CLI
* Web worker support
* Angular 8.3 has added new UX for an intial apa created using ng new
* Ng deploy is added in angular CLI 8.3

**Angular 9**

**---------------------**

* Angular9.0.0 was released on feb 2020, synchronized major with angular CLI9 and angular material 9.

**Some major changes**

* IVY compier:From version 9 all application are moved to IVY complier and runtime by default (in angular 8 it was in opt-in mode).it provides the following advantages
* Smaller testing,better debugging
* Improved css class and style binding
* Improved type checking ,improved build errors
* Improved build times ,enabling AOT on by defauilt
* Improved internatiolization
* New options for providein property in @injectable decorator to the previous root and module options ,you have two additional on the page.
* Platform:specifying providein:’platform’makes the service available in singleton platform injector that is shared by all applictaions on the page
* Any :providers a unique instance in every module (including lazy modules) the injects the token
* Componets harndsses
* Angularmaterail new component

Youtube playes component

Google maps

Typescript 3.7 support

**Angular 10**

**---------------------**

* Angular10.0.0 was released on jun 2020, synchronized smaller with angular CLI10 and angular material 10.

**Some major changes**

* New angular material component
* Date range picker:checkout this blog to see how to use angular material date range picker
* Warrning about common JS imports:when you use a dependency that is packed with commons , it can result in larger
* Slower applications .Starting with version 10, we now warn you when your build pulls in one of these bundles.
* Optional stricter settings.
* Typescript 3.9
* TSLib has been updated to v2.0
* TSL INT has been updated to v6
* New default browser configuarion.

**Upgrade from angular 9 to angular10**

**----------------------------**

* **How to upgrade from angular 9 t angular 10**
  + ng update @angular/cli @angular/core
* **if using angular material**
  + ng update @angular/material
* **Each and project and app are different and unique**
* **Each application will have its own uninque way of impletation**
* **Get detailed info and stepson angukar update portal**

https://update.angular.io/

**Angular 10:Install angular**

**------------------------------**

* First ,step 1 is to install the angular CLI globally
  + npm i –g @angular/cli
* Create new angular application CLI globally
  + ng new <project-name>
* Run the application
  + ng serve

**Bootstrap install in angular**

**--------------------------------**

* npm i bootstarp jquery popper --save

**install angular material**

**----------------------------------**

* ng add @angular/material
* yes

packages installed successfully

app.module.ts

step1)import { MatButtonModule } from @angular/material/button;

step2) MatButtonModule

step3)app.component.html

<button mat-button>primary</button>

Step4) https://material.angular.io/

**Folder strauctue**

**-----------------**

1.Project folder<simpleCRM>

Ng new <project-name>

2.)e2e

-protractor frameworks

-Default

-End to end test scripts

-will end with .e2e-spec.ts extension

Src

-app.e2e-spec.ts

Login.e2e-spec.ts

Protractor.conf.js

-configuration settings for running the end to end test scripts

Tsconfig.json

->basic typescript setting

3.src

-entire sources code of your application is inside src

-app module

-app component

-app.component.html ->view/ui/html code

-app.component.scss ->stylesheet

-app.component.spec.ts->unit test script

-app.component.ts ->class file

-assets

-images

-mock data

-apis

-environments

-local

-dev

-Qa

-uat

-prod

-favicon.io

-Index.html

Single page application

Index.html

<app-root>

-main.ts

-bootstrapModule

-Appmodule

--startingpoint of your application

**Interview question**

1)If I want to change bootsrap module Which one I should change and where should I change

Ans)main.ts and updated bootstrap

2)Is it necessary can we bootstrap with another modules

Ans )yes Any other module in our bootstrap

-no mandatory that it should be appModule

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

-polyfills.ts

Supporting older browsers

-Styles.scss

-global/common stylesheet for elemensts

a{

}

-test.ts

-test script for the main.ts file

-Testing the main.ts

-angular.json

->backbone of your application

->scripts

->ports

->settings

-<schematics

->angular cli

->build

->test

-karma.conf.js

->spec.ts

->test script runner

->running or executing all our scripts

**Boot process of angular**

**---------------**

* ng server
* main.ts
  + bootstrapModule
    - AppModule
      * Appcomponent

Authmodule

-login

-checkAuthentication

-Newuser

-Forgot

-loogedIn

-test.ts

-test scripts for main.ts

Index .html

<app-root>

**Package.json and package-lock.json**

* Name of app
* Version of app

Scripts

->”npm run start”

->ng serve

->angular.json

**Interview questions**

->can we add more scripts to package.json?

-yes yes

Yes

->scripts are your starting point of learning

-exsiting projects

->dependencies

->all the modules/libraries you MUST have to run in prod env

->the prod code will mostly be optimized/minimum

->devdenpendencies

->all the modules/libraries you MUST have to your app

->we may add packages/modules but we may not use time

**Angular cli- command line interface**

**------------------------**

-schematics ->angular.json / package.json

Ng serve -

Ng build 🡪build your app and make it ready for production env deployment

Ng test 🡪test and run all our unit test scripts

Ng lint 🡪will get error when we max length morethan length text

Ng e2e 🡪test and run all our end to end tests

-commands

Ng new <project\_name>

Ng generate component <component\_name>

Ng generate module <module\_name>

Ng generate pipe<pipe\_name>

Ng directive

**8 Lifecycle Hooks**

* ngOnchanes()
* ngOnInit()
* ngDoCheck()
* ngAfterContentInit()
* ngAfterContentChecked()
* ngAfterViewInit()
* ngAfterViewChecked()
* ngOnDestore()
* ngOnchanes():

1. Used in pretty much any component that has an input.
2. Called whenever an input value changes
3. Is called the first before ngOnInit

* ngOnInit():

1. Used to initialize data in a component
2. Called after input values are set when a componet is initialized.
3. Added to every component by default by the angular CLI.
4. Called only once

* ngDoCheck():

1. called during all change detection runs
2. A run through the view by angular to update/detect changes.

* ngAfterContentInit()

1. called only once after first ngDoCheck()
2. called after the first run through of intilizing content

* ngAfterContentChecked():

1. called after every ngDoChecked()
2. waits till after the first ngAfterConetentInint() on first run through

* ngAfterViewInit():

1. called after angular intilalies component and child component content.
2. Called only once after view initialized.

* ngAfterViewChecked():

1. called after all the content is initialized and checked .(componet and the child components).
2. First call is ngAfterViewInit()
3. Called after every ngAfterContentChecked() call is completed.

* ngOnDestore()

1. Used to clean up any necessary code when a component a component is removed from the DOM.
2. Fairly often used to unsubscribe from things like services.
3. Called only once just before component is removed from the DOM.

Notes:

Whichever lifecycle hooks we want to use

1.import it in the class

import { Component, OnChanges, OnInit, SimpleChanges } from '@angular/core';

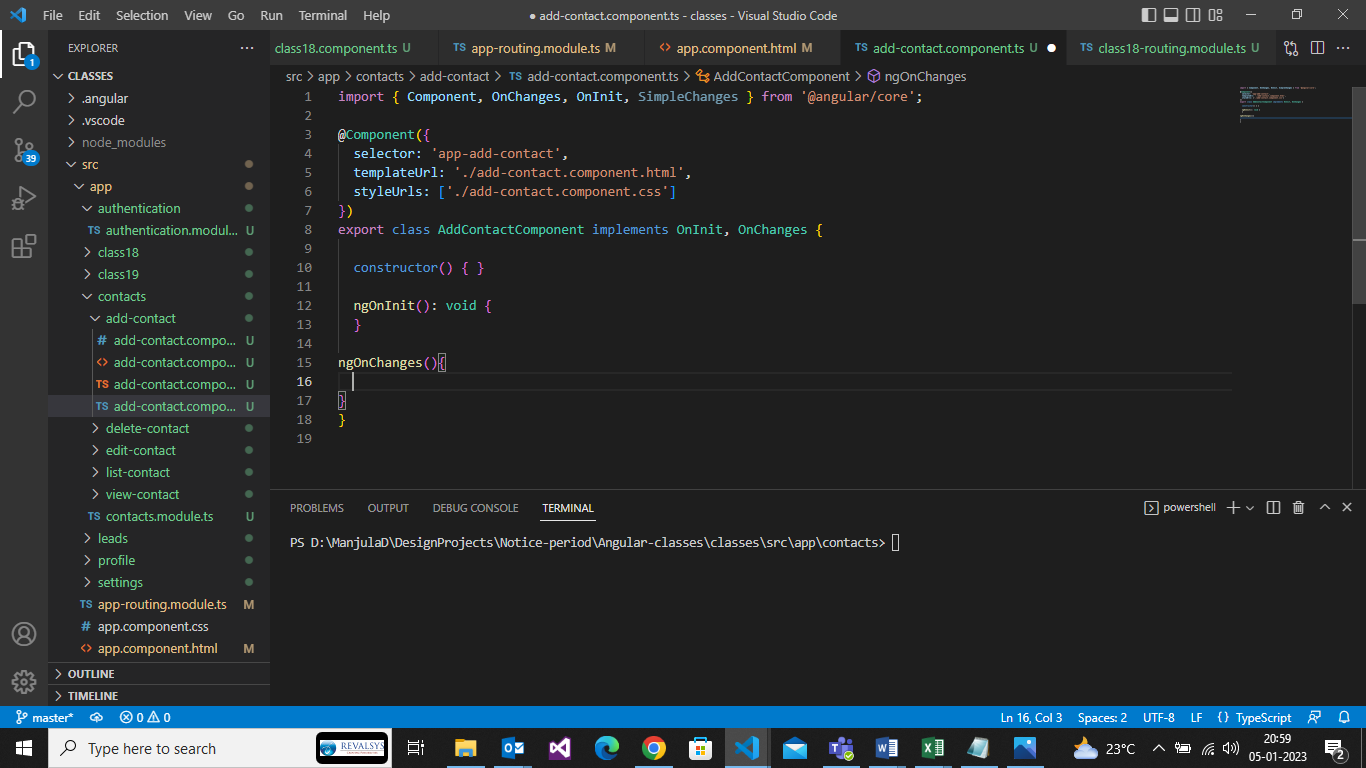
2.Extend the implements interface

export class AddContactComponent implements OnInit, OnChanges {

3.Implement methods

ngOnChanges(){

}



Communication between various angular components

1. Parent component 🡪 child component(@Input)
2. Child component 🡪 parent component(@Output)

Via parent to child (@**input** )

Via child to parent

* View child
* @**output**/**Event Emitter**

Between different components

* Via **Serivices**

Communication b/n totally unrelated components

Component1 🡪 services <-- componennt2

Services

Is a common reusable piece of functionality shared between differnet components

* Templates in components
* Different ways to include templates in components
* How to style component templates

When we generate a component

1. Template file(.html)
2. Style.css(style sheet)
3. Class(component.ts file)
4. Spec (unit test file)

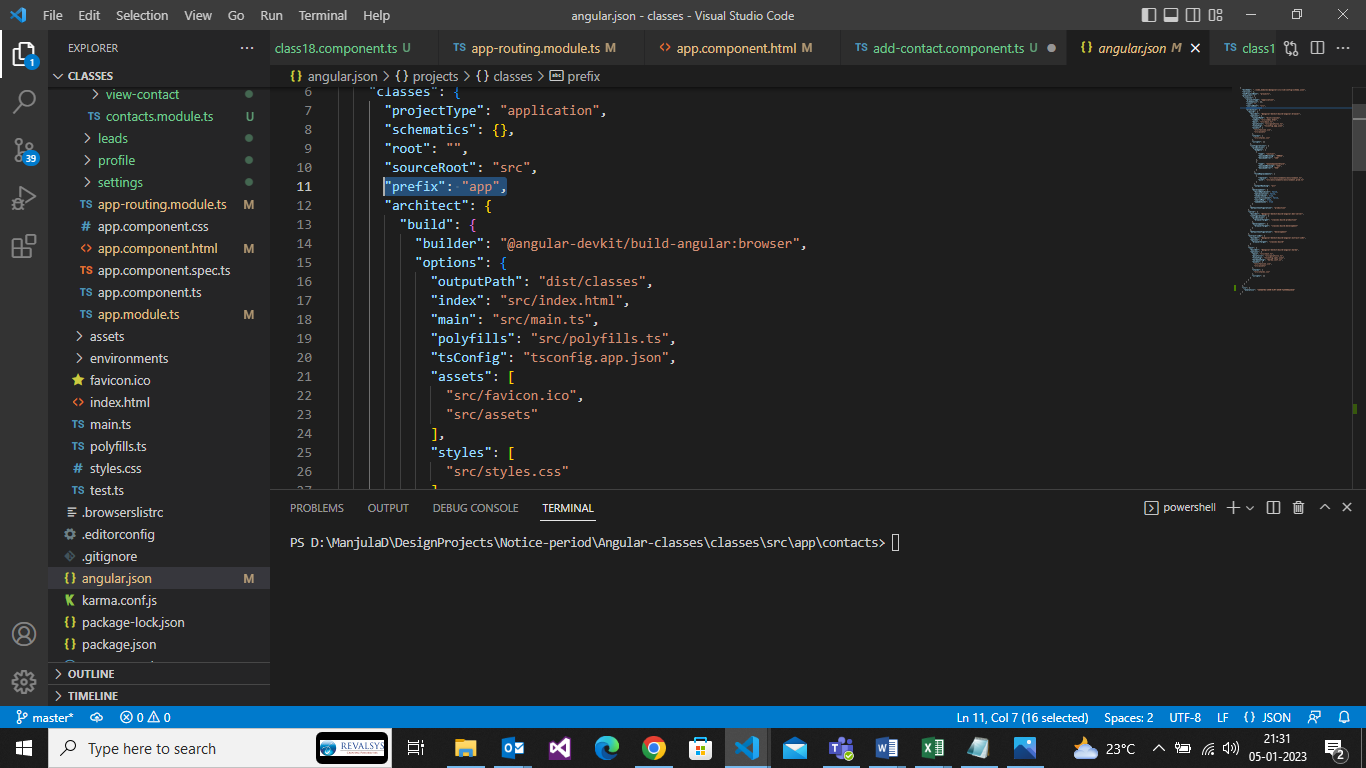
Can we change the default”app” prefix?

Yes –we can change it through the app

“app” –> “arc-tutorials”

What will happen If I change?

Nothing happens .Only thing you can change,make sure you update latest.



templateUrl

🡪is always 1 single html file

Template

🡪we will pass the template itself instead

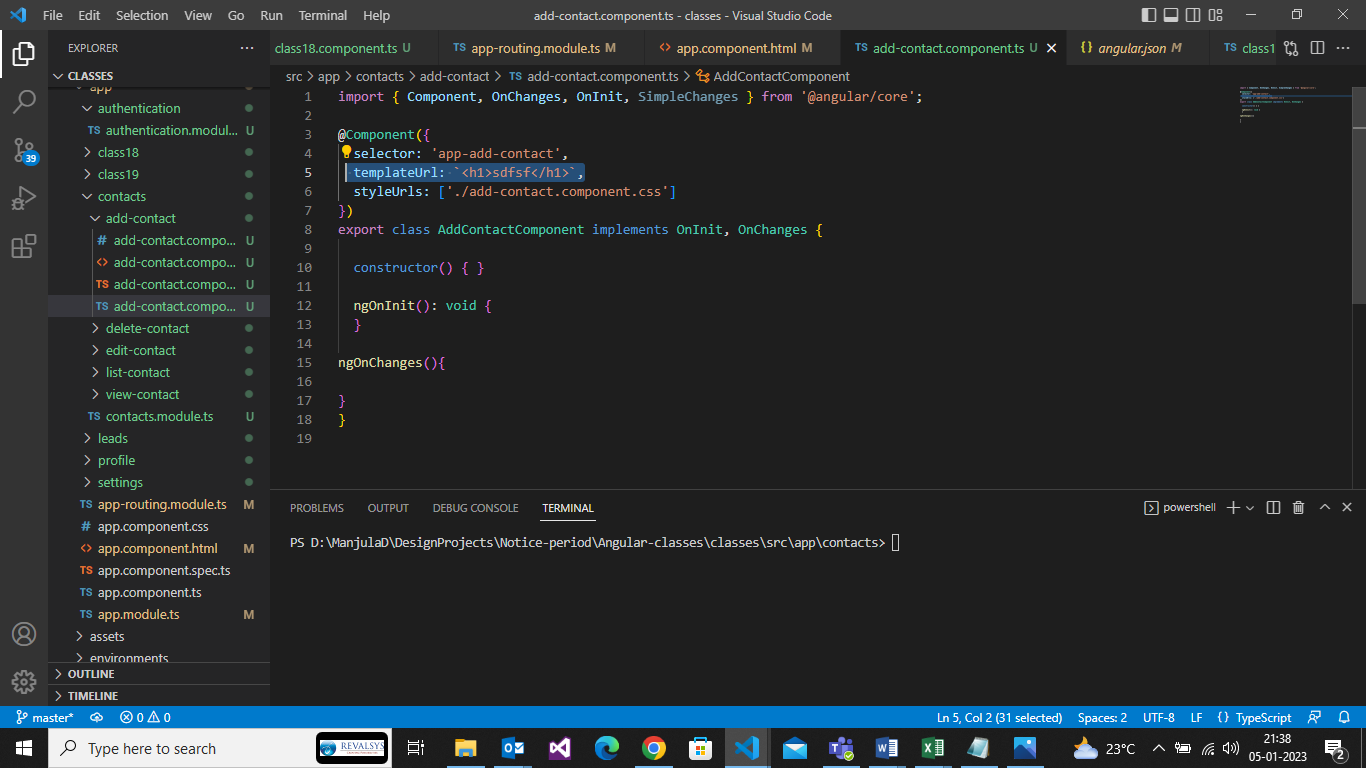
🡪backtick `<h1>sdfsf</h1>`

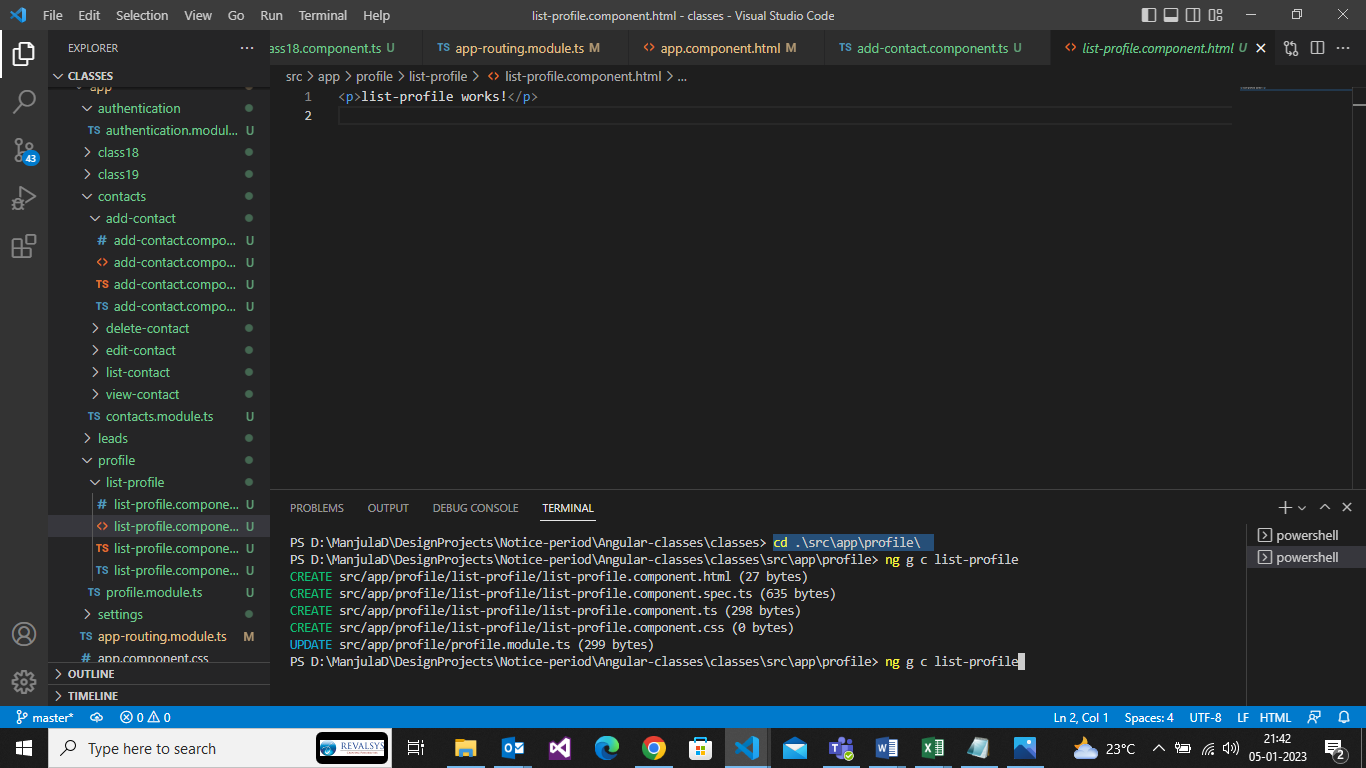
stylesheetUrl

🡪is an array

🡪it can take multiple stylesheet as input

🡪it can be one or more stylesheet





What is an Angular Directive?

* Angular directives are used to extend the power of the HTML by giving it new syntax
* Directive is a way to extend our HTML including both view as well as behaviour.
* Directives are used to extend the power of HTML.
* Directives can extend, change or modify behaviour of the DOM elements

There are 3 types of directives

* Component Directives

1. Every Angular application must have at least 1 component
2. Have it’s own templates
3. Events attached

* Structural Directives

1. Updates structure of the view
2. ngFor, ngIf and ngSwitch

* Attribute Directives
* ngStyle,ngClass

How to generate directives

.ng generate component <component-name>

.ng generate directive <directive-name>

**Structural Directives**

1.Structural directives are responsible for HTML layout .They shape or reshape the DOM’s structure,typically by adding, removing, or manipulating elements.

2.As with other directives, you apply a structural directive to a host element.The directive then does whatever it’s supposed to do with that host element and it’s descendants

3.Structural directives are easy to recognize .An asterick(\*)preceds the directive attribute name as in this example.

Types of structural directives

1.ngIf

2.ngFor

3.ngSwitch

**\*ngIf**

1.Is a built-in structural directive that can add remove elements

2.Add \* symbol in ngIf in template

3.Conditionally includes a template based on an expression

4.Resolves to true or else result of any given expression

5.Add or Remove an element dynamically

6.Example code: <div \*ngIf=”condition”></div>

Examples:

1) <div style="background-color:#f00;" \*ngIf="success\_msg">

    User added sucessfully

 </div>

**.ts**

 success\_msg = true;

**output:**user added successfully

2) <div style="background-color:#0f0;" \*ngIf="success\_msg || success\_flag">

    User added sucessfully

 </div>

**.ts**

success\_msg = true;

  success\_flag = false;

**output:**

****

3)

<div style="background-color:#f00;" \*ngIf="!success\_msg">

  User added sucessfully

</div>

4)

<div style="background-color:#00f;" \*ngIf="success\_msg; else error\_msg">

  User added sucessfully

</div>

5)

<div style="background-color:#00f;" \*ngIf="success\_msg; else error\_msg">

  User added sucessfully

</div>

6)

<div style="background-color:#f0f;" \*ngIf="success\_msg; then updated\_msg else error\_msg">

  User added sucessfully

</div>

<ng-template #error\_msg>

  user are not added

</ng-template>

<ng-template #updated\_msg>

  user are not added

</ng-template>

Output:

user not added

**\*ngFor**

1.Used along with ngIf statement

When the condition is true –using the template variable reference, the then block will be executed

When the condition is false –using the template variable reference, the else block will be executed

For e,g

1)<div \*ngIf=”showValue then thenBlock else showMessage”>Show value</div>

2)<ng-template #thenBlock>showing then msg</ng-tempate>

3)<ng-template #elseBlock>showing else msg </ng-template>

**ngSwitch**

It’s a built-in directive and starts with the [] bracket symbol

Very similar to switch statements in any other programing languages

Allows element to be shown or hidden based on a condition expression

Unlike if statement –switch can take multiple value parameters for condition check

We can also define a default action for the ngSwitch

ngSwitch

ngSwitchCase

ngSwicthDefault

Examples:

**example1:**

<div [ngSwitch]="superPower">

    <div \*ngSwitchCase="'wonderwomen'">Display wonder woman supre powers</div>

    <div \*ngSwitchCase="'wonderwoen'">Display wonder woman supre powers</div>

    <div \*ngSwitchCase="'wondermen'">Display wonder man supre powers</div>

    <div \*ngSwitchDefault="'wonderDefault'">Display wonder wonderDefault powers</div>

</div>

.ts

superPower = 'wonderwomen';

output: Display wonder woman supre powers

**example2:**

<div [ngSwitch]="tax">

    <div \*ngSwitchCase="10">Display wonder</div>

    <div \*ngSwitchCase="20">Display powers</div>

    <div \*ngSwitchDefault="20"> Display default </div>

</div>

.ts:

 tax = 30;

output: Display default

**example3:**

.html

<div [ngSwitch]="g">

  <ng-template ngSwitchCase="m">i am for male</ng-template>

  <ng-template ngSwitchCase="f">i am for female</ng-template>

  <ng-template ngSwitchCase="o">i am for other</ng-template>

  <ng-template ngSwitchDefault>default case</ng-template>

</div>

.ts

 g = 'ds';

Output: default case

2)

<input type="text" [(ngModel)]="g" />

<div [ngSwitch]="g">

  <ng-template ngSwitchCase="m">i am for male</ng-template>

  <ng-template ngSwitchCase="f">i am for female</ng-template>

  <ng-template ngSwitchCase="o">i am for other</ng-template>

  <ng-template ngSwitchDefault>default case</ng-template>

</div>

**.ts**

g = 'ds';

Output: 

i am for other

3)onkeyup ?

<input type="text" [(ngModel)]="g" #t (keyup)='0'/>

<div [ngSwitch]="g">

  <ng-template ngSwitchCase="m">i am for male</ng-template>

  <ng-template ngSwitchCase="f">i am for female</ng-template>

  <ng-template ngSwitchCase="o">i am for other</ng-template>

  <ng-template ngSwitchDefault>default case</ng-template>

</div>

.ts

g = 'ds';

output:



i am for male

**ngStyle**

The ngStyle directives lets you set a given DOM elements style properties.

We can pass dynamic via variables

For e,g <div [ngStyle]=”’background-color’:value”>Example </div>

Ngstyle is a built in directive used to set/css properties

[ngStyle]

We can also pass on dynamic values to ngStyle

* ngStyle –hands on experience

1.basic use case of ngstyle – setting value using ngstyle

    <div [ngStyle]="{'background-color':'orange', 'color':'#f00'}">this sis orange color div</div>

2.dynamic value from component

    <div [ngStyle]="{'background-color':styleprop}">this is dynamic color div</div>

Ts code

  styleprop = "purple";

3.ngStyle with conditional operators

    <div [ngStyle]="{'color': textcolor === 'blue'? 'blue' : 'green'}">This is conditional div</div>

Ts code

  textcolor = 'blue';

**ngClass**

* The ngClass drectives lets us set a class name for the element.
* We can pass dynamic values via variables

1.ngClass with sring

2.ngClass with array

3.ngClass with object

4.ngClass with component mehod

Examples:

* <div [ngClass]=”’one’”>Uisng ngclass with sring example</div>
* <div [ngClass]=”{‘one’:true, ‘two’:false}”>with multiple class names</div>

1.<div [ngClass]="'c1'">This is ngClass example</div>

2.<div [ngClass]="'c1 c2'">This is multiple ngClasses example</div>

3.<div [ngClass]="styleclsprop">This is ngClass using variable example</div>

Ts code

  styleclsprop = 'c3';

4.<div [ngClass]="condition === 'c4'? 'c1' : 'c2'">This is ngClass using variable example</div>

Ts code

 condition = 'c4';

5.<div [ngClass]="{c3:true , c2: false }">This is ngClass using object example</div>

6.<div [ngClass]="getclass()">This is ngClass using method example</div>

  getclass(){

    return 'c2';

  }

**Data Binding**

* Means to bind the data from view (Template) to controller ( component class) and vice versa
* Data binding as the name suggest – interacting with data
* Defines how the data flows and how the data gets updated based on business logic

1.One-way data binding

.Component to view

.Interpolation

.property binding

.style binding

.Attribute binding

.View to component

.Event Binding

.Two-way data binding

.Data flows from view to component and back to component from the view

1.Data is spread throughout these files

2.Data binding interacting with data of the component

3.Data can be from comoponent to template 🡪 oneway

4.Data can be from template to component 🡪oneway

5.Two way to/from component

From/to template

6.One way/Two way is nothing but representation of data flow.

**Interpolation**

* Is a technique that allows the user to bind from component to view (template)
* The data flow only one-way i.e from component to view
* Can be used for integers, strings, objects, arrays and much more
* Syntax for defining interpolation is double curly braces
* {{ Variable\_name }}

<p>{{ page\_heading }}</p>

  page\_heading = "this is interpolation";//string

<p>{{ page\_count }}</p>

  page\_count = 10;//number

<div>{{ user\_object.firstName }} {{ user\_object.lastName }}</div>

  user\_object = { 'firstName': 'manjula', 'lastName':'devarala'};//object

<div \*ngIf="isUserloginedIn">

<div>{{ user\_object.firstName }} {{ user\_object.lastName }}</div>

</div>

   isUserloginedIn = true;

<div \*ngFor="let contact of contacts">

{{contact.firstName}} {{contact.lastName}} {{contact.contactId}}

</div>

 contacts = [

    {

      'firstName':'ABC',

      'lastName':'Tutorials',

      'contactId':1234

    },

    {

      'firstName':'Mark',

      'lastName':'Hender',

      'contactId':6776

    },

    {

      'firstName':'jax',

      'lastName':'Honda',

      'contactId':2653

    }

  ]

**Propert Binding**

* Is a technique that allows the user to bind properties of elements from component to view

(template)

* The data flow data is one-way i.e from component to view
* Can be used for all properties like title,placeholder,innerHTML,src etc
* Syntax for defining interpolation is double curly braces
* [property]=”’expression’”;
* <table>
* <tr>
* <th>ContactId</th>
* <th>firstname</th>
* <th>Lastname</th>
* </tr>
* <tr \*ngFor="let contact of contacts">
* <td [title]="contact.contactId">{{contact.contactId}}{{contact.firstName }}{{ contact.lastName}}</td>
* </tr>
* </table>

<img [src]="imgurl" [alt]="imgalt">

<p [style.color]="tsxtcolor">{{ page\_heading }}</p>

<div \*ngIf="isUserloginedIn">

    <div [innerHTML]="user\_object.firstName"></div>

</div>

**Attribute Binding:**

* Is a technique that allows the user to bind attributes of elements from element to view(template)
* Attibute binding is a unidirectional – one-way data binidng
* The data flow is only-way i.e from component to view
* Can be used for any existing properties or custom attributes
* Syntax for defining binding is
* [attr.attribute\_name]=”’expression’”;

**Event Binding**

Is a technique that allows the user to bind events of elments from view/template to component .

The data flow is only one-way i.e from view to component

Can be used for all available events

Syntax for definding attribute binding is

<button (event\_name)=”function()”>Example</button>

**AngularJS Events**

* ng-blur
* ng-change
* ng-click
* ng-copy
* ng-cut
* ng-dblclick
* ng-focus
* ng-keydown
* ng-keypress
* ng-keyup
* ng-mousedown
* ng-mouseenter
* ng-mouseleave
* ng-mousemove
* ng-mouseover
* ng-mouseup
* ng-paste

<button (click)="sayhello()">click me </button>

sayhello(){

  alert("hello from arc turot");

}

<div (mouseover)="saymouseover()">on mouse over</div>

saymouseover(){

  console.log("ia m being mouseover");

}

<input type="text" (focus)="inputfocus()">

inputfocus(){

  console.log("this is onfocus");

}

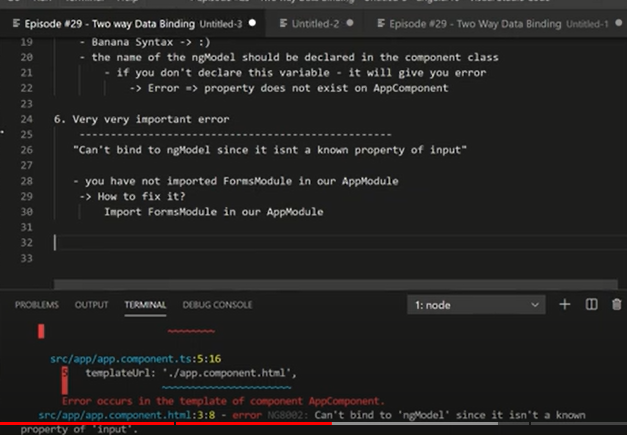
**TWO way data binding:**

* Is a technique that allows the user to bind events from view/template to component and vice versa
* The data flow is only both ways i.e from view to component and from component to view
* Two-way data binding is a combination of property binding and event binding
* We bind data using ngModel – square brackets of property binding with parentheses of event binding in a single notation
* Syntax for defining attribute binding is
* <input [(ngModel)]=’data’>

Alternative of writing ngModel

🡪property binding and event binding on the same element

E.g :<input [value]=”data” (input)=”$event.target.value” />



In add app.module.ts , practice-arc-turoial.component.ts also

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

 imports: [

    FormsModule

]

<input type="text" [(ngModel)]="username" />

<p>{{ username }}</p>

**Pipes**

1.Pipes are used to transform the data

2.Pipes will take data input and convert/transform into a desired format

3.Pipes are written using the pipe operator(|)

4.We can apply pipes to any view/template and any data inputs

5.pipe is used to transform the input data into output desired format

**Types of Pipes**

1. **Built in pipes**

**Lowercase**

**Uppercase**

**Currency**

**Percent**

**Date**

**JSON**

**2)Parametrized Pipes**

We can pass one or more parameters to pipes

**3)Chanining pipes**

We can connect multiple pipes to data input

**4)Custom pipes**

We can create our own custom pipes for various data formatting

**Pure and impure pipes**

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

A pure pipe is only called when Angular detects a change in the value or the parameters passed to a pipe. An impure pipe is called for every change detection cycle no matter whether the value or parameter(s) changes.

* Built in pipes – we use them in templates
* We can multiple pipes in the templates on element
* Built in pipes which are readily available for us to use

Syntax

**Built in pipes**

Lowercase

<div>{{ usertext | lowercase }}</div>

usertext ="YOUTUBE";

Uppercase

<div>{{ uppertext | uppercase }}</div>

uppertext = "fsebfDBAJsadaB";

date

<div>{{ datetext | date }}</div>

datetext =Date.now();

json

<div>{{ Jsontext | json }}</div>

Jsontext = {"username":"manjula", "password":"manjula"};

Currency

<h2>{{ price | currency:'INR' }}</h2>

.ts

  price = 4456;

## Output: ₹4,456.00

<div>{{ currencytext | currency }}</div>

currencytext ="20";

percentage

<div>{{ percenttext | percent }}</div>

percenttext = 2342342.322;

.html

<h2>{{ price | number:'4.3-5' }}</h2>

.ts

 price = 56;

output:

## 0,056.000

**Slice pipe:**

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

<h2>{{ firstname|slice:1-3 }}</h2>

.ts:

firstname = 'manjula';

output:

## la

.html

<h1 #t (keyup)="(0)">{{ a | uppercase }}</h1>

<h2>{{ cost | currency: 'jpy' }}</h2>

<h2>{{ price|number:'2.2-3|currency}}</h2>

<h2>{{ firstname | slice: 5 - 1 }}</h2>

<h2>{{ cdate | date }}</h2>

<p>{{ t | json }}</p>

<h2>{{ cdate | date:'dd-mm-yy' }}</h2>

.ts

 a="manimunits.com";

  cost=33.55;

  price=44.553534;

  tax='33';

  firstname="manjula";

  cdate=new Date();

output:

# MANIMUNITS.COM

## jpy33.55

## {{ price|number:'2.2-3|currency}}

## ula

## Jan 22, 2023

{ "\_\_ngContext\_\_": 295, "\_\_zone\_symbol\_\_keyupfalse": [ { "type": "eventTask", "state": "scheduled", "source": "HTMLHeadingElement.addEventListener:keyup", "zone": "angular", "runCount": 0 } ] }

## 22-44-23

**Parametrized Pipes**

We can pass one or more parameters to pipes

We pass parameters using the colon symbol(:)

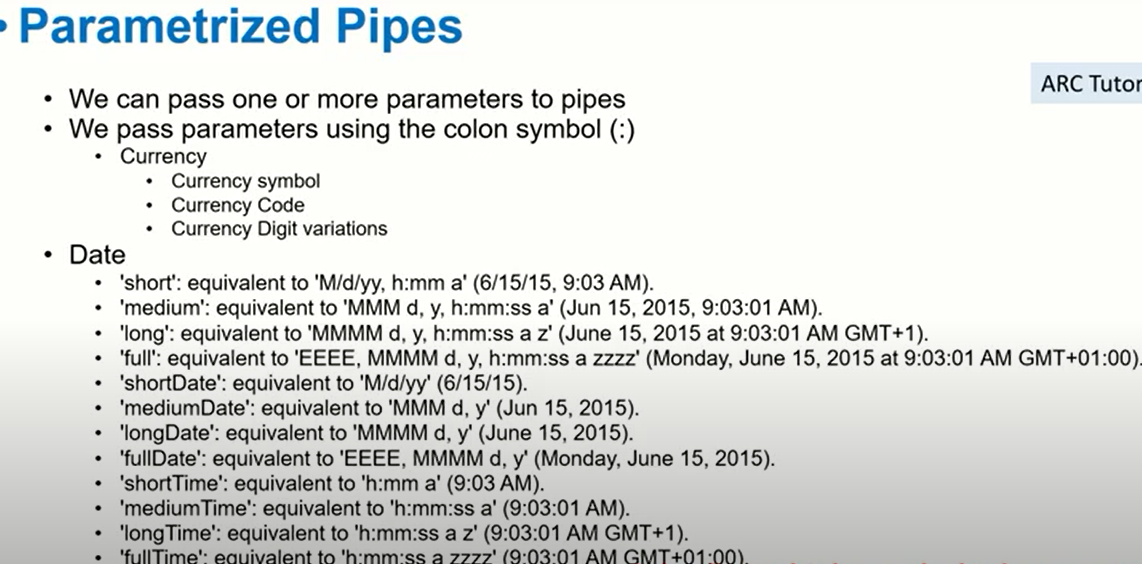
Currency symbol

Currency code

Currency digital variations

Date:

* ‘short’:equivalent to ‘M/d/yy, h:mm a’(6/15/15/9:03AM).
* ‘medium’: equivalent to ‘MMM d, y, h:mm:ss a’(Jun 15, 2015, 9:03:01







<div>{{ currencytext | currency : 'usd'}} </div>

<div>{{ currencytext | currency : 'INR'}} </div>

currencytext =20;

<div>{{ currencytext | currency : 'CAD'}} </div>

currencytext =20;

<div>{{ datetext | date : 'short'}} </div>

<div>{{ datetext | date : 'medium'}} </div>

<div>{{ datetext | date : 'long'}} </div>

<div>{{ datetext | date : 'mm-dd-yy'}} </div>

<div>{{ datetext | date : 'mmm-dd-yy'}} </div>

<div>{{ percenttext | percent : '3.1-2'}} </div>

percenttext = 2.322;

Notes:

-----------

**Percent :**

<minimumBeforeDecimal>.<minimumDecimal>-<maxDecimal>

**Chaning pipes**

<div>{{ datetext | date | uppercase }}</div>

**Custom pipes**

1)

.html

{{ firstname:mypipe}}

Step1):create pipe

Ng g p mypipe

Step2)mypipe.pipe.ts

(default)

transform(value: unknown, ...args: unknown[]): unknown {

    return null;

  }

changing:

transform(value: unknown, ...args: unknown[]): unknown {

    return ‘welcome’+value+args;

  }

Output:

welcomemanjula

**2)**

.html

{{ firstname:mypipe:’123’}}

Step1):create pipe

Ng g p mypipe

Step2)mypipe.pipe.ts

(default)

transform(value: unknown, ...args: unknown[]): unknown {

    return null;

  }

changing:

transform(value: unknown, ...args: unknown[]): unknown {

    return ‘welcome’+value+args;

  }

Output:

Welcomemanjula123

**3)**

<div id="m1">

  <table class="table table-stripped">

    <tr \*ngFor="let student of students|mypipe:'2'">

      <td>{{ student.firstname }}</td>

      <td>{{ student.lastname }}</td>

      <td>{{ student.age }}</td>

    </tr>

  </table>

</div>

 students = [

    {

      firstname: 'manjula',

      lastname: 'devarala',

      age: 25,

    },

    {

      firstname: 'manju',

      lastname: 'devarala',

      age: 25,

    },

    {

      firstname: 'anjula',

      lastname: 'devarala',

      age: 25,

    }

  ];

 transform(value: unknown, ...args: unknown[]): unknown {

    var temp=[];

    for(let i=0;i<args;i++){

      args.push(value[i]);

    }

    return temp;

  }

**Angular Router**

* Routing is a mechanism used by angular framework to manage the “paths” and “routes” of our angular applications.
* Routing strategy helps In navigation between various views in our angular application
* Angular framework comes with “router”Module which has everything we need to design, develop and implement routes and navigation links
* Router is a singleton – which means there is only one instance of the router in our application
* Angular router is the official router module which is written and maintained by core angular team
* The router module
* Is found in the package @angular/router
* We need to setup Router array – every time a request is made , the router will search in the list of array and find the most relevant match
* Router has states which helps us get important information about the current state and data related to routes
* All batteries included for router.

**Routing in Angular**

We can handle various types of routes in angular app

* Routes for components
* Getting query params from routes
* Getting the URL segments
* Loading child routes for a module
* Lazy loading
* Handling wild card routes
* Handling default routes
* Handling 404 route

All batteries included for router

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

 imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

**Home:** <http://myapplication/com/>

**Search:** <http://myapplication/com/search?user=abc> 🡪query params

**Profile:** <http://myapplication/com/profile> --> component Routing

**Tasks:** http://myapplication/com/tasks/10/category/pemding 🡪URL segments

**Users:** <http://myapplication/com/users> 🡪module

View-user 🡪 <http://myapplication/com/users/view/10> 🡪child routes

edit-user 🡪 <http://myapplication/com/users/edit/10> 🡪child routes

add-user 🡪 <http://myapplication/com/users/add> 🡪child routes

manage-user 🡪 <http://myapplication/com/users/manage> 🡪child routes

PageNotFound 🡪http://myapplication.com/pageNotFound 🡪404 error 🡪No matching routes

**Routes for components**

Each component can have its own Routes

Various example of components routes are:

* /products
* /products/view
* /products/add
* /users

import { LoansTypesComponent } from './loans-types/loans-types.component';

import { LoansComponent } from './loans/loans.component';

  {

    path:'loans',

    component:LoansComponent

  },

  {

    path:'loans/add-loan',

    component:AddLoansComponent

  },

  {

    path:'loanstype',

    component:LoansTypesComponent

  },

**Router-Outlet**

1. The router-outlet is a directive that’s available from the router library where the router inserts the component that gets matched based on the current browsers URL.
2. You can add multiple outlets in your angular application which enables you to implement advanced routing scenarios.
3. By default there is always one router outlet – in app.component.html
4. Router-outlet is a built in directive
5. Every angular app should have atleast 1 router outlet
6. By default – the router outlet is defined in app.component.html
7. Router outlet will matches the routes for the componets

🡪Takes its output

🡪inside the DOM

8)Multiple router outlets in application

🡪We can have more than 1 router outlet

**Multiple Router-Outlets**

* The router-outlets is a directive that’s available where the router insert the component that gets matched based on the current browser’s URL.
* You can add multiple in your angular application which enables you to implements advanced routing scenarios.
* By default – there is always a router-outlet and it’s treated as “primary”
* We need to define named router outlets
* Examples of declared multiple router outlets

{

path:'add',

component:AddLoansComponent,

outlet:'router1'

}

<http://localhost:4200/loans(router1:add)>

notes:

we cant access directly path

why because its talking router-outlet primary it will take

so how to get directly means

<http://localhost:4200/add(router1:add)>

Example:

<router-outlet name="addLoan"></router-outlet>

 {

    path:'add',

    component:AddLoansComponent,

    outlet: 'addLoan'

  },

Output:

<http://localhost:4200/loans(addLoan:add)>

loans works!

add-loans works!

Syntax:

[http://localhost:4200/<primary-router>(<routerOutletName>:<secondaryPath>)](http://localhost:4200/%3cprimary-router%3e(%3crouterOutletName%3e:%3csecondaryPath%3e))

Why are using this multiple routers?

-Avoid this use case in applications

-you can inject components

**Routing strategy:**

Before we start implementation our routes in our application, its important to understand and plan what will be our routing strategy

**Import { LocalStrategy } from ‘@angular/common’;**

We need to add this in providers of our module

**{provide:locationStrategy, useClass:HashLocationStrategy}**

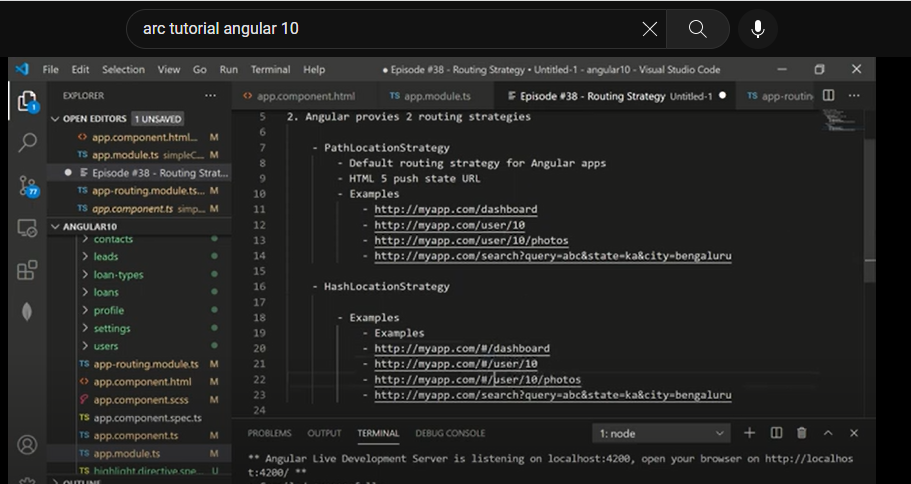
Angular providers 2 types of running stretagy we can use:

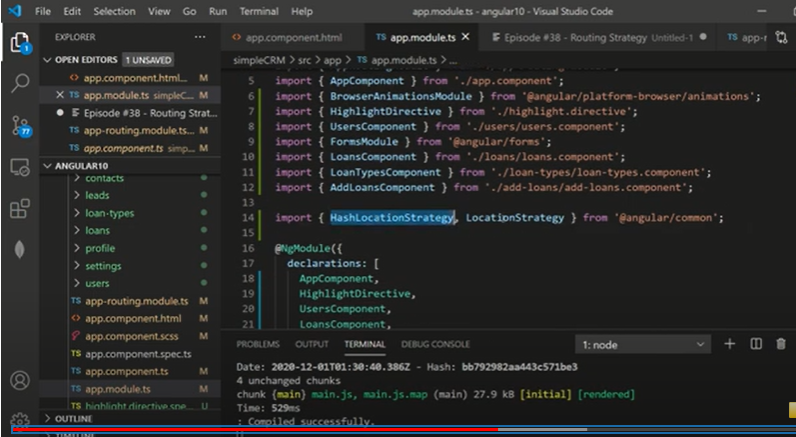
.pathLocationStrategy

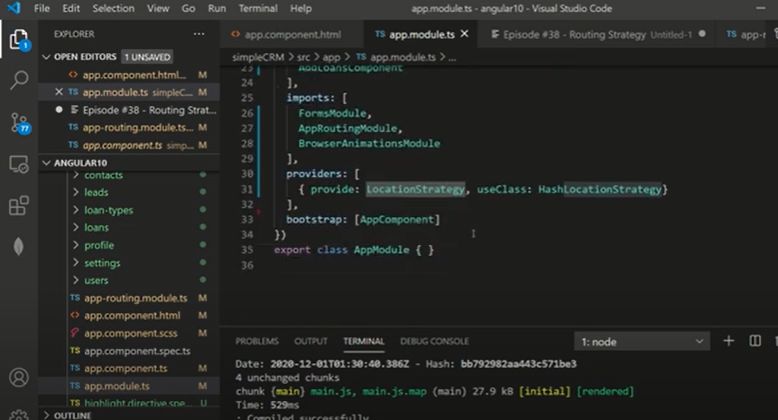
.HashLocationStrategy

Bydefault –Angular makes use of the pathLocationStrategy

With **HashLocationStrategy** –we will see # in the URL





 ­­­­

1.)Handson examples for pathLocationStrategy

Default behaviour of angular apps

2.)Hands on example for HashLocationtrategy

We need to import HashLocationtrategy from @angular/core

Add it providers array­

Angular will start loading our URLsusing #

3.)Why do we need 2 different types of routing ?

Angular is aSPA(single page app)

Index.html

[**http://localhost:4200/#/**](http://localhost:4200/#/) **(url)**

1)Which one you should use when?

Really there is no difference affect your application

PathLocationtStrategy

Clean URLs

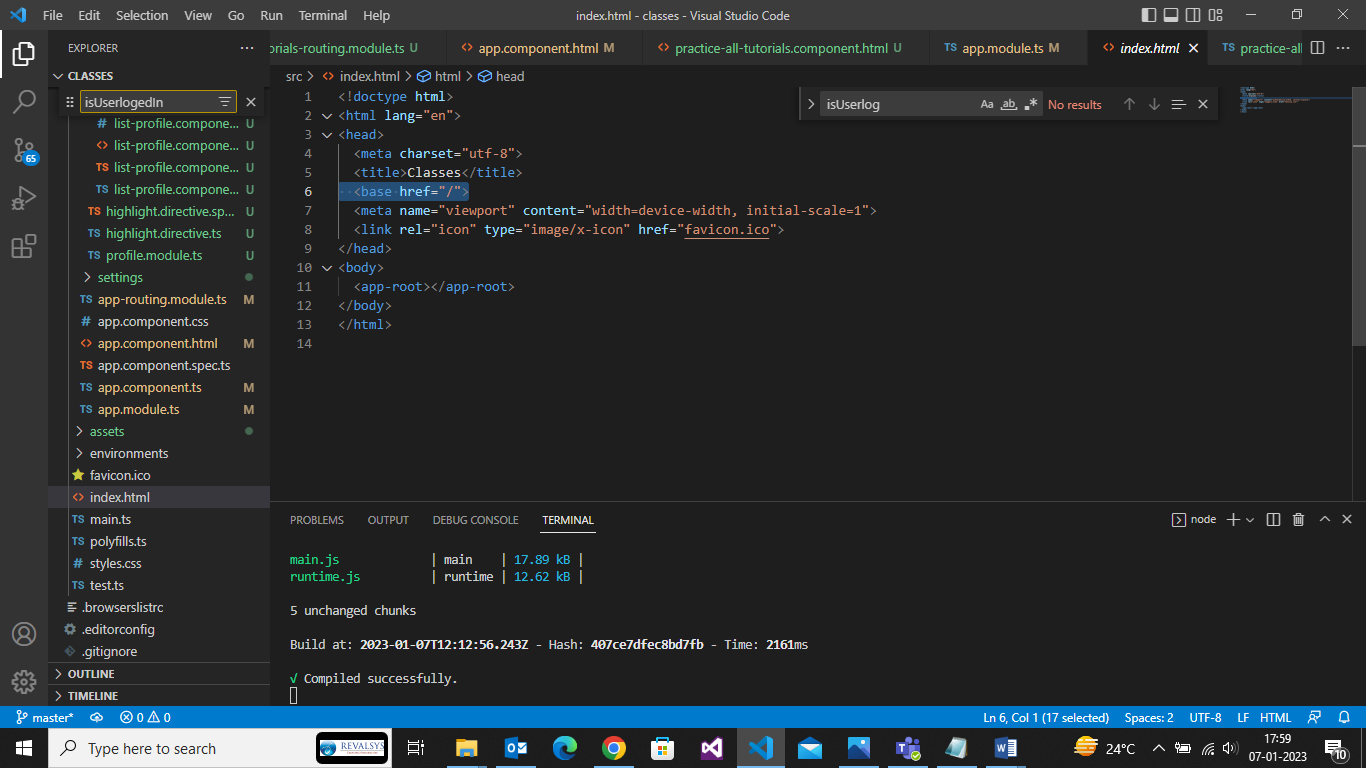
Simple

Bookmarbale

Easy to remember

**Routing Base Href:**

* Every angular application leads to pointing to wrong folder root path
* Setting the base href using the command line- base-href=
* Syntax
  + <base href=”/”>



<base href="/"> <!—default shown 🡪

<base href="/app1">

http://localhost:4200/app1/

**Router Module**

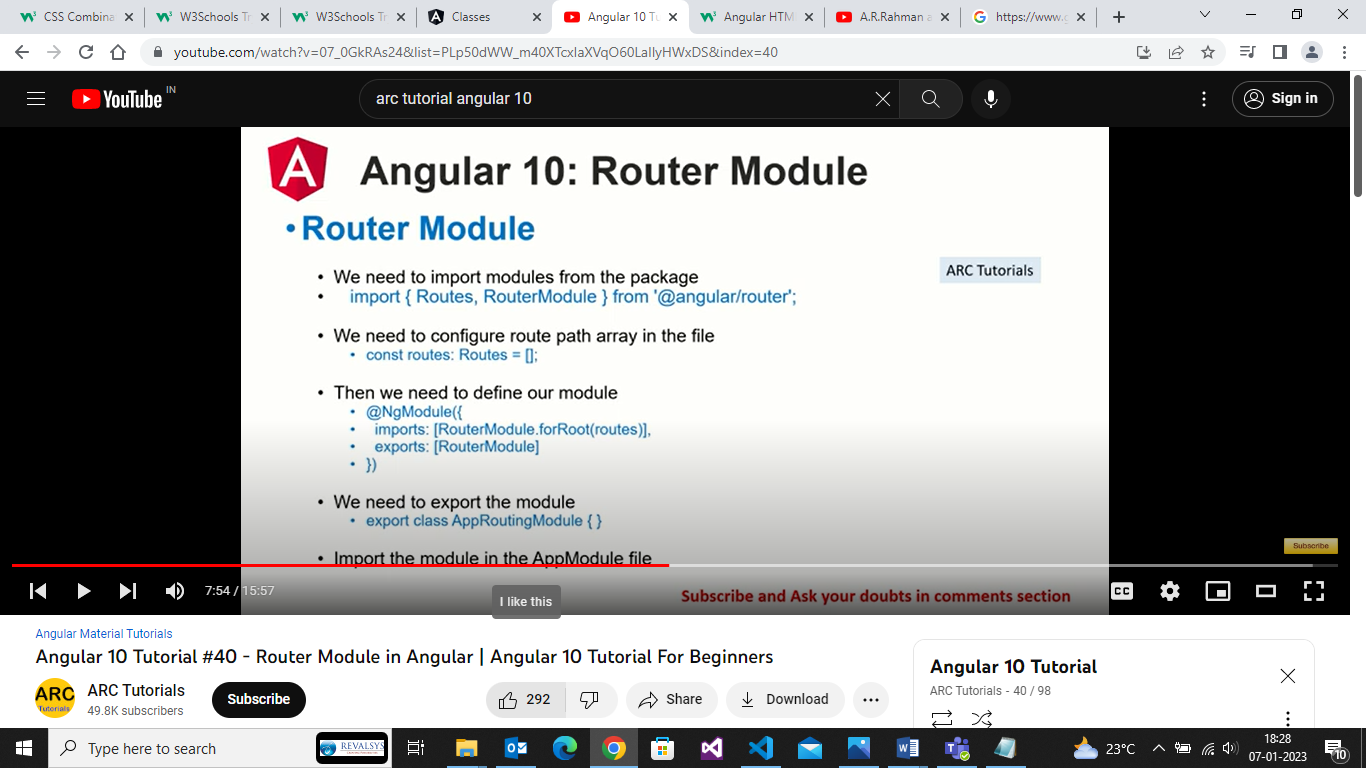
* RoutingModule is a placeholder for configuring all routes and navigations in one module
* Best practice is to have all routes configured in one place
* Easy to maintain and debug
* We can the app routing module using the CLI

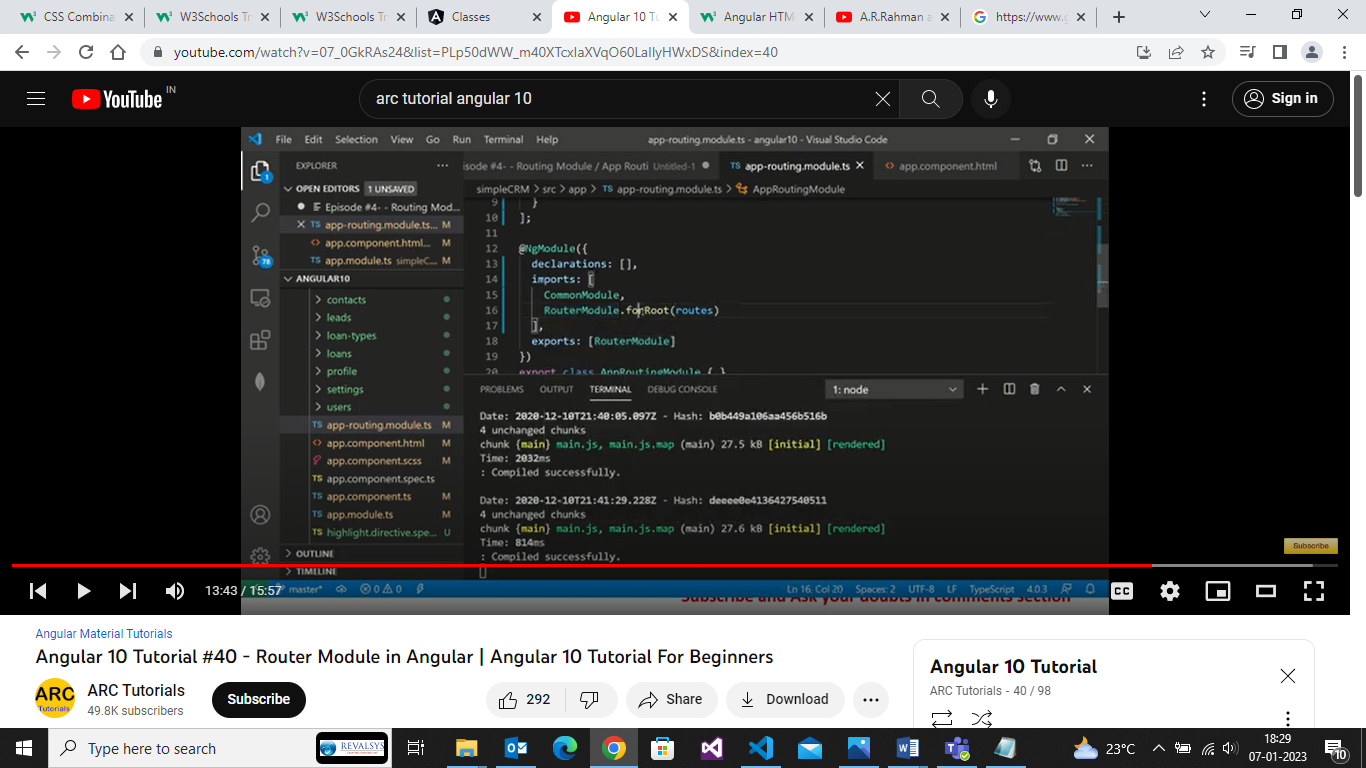
**ng generate module app-routing –flat –module=app**

**Remove app-routing-module.ts directly from project**

**After install**

**ng generate module app-routing –flat –module=app**





**Component Routes:**

We can configure routes to redirect route for various paths

* Path
* Component
* RedirectTo
* Children
* Outlet
* pathMatch

Let’s learn how to configure routes in the routing module

  {

    path:'add',

    component:AddLoansComponent,

    outlet: 'addLoan',

    children:[

      {

        path:'add-loan',

        component:AddLoansComponent

      }

    ]

  },

  {

    path:'add-new-loans',

    component:LoansComponent,

    redirectTo:'add-loan'

  },

**Parametrized Routes**

* We can configure and send parameters to our routes
* We need to configure the route and mention that the value is dynamic
* { path:’product/:id’, component:’componentName’}
* For e.g
* Product/10
* Product/10/20
* We can read the values in the component class and process the paramters.
* We can send dynamic data or parameters
* URLs will look something like this
  + <http://localhost.com/user/10>
  + <http://localhost.com/search/ka/bangalore> -->state and city
  + <http://localhost.com/user/10/photos/34> --> user id -10 and id =34

app-routing.module.ts

{

    path:'product/:id',

    component:ProductComponent

  }

Product.component.ts

import { ActivatedRoute} from '@angular/router';

private activatedRoute: ActivatedRoute) {

    this.activatedRoute.params.subscribe((params) =>{

      console.log(params);

    } )

app-routing.module.ts

 {

    path:'product/:id',

    component:ProductComponent

  },

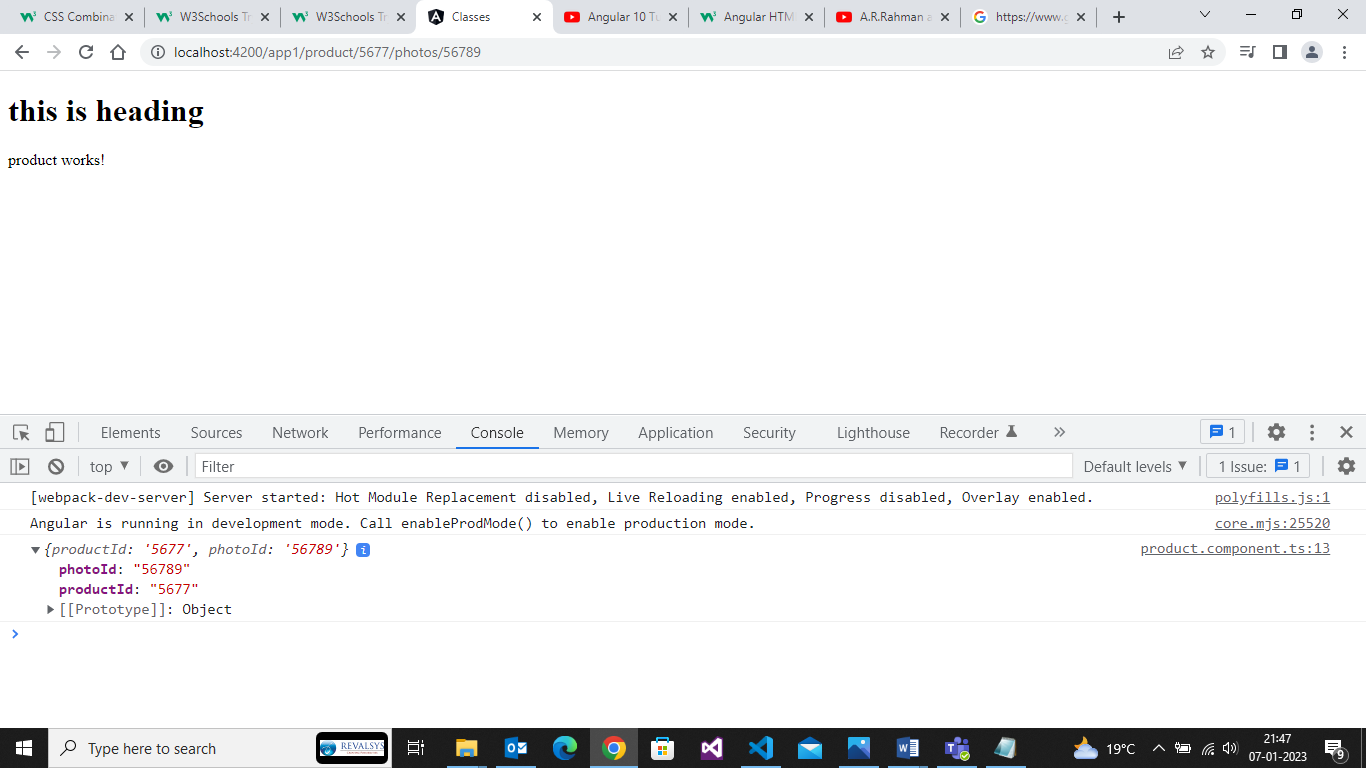


 {

    path:'product/:productId/photos/:photoId',

    component:ProductComponent

  },



**RouterLInk**

* When applied to an element in a template, makes that element a link that initiates navigation to a route
* Navigation opens one or more routed components in one or more <router-outlet> locations on the page.
* For e.g
* <a [routerLink]=”[‘/user/bob’]”>somelink</a>

**Router Redirect**

* Whenwe want a route to redirect to another route – we will implement the redirect in our routes array.
* Thesyntax to define the same is given below

.{path:”,redirectTo,:’home’,pathMatch:’full’},

* The empty path indicates that it’s the **default route** of the application
* The empty path also requires us to mention that **pathMatch** should be “**full”**
* Lets learn how to redirect route in the routing module

{

Path:’’,

redirectTo:’home’,

pathMatch:’full’

}

**Query Params**

* We can configure and send query parameters to our routes.
* Search?keyword=toys&country=usa
* We can read the values in the component class and process the parameters.
* We can send data from form🡪
* We can have data from click
* Basiically 🡪 URL <http://localhost.com/search?key=10&state=ka&ct=bangalore>
* Query params 🡪visible in the URL
* Most used for querying , searching or filtering data etc
* Facebook.com/search?page=10&pagesize=20

 {

    path:'search',

    component:SearchComponent

  },

import { ActivatedRoute } from '@angular/router';

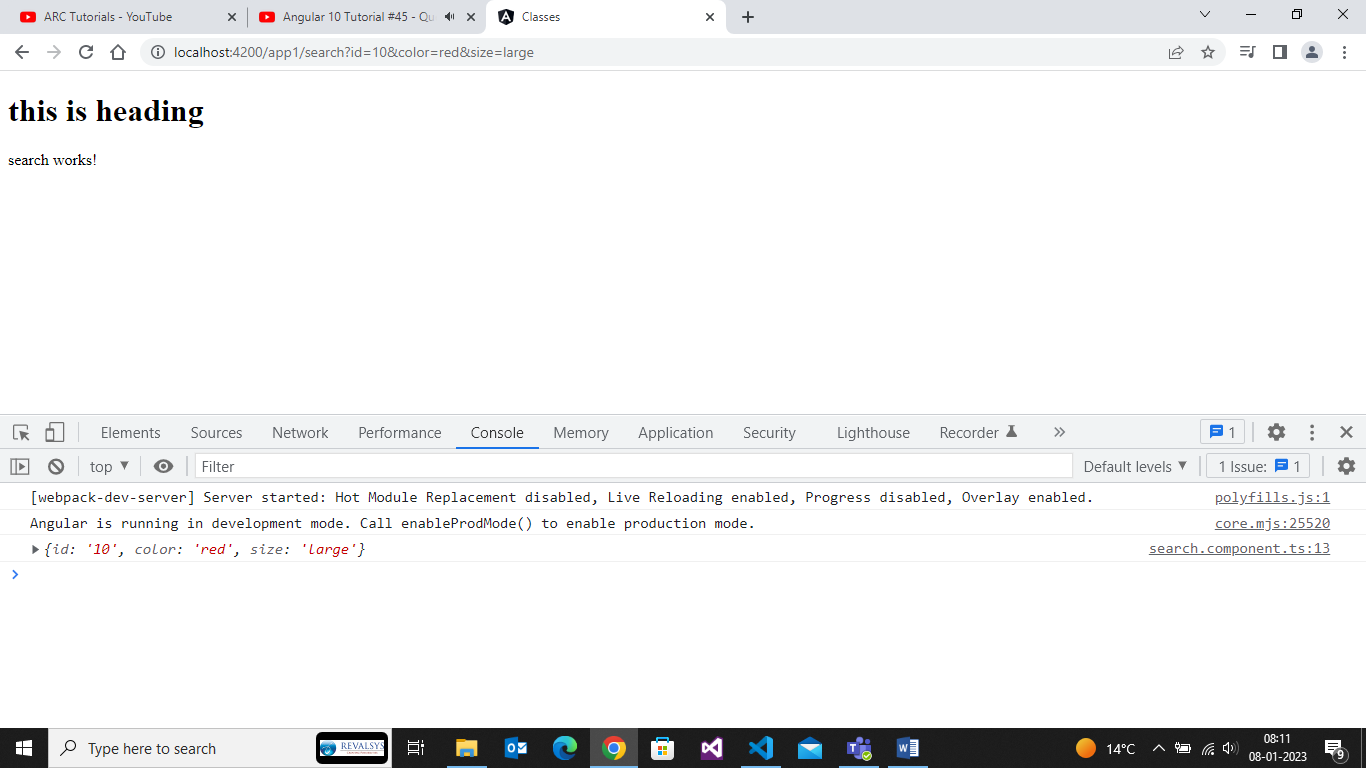
 constructor(private activatedRoute:ActivatedRoute) {

    this.activatedRoute.queryParams.subscribe(params =>{

      console.log(params);

    })

   }



**Routing – Wildcard Routes**

* Wild card intercepts any invalid URLs in our application
* When NO matching routes are found in the routes array, the router does not know where to go and hence results in console errors.
* Wild card routes are defined in the routes array using
* {

Path:’\*\*’,

}

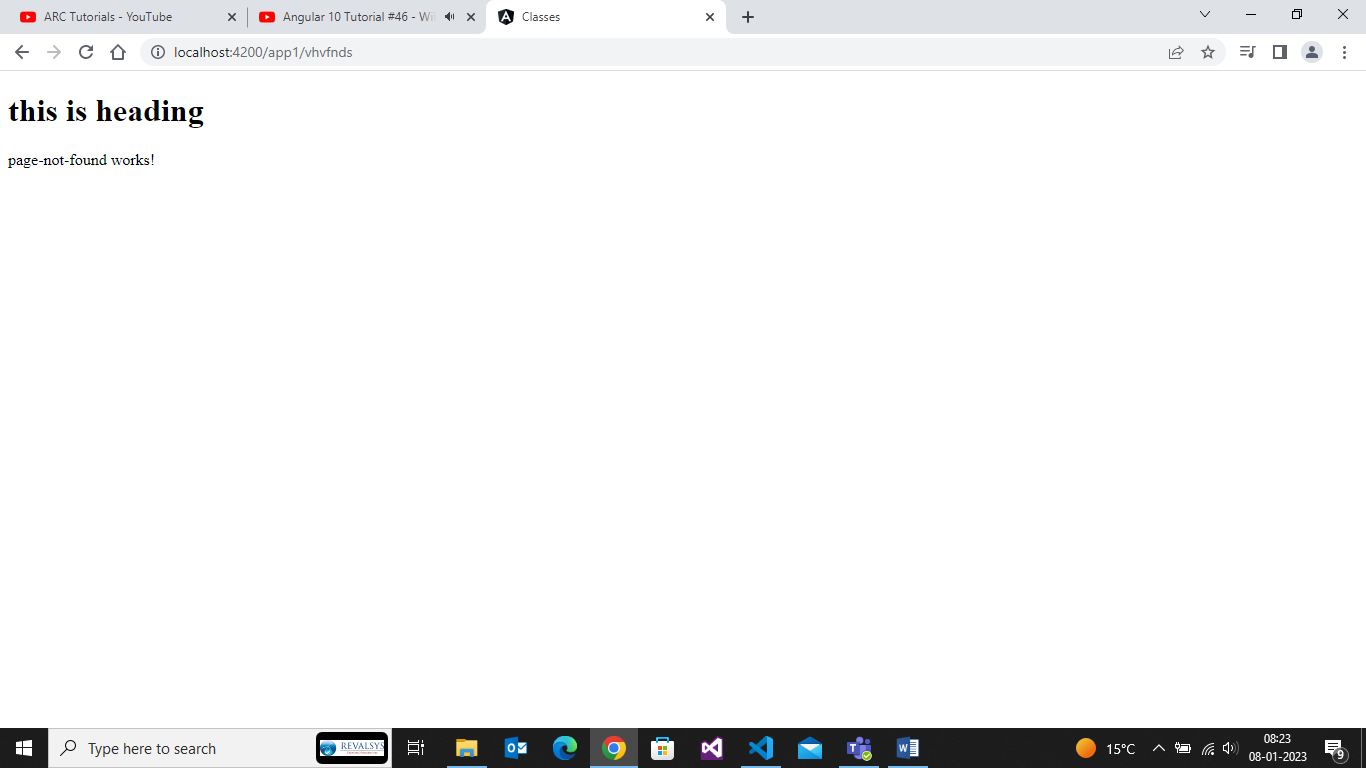
* Usually a component named pagNotFound is mapped as best practice
* Lets learn how to use wildcard routes in the routing module.
* Any unmatched route will be intercepted by wild card route
* This has to be last route in your application

{

  path:'\*\*',

  component:PageNotFoundComponent

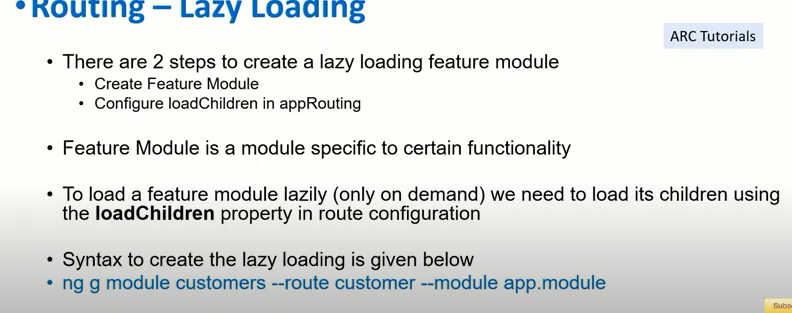
}



**Lazy Loading**

* By default , **NgModules** are eagerly loaded, which means that as soon as the app loads, so do all the **NgModules**, whether or not they are immediately necessary.
* For large apps with lots of routes , consider lazy loading – a design pattern that loads NgModules as needed.
* Lazy loading helps keep intial bundle sizes smaller ,which in turn helps decrese load times .
* From Angular 8 , **load children** expects a function uses the dynamic import syntax to import your lazy-loaded module only when it’s needed.
* Angular default will load all modules at start
* -Login
* Lazy Loading comes into picture
  + Intially we will load only modules which are mandatory
  + Rest we will serve as “required”
  + We will load fast
  + It will respond better
* If you are coming from previous version of angular 8
  + The syntax has changed
  + Please use the expects a function
* The modules generated using theangular CLI – for lazy loading
  + There will be No entry in AppModule
  + Hence, It will not be loaded initially
* Any angular applications is made up of multiple modules
  + Loan management system
    - Loans
    - Customers
    - Payments
    - Invoices
    - Reports
    - Authentoction
    - Downloads
    - Admin
* Ng g module <module\_name> --route <module\_route> --module app.module

ng g m payments --route payments --module app.module



Note:

* Initially we will load only modules which are mandatory
* Rest we will serve as “requsted”
* It will save performance app
* We can verify if the user has access to this module.

**Routes Guards**

* Use route guards to prevent users from navigating to parts of an appwithout authorization.
* Route Guards are used to secure the routes paths
* In most cases , the routes and screens are protected behind a good authentication system.
* The route guard resolves to true or false based o custom logic and functionality.
* We can generate any number of guards based on our application requirements.
* Route Guards have something called “interfaces”.
  + canActivate 🡪can user access a route
  + canActivateChild 🡪can user access child routes of a parent route
    - /payments
      * /payments/process
  + canDeactivate 🡪check if user can exit the route
  + canLoad 🡪can a lazy loaded module be loaded
  + resolve 🡪route date retrival before route activating
* Generate Route Guard
  + Ng g guard <guard-name>
* E.g
  + User 🡪Route Guard 🡪Admin
    - Custom Logic
      * True
        + Can access the routes
      * false
        + Custom logic on failure condition 🡪home

1)Inject the guard in our module under providers

2)There are various types of route guards available

**canActivate :**Checks to see if a user can visit route

**canActivateChild:**Checks to see if a user can visit a routes children

**canLoad:**Checks to see if a user can route to a module that lazy loaded

**canDeactivate:** Checks to see if a user can exit a route

**Resolve:** Performs route data retrieval before route activation

The route guard resolves to true or false based on custom logic and functionality

**canActivate**

* Use route guards to prevent users from navigating to parts of an app without authorization.
* Route Guards are used to secure the routes the route paths
* In most cases, the routes and screens are protected behind a good authentication system
* The routes resolves to true or false based on custom logic and functionality
* We can generate any number of guards based on our applications requirements.

**Def:**

A **canActivate** guard is useful when we want to check on something before a component gets used.

Syntax

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

1.)ng g guard admin(<guard\_name>)

2.)choose the option CanActivate

3.)In the routing module -> we will use option canActivate

->it will resolve to true or false

-> true means ->user can access route

🡪false means->user cannot access route

4.)We can use any number of route guards on canActivate

->Its an array

->All have to resolve to true

5.)use cases

1.check if user is loggedIn

2.chceck if user can edit the product/order/details/profile

3.check if the user is an admin

 {

    path:'loans',

    component:LoansComponent,

    canActivate:[AuthGuard]

  },

  {

    path:'loans',

    component:LoansComponent,

    canActivate:[AuthGuard ,AdminGuard]

  },

**CanActivateChild**

**----------------------------**

The **CanActivateChild** guard works similarly to the canActivateChild guard,but the differnet is it runs before each child route is activated

{

    path:'admin',

    canActivateChild:[AdminAccessGuard],

    children:[

      {

        path:'',

        component:AdminComponent,

        canActivate:[SuperAdminGuard],

      },

      {

        path:'manage',

        component:AdminManageComponent

      },

      {

        path:'edit',

        component:AdminEditComponent

      },

      {

        path:'delete',

        component:AdminDeleteComponent

      }

    ]

  },

**CanLoad**

**---------------**

1.Generate a lazy loading Module

Ng g module customers –route customers –module app.module

2.canLoad

This protects the route completely .Such as lazy loading the module and also protects all the routes associated with that module.

ng g module prederences --route settings --module app.module

**Forms-Introduction**

**---------------------------**

Forms are very integral and essential building blocks of “almost” apps

Common form examples we can see are

Login

Forgot

Register

Checkout form

Contact us

Forms allows us to gather information and data from users

Good way to interact with the users and almost all websites wil need forms in some or other way

We can use any CSS framework of our choice – Bootstrap or Material Design

**Angular Support for Forms**

**-----------------------------------------**

* Two data binding
* Change Tracking
* Validations
* Error Handiling
* Unit testing

**Types of Forms in angular**

1.)Static /Template driven forms

2.)Dynamic /Reactive Forms

**Static or template drive forms**

* Easy to use
* Template driven forms are simple and straight forward
* All the validators, form elements are all defined in the template file
* We will need to import Forms Module in app module to work with template driven forms.

**Dynamic or Reactive or Model Driven Forms**

* All the form elments , user interactions and validations are handled in the component class
* We will make use of angular’s built in **formGroup** and **FormControl.**
* Can control better data binding
* Exclusive define custom regular expression patterns of error handling
* We will need to import **ReactiveFormsModule** in ourapp module
* Very flexible and allows users to define, develop complex rewuirments of forms
* More logic in the componet class and less in HTML mark up itself

**Which is better-template driven forms or reactive forms?**

**Template driven forms:**

* If your application forms are simple straight forward
* Fixed static form fields and elements
* No complex validations or pattern matching

**Reactive forms:**

* If your application forms are complex
* Uses multiple dynamic components
* Advanced validation requirements
* Dependent form elements
* Dynamic form generation based on user prefernces

**Template Driven forms - introduction:**

* Easy to use
* Template driven forms are simple and straight forward
* All the validations, form elements are all defined in the template file
* Forms are tracked automatically
* Tracked fprm datd traverses via various states (pristine etc)
* Uses two-way data binding techiniques to bind datad
* Most of the code resides in template file
* Validations are mostly the default HTML5 validations
* mINnimal component code as most of the code is in template file
* Unit testing will be a challenge

**Step by step process for Template driven forms**

* Step #1 – import and add in the formsModule in the app.module.ts

1. For template driven forms –FormsMOdule needs to be imported
2. If we do not imports – we will get error when doing two way data binding
3. Add the module into the array list of imports

* Step #2 – Create the form in app.module.component.html

1. ngForm:

Form names as template vaiable using “#” for e,g #loginForm

1. ngModel:

Every form field should have a “name” attribute and ngModule attached to it.

**Add different form input types**

* Input type = “text”
* Input type = “radio”
* Input type = “checkbox”
* Input type = “email”
* Textarea
* Seleect Drop Down
* <form #addCustomerForm="ngForm" (submit)="addCustomer(addCustomerForm)">
* <input type="text" [(ngModel)]="firstname" name="firstname" id="">
* <br>
* <input type="checkbox" [(ngModel)]="terms" name="firstname" id="">
* <select name="" id="" [(ngModel)]="terms">
* <option value="">gdgg</option>
* <option value="">gdgsdg</option>
* <option value="">gdgsdjdg</option>
* </select>
* <button type=submit>Add Customer </button>
* </form>
* <p>{{ firstname }}</p>

  firstname: string = '';

  addCustomer(formValue:NgForm){

    console.log(formValue);

  }

  terms = false;

**Adding Validations in Template Driven Forms**

**Disable the form**

.Disable the form if the form is not valid

**HTML5 validations**

Required

minLength

maxLenght

checked

**Reset Forms:**

**Reset the form using**

**.**Restmethod

**.**Form.reset();

<form #addCustomerForm="ngForm" (submit)="addCustomer(addCustomerForm)">

    <input type="text" [(ngModel)]="firstname" name="firstname" id="" required="">

<br>

    <input type="checkbox" [(ngModel)]="terms" name="firstname" id="" required>

    <br>

    <select name="" id="" [(ngModel)]="terms" required=" " minlength="10">

        <option value="">gdgg</option>

        <option value="">gdgsdg</option>

        <option value="">gdgsdjdg</option>

    </select>

    <br>

    <textarea name="" [(ngModel)]="description" id="" cols="20" rows="10"></textarea>

    <br>

    <button type=submit [disabled]="!addCustomerForm.valid">Add Customer </button>

    <button type=submit (click)="resetForm(addCustomerForm)">Reset form </button>

    <button type=submit (click)="loadValues(addCustomerForm)">set form </button>

</form>

 firstname: string = '';

  description:string = '';

  addCustomer(formValue:NgForm){

    console.log(formValue);

  }

  terms = false;

  resetForm(formValue: NgForm){

    formValue.reset();

  }

loadValues(formValue: NgForm){

  let userDetails = {

    firstname: 'ARC',

    terms:'false',

    customerType:'premimum',

    description:"this is set value"

  }

  formValue.setValue(userDetails);

}

**Reactive Forms:**

* Reactive forms are a way to create forms in angular application
* Whats diffenet is how we implement, design and handle the form and the data
* All the from elements , user interactions and validations are handled in the component class
* We will make use of angular built in FormGroup and FormControl
* Uisn recative forms we can control better data binding
* Exclusive define custom regular expression patterns of error handling
* Veruy flexiable and allows users to define ,develop complex rewuirmensts of forms
* More logic in the component class html and less in html mark up itself
* Angular maintains the state information of forms at all times
  + Ng-touched
  + Ng-untouched
  + Ng-dirty
  + Ng-pristine
  + Ng-valid
  + Ng-invalid

**Reactive forms – misconception**

* Reactive forms are tough
* Reactive forms are very complex
* Reactive forms are difficult to learn and implement
* Reactive forms are only for “complex” applications
* Adding custom are tricky inreactive forms

Reactive forms

Form control

Form Group

Form Builder

**HttpClent**

* HttpClient request is used for performing HTTP requests and responses.
* The HttpClient service is available in the @angular/common/http package
* The new HttpClient service is included in the Http Client module which is used to intiate HTTP request and responses in angular apps
* The HttpModule needs to be imported into the module . Usually in the app module.
* The HttpClent also gives other useful functionality like params, interceptors, headers etc.

**Https Methods**

* get() 🡪we are request the resources from the api
* post() 🡪we are trying send the data to the api
* put() 🡪trying to update an existing resources
* delete()🡪delete
* head() 🡪
* jsonp() 🡪
* options()🡪
* patch()

**Benefits Https Client**

* HttpClient includes observable APIs
* HttpClient can have the HTTP headers in options
* HttpClient includes the highly testability features
* HttpClient includes typed request
* HttpClient includes response objects
* HttpClient includes request and response interception
* HttpClient include error handling

**Standard CRUD Operators**

* Create – usually a POST method call
* Read – usually a GET method call
* Update –usually a PUT method call
* Delete – is a delete methods call

Technically – we should NEVER delete anything. Only soft deletes which means setting a flag or so.

* HttpModule is used for communicating between frontend app and backend APIs or restends points
* HttpClentModule in our appModule
* We need to inject httpCleint in component/services and create its instance in constructor
* Mostly crud operators
  + Create new cutomer
  + Edit customer
  + Read customer
* HTTP mETHODS
  + POST
  + GET
  + PUT
  + DELETE
* HTTPClent for sending

<https://jsonplaceholder.typicode.com/>

1)    What is a prototype chain ?   
2)    Explain Prototypical Inheritance with a example?  
3)    What is the difference between Call, Apply and Bind, with an example?  
4)    What is Hoisting and why it happens ?  
5)    What are closures, explain with an example?  
6)    What is a promise, explain with an example?