Angular Framework

Angular Framework is used to develop front end applications, it allows you to create applications which are called as client applications for variety of platforms like Mobiles, Desktops, Web and so on.

It is created by Google.

What is framework

Framework helps to build complex application easily by providing all the common features for an application inbuilt so that developers don’t have to write code for these common features, some of the common features most of the application needs are:

* Type conversion
* Internationalization
* Exception Handling

Angular Framework is a framework for front-end it uses two languages to develop the front end applications & it is used to develop single page application the two languages are:

1. HTML for displaying
2. Typescript for writing the logics

Typescript is a superset of JavaScript which is better than the Java Script

Note: Angular Framework uses CSS for styling

Single Page Application:

Single Page Applications will have one single page where you can perform all the tasks, it loads only part of the page when any action is performed and doesn’t refresh the entire page, ex: Gmail, Angular official website, React.js official website, Git Hub.

Single Page Application is much faster compare to multiple page applications, because in single page application you need to update only small part of the page not the entire page.

Software required

1. Node.js
2. Visual Studio Code
3. Angular CLI (Angular Command Line Interface) toolkit

Angular CLI

It is a toolkit released by Google to improve the development process

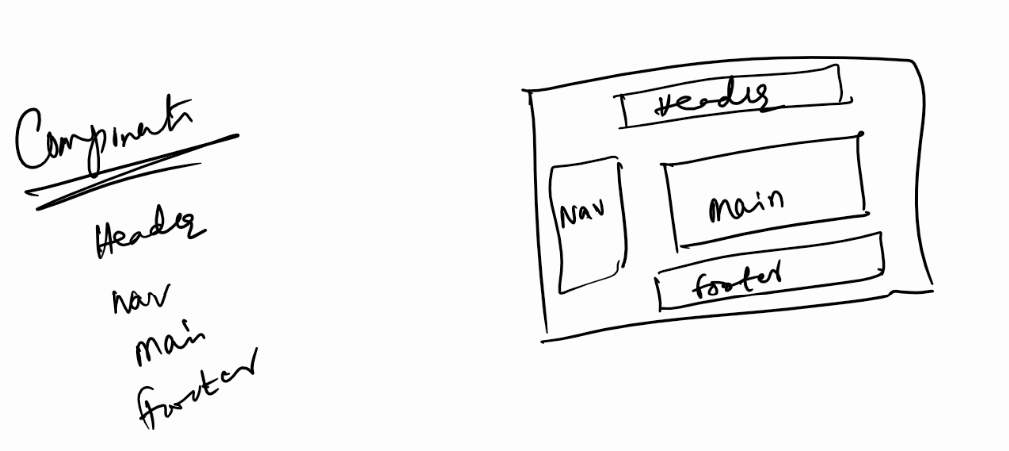
* It allows you to create a ready to run and build angular project
* It will have an inbuilt typescript compiler in the project which automatically compiles the typescript code to javascript code
* It will provide a live reload feature which will refresh the browser while you are typing the code
* It will provide an embedded server to run your angular application
* It will provide lot of angular commands to accelerate the development process, so that developers work will be reduced

How angular can create a single page application

Angular uses component to create the UI, these components are independent and reusable with other components.

Components: These are UI’s which you see in the web page, they can be created independently and nest in other components.

A single page application is a collection of multiple components



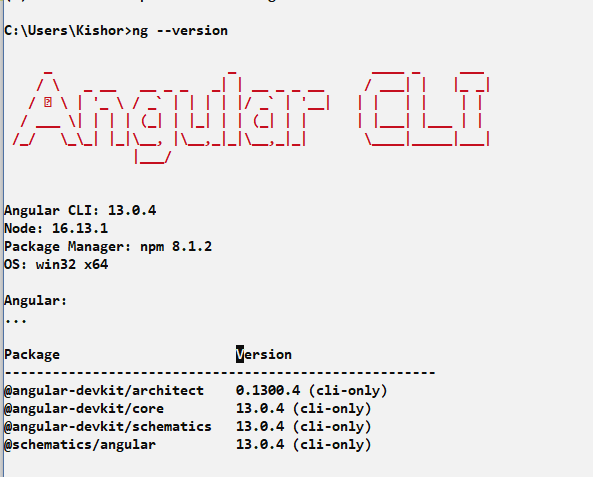
Angular CLI Installation

We should have node & npm installed already, because you will install angular cli using npm.

>> npm install -g @angular/cli

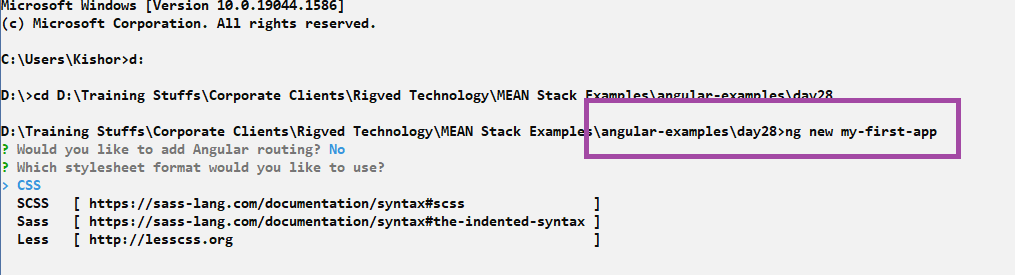
Note: Angular CLI is a one time installation, it gives you a command called ‘ng’ using which you can create angular applications & also run angular applications

After installing you can use ng --version

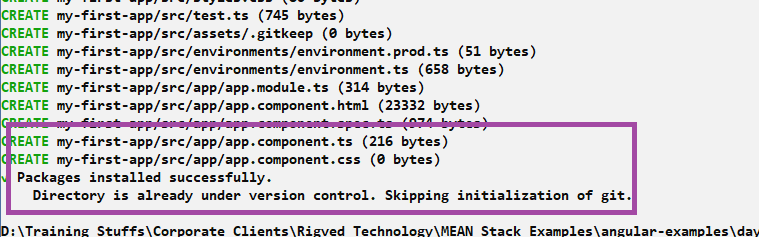


Command used to create angular project

>> ng new project-name



Once you install you will get the success confirmation



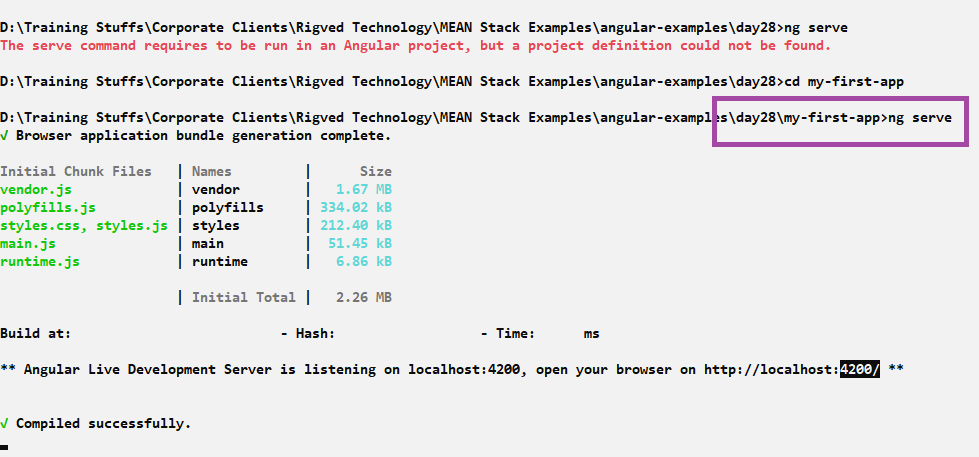
Purpose of *ng new <<app-name>>*

ng is an angular command

new is a keyword to create a new project

<<app-name>> must be the project name

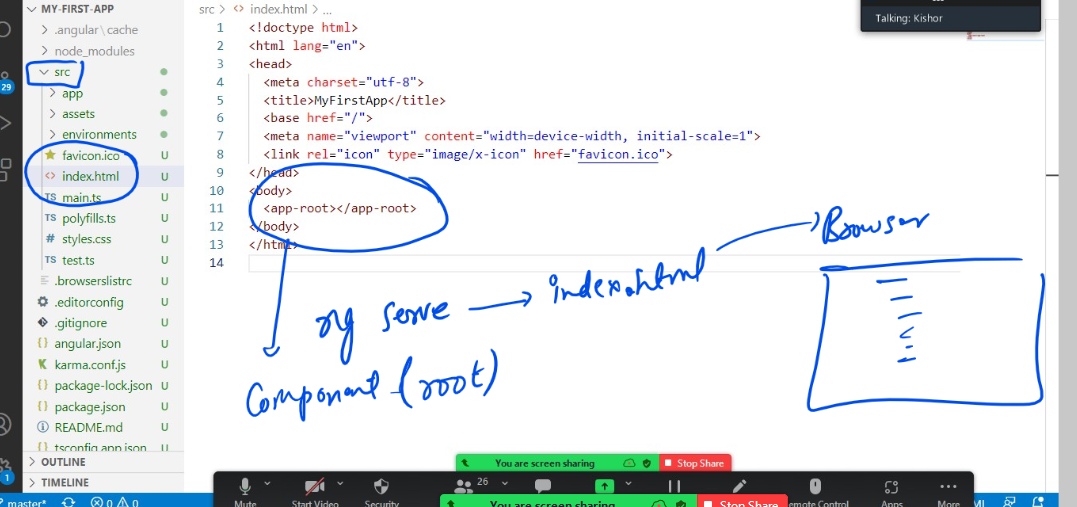
Navigate to the project & use ng serve to run your project



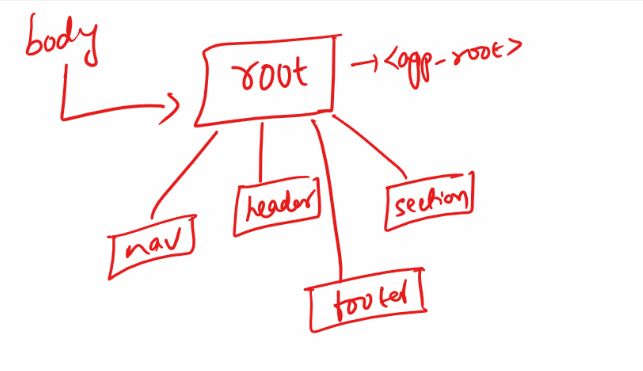
By default the project runs in 4200, which you can enter in the browser and by default every new project will have some default content which can be changed by adding UI’s in angular application according to your requirement.



You can open angular project folder from the VS Code



The moment you enter ng serve, angular loads index.html in the browser, that is only file which is loaded in the browser, but everything happens inside the index.html <app-root>, which is a root component, this root component can nest any number of components



Every angular application must have one root component and it be only one root component all the other components are part of this root component.

<app-root>: This is the tag name for root component, when you use this tag the content of root component is displayed

Where’s the content of this <app-root>

src/app/app.component.html: This file has the content for <app-root>

How does angular knows <app-root> content is in app.component.html

It is mentioned in the src/app/app.component.ts

Two important files to know

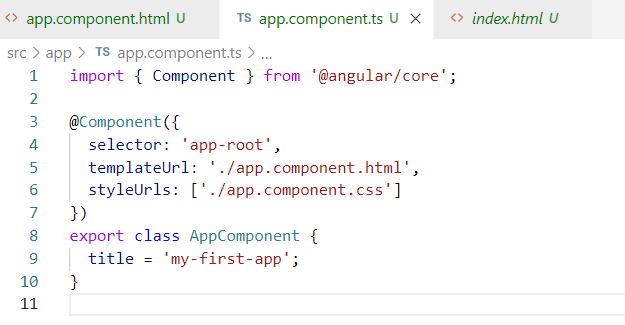
app.component.html: It has root component content

app.component.ts: It has @Component({}) decorator which specifies the tag name for the root component and content for the tag

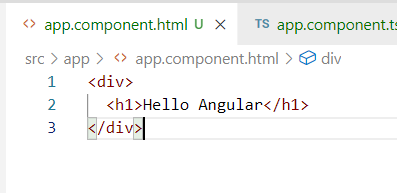
i.e., @Component({selector:’app-root’, templateUrl:’./app.component.html’})

@Component: It is a decorator which must written on top of the class when you want the class to behave like a UI component

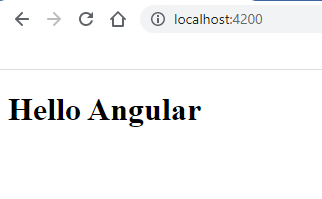
Currently src/app/app.component.ts has below code not changed



Modify the src/app/app.component.html to have a simple content



Angular can detect the changes and reload browser automatically



Note: The only file loaded in the browser is index.html & it must never be modified

Activity:

1. Try the above example as it is
2. Create a new project with the name my-second-app inside day28 folder and remove everything of src/app.component.html and add {{title}}.

Angular building blocks

1. Components
2. Modules
3. Databinding
4. Pipes
5. Services
6. Routers

Components: These are visible part in the page

Modules: This represents functionality of an entire application or part of the application, an angular application is a collection of all the building blocks which are part of an angular module called root module

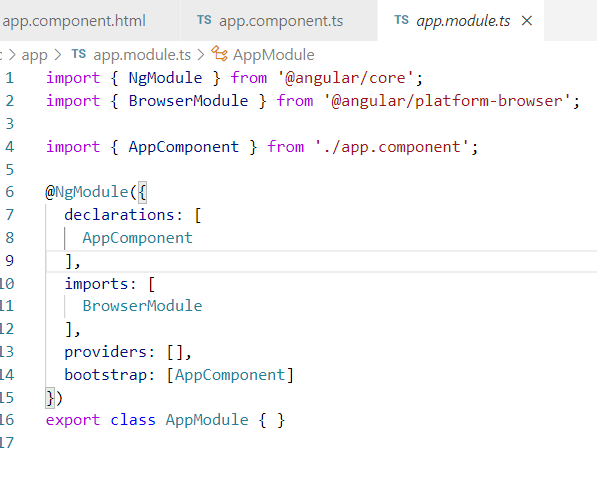
Root module will have the entries of all the components, pipes, routers, services an so on. It represents an entire application.

Your angular project also have a root module in src/app/app.module.ts

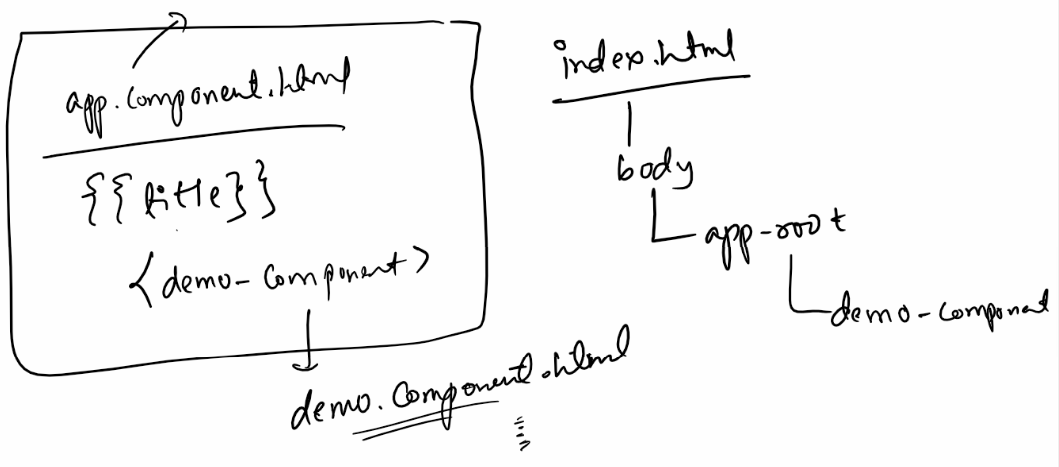
This module is launched when you enter ng serve, this takes care of loading all the elements of the application.

@NgModule({}): It is decorator used to create the module

By default app.module.ts will have below entries



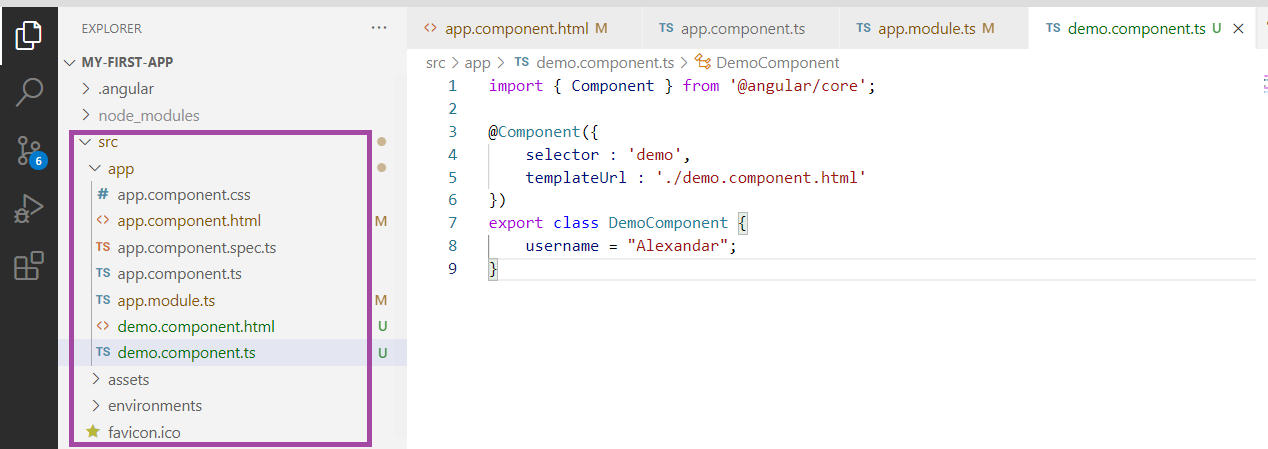
Here the declarations property mentions all the components you create in your application, any components you create is recognized by angular only if you declare in AppModule.



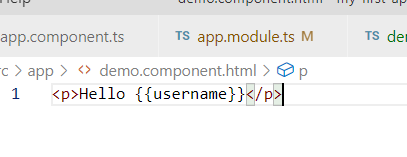
Creating our own components

1. Create a ts file
2. Create an html file
3. Mention the component in the app.module
4. Add your component(<demo>) in another component(app.componen.html)

app/demo.component.ts



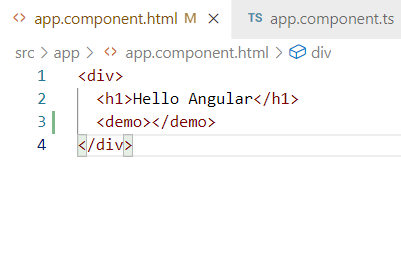
app/demo.component.html



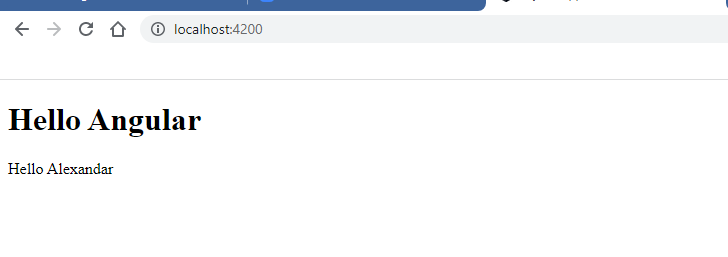
app/app.module.ts



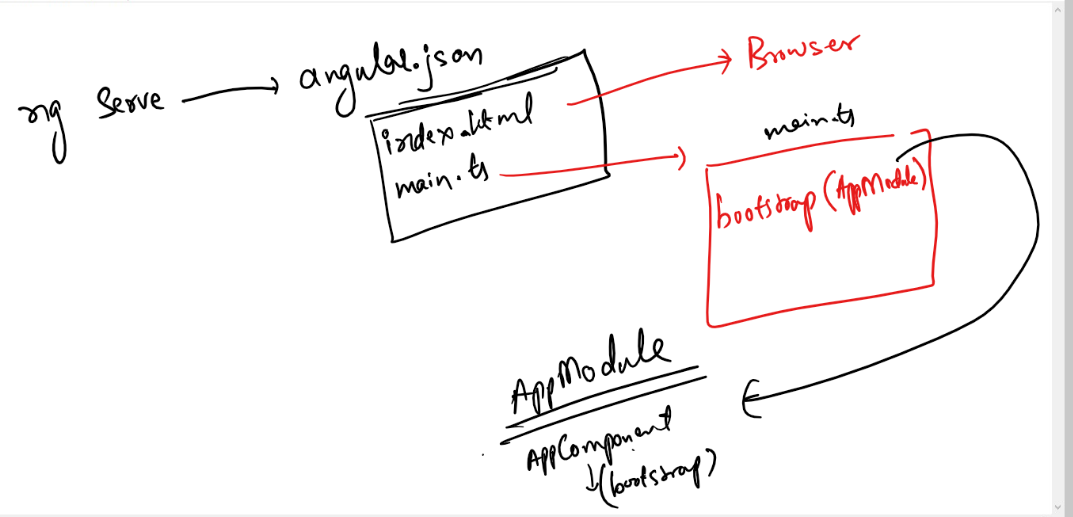
app/app.component.html



Output:



What happens when you use ng serve



The ng serve is a script that looks for angular.json file which is a configuration file used by angular there it finds index.html and main.ts file as an entry point file, the index.html will be loaded in the browser & main.ts will be executed, the main.ts will have a code to bootstrap AppModule, since the AppModule represents everything of your application the main.ts is the one who loads your entire application by loading AppModule, the AppModule will have @NgModule({}) decorator that declares all the components, other building blocks of angular but bootstraps the root components as it is the very first component appears in the browser DOM tree.

Lets create another component in our angular application

You can create a component named user.component.ts which will have name & age properties in its class.

A class must have @Component({}) decorator to behave like a component, the @Component({}) must specify the selector & templateUrl,

selector: tag name of the component

templateUrl: content for the selector

You can use the selector in any other components either in root component or some other components also

@Component: It is a decorator used in angular to make class to behave like a UI component

Decorator: It is a name prefixed with @, it adds extra feature to the class, variables that helps them to have extra capability in the application, there lot of inbuilt decorators in angular which are used on top of the class / variables

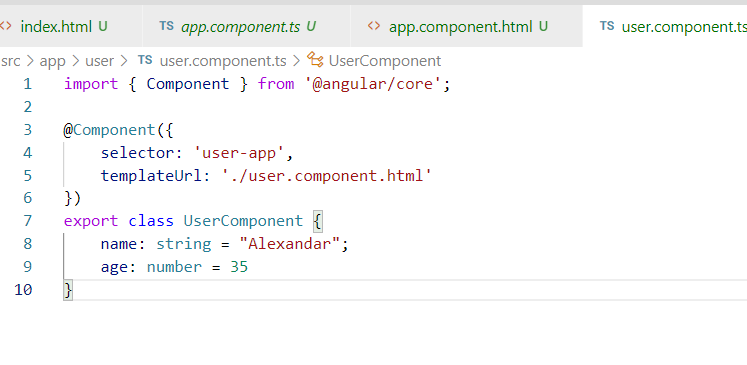
If a class App{} is created it is just a normal class

If a class App{} is created by having @Component({}) on top of it, then it becomes component class, so that the class can share the data to the user and get the data from the user

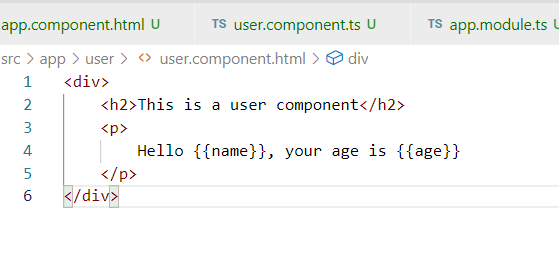
Creating a user component inside app/user folder

Here the ‘user’ folder would have all the files related to user components like user.component.ts, user.component.html, user.component.css

src/app/user/user.component.ts

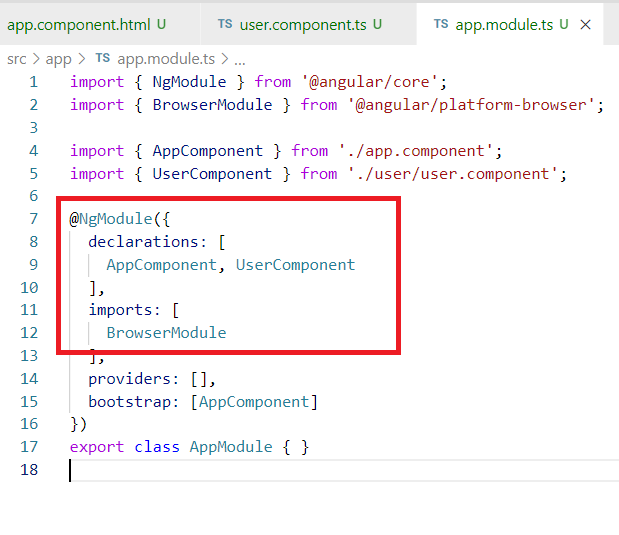


src/app/user/user.component.html



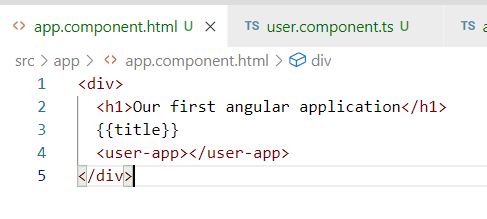
Register the UserComponent in the AppModule

src/app/app.module.ts

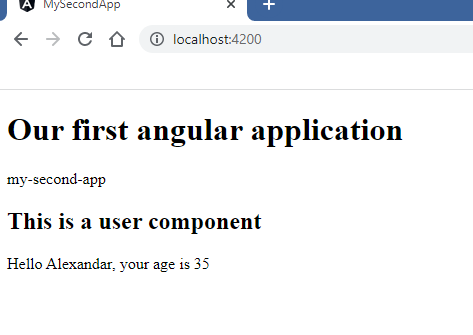


Add the user component in any other components which is part of the root component, since we have only two component here i.e., UserComponent and AppComponent we are adding UserComponent in AppComponent, but it must be added by tag names i.e., selector i.e., <user-app>

src/app/app.component.html



Output:



Templates for the components

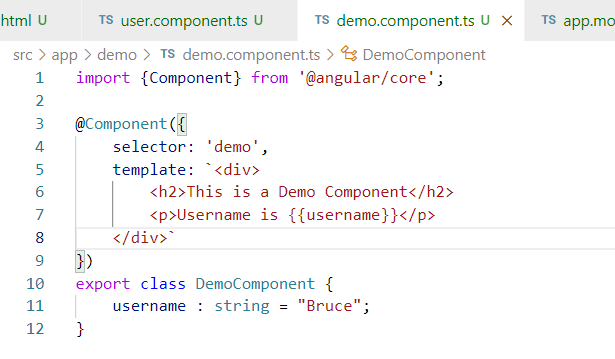
These are the contents written for the component selector (tags), it can be written externally using templateUrl or internally using template which is treated as inline template

@Component({  
 selector:’tag-name’,  
 template: ‘content’  
})

The above component uses inline template, this you can use when the component has only few content, but still you can use external template to get the advantages of the auto-suggestions.

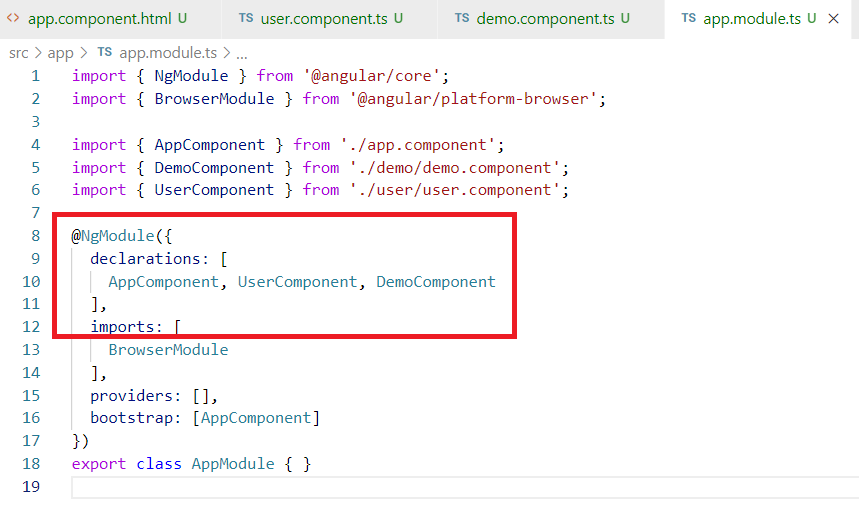
Inline template example

app/demo/demo.component.ts



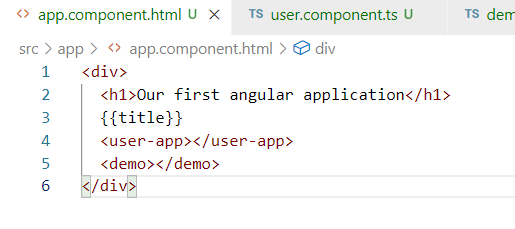
template is used to create an inline content for the component, but it is suitable only when you have a very few lines of contents, if the component contents are more, then you can use templateUrl

Register the DemoComponent in the AppModule app/app.modoule.ts



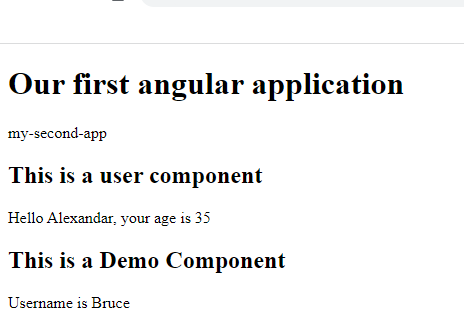
Add the DemoComponent in root component

app/app.component.html



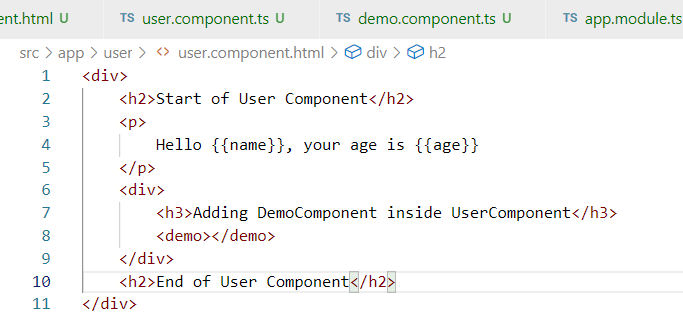
Note: You can add your components in other components also apart from the root component, i.e., you can add demo component int the user component if required

Output:



Let us add <demo> component in the UserComponent i.e., user.component.html

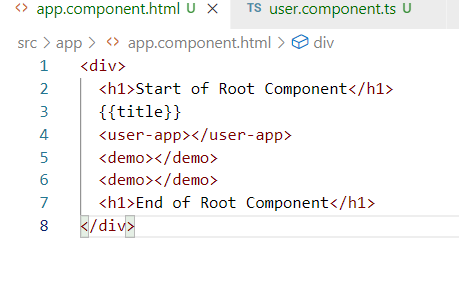
user.component.html



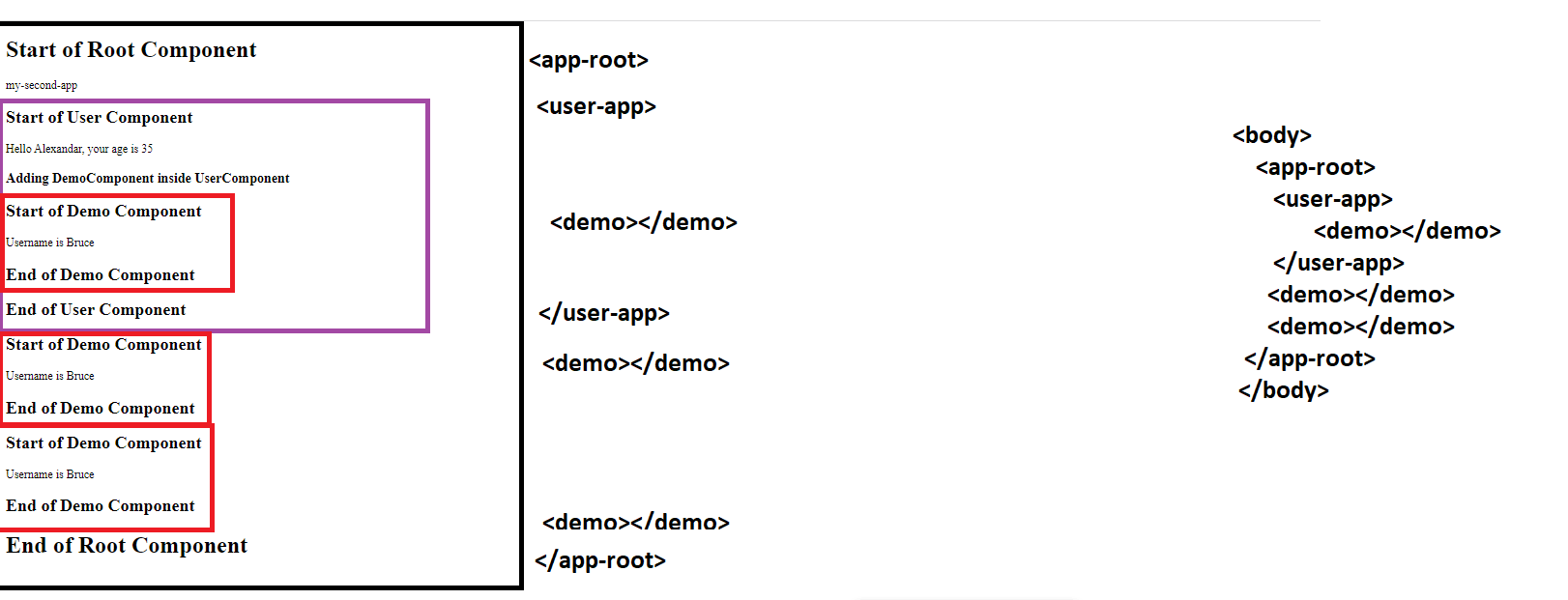
demo.component.ts



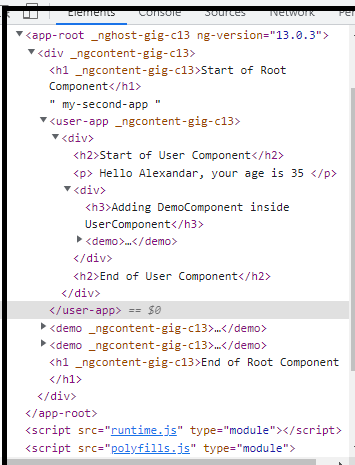
app.component.html



Output:



Browser DOM tree structure

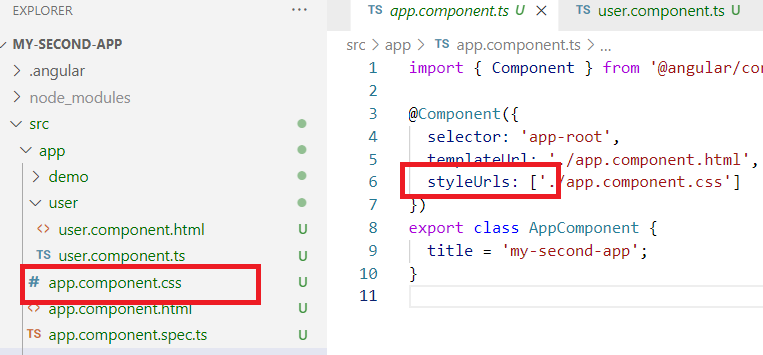


How to style these components

For every components there is a local CSS which applies styles only to the component it is bound to, you can also use global CSS which applies styles to all the components.

Local CSS:

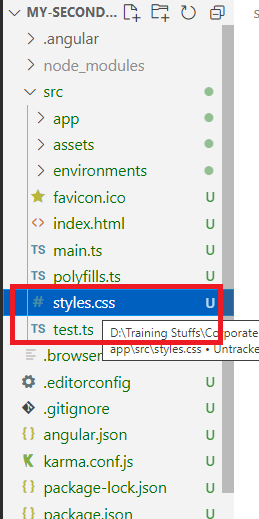
The @Component({styleUrl: [path]}) would mention the Local CSS, where you can apply styles to a particular component



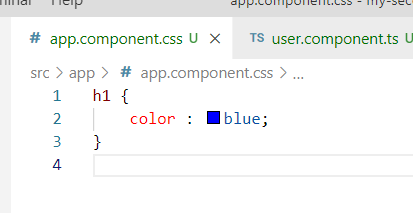
Note: You can create CSS for each component and reference it by using stylesUrls

Global CSS

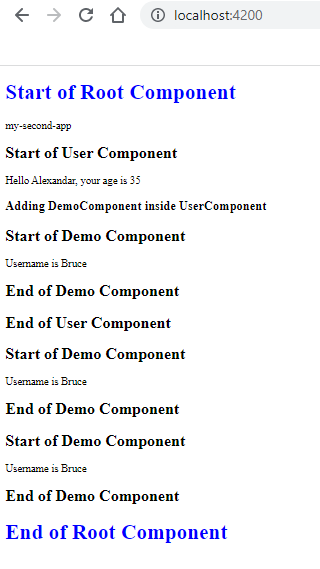
You have a styles.css file in the angular project, that applies CSS to all the components



Let us add style to the app/app.component.css

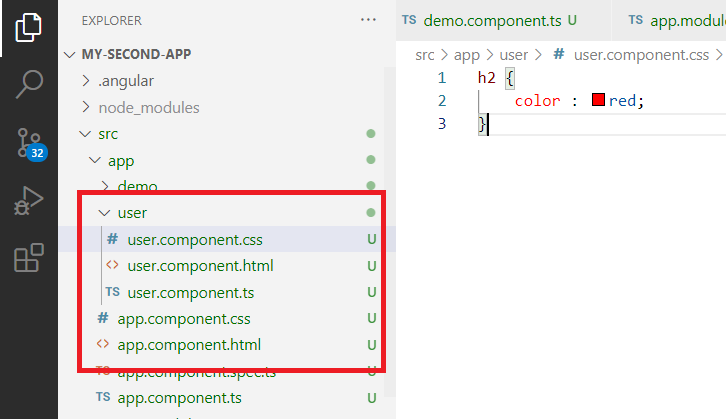


You can see all the <h1> in app.component.html taking blue color



Note: We can create local CSS for the other components

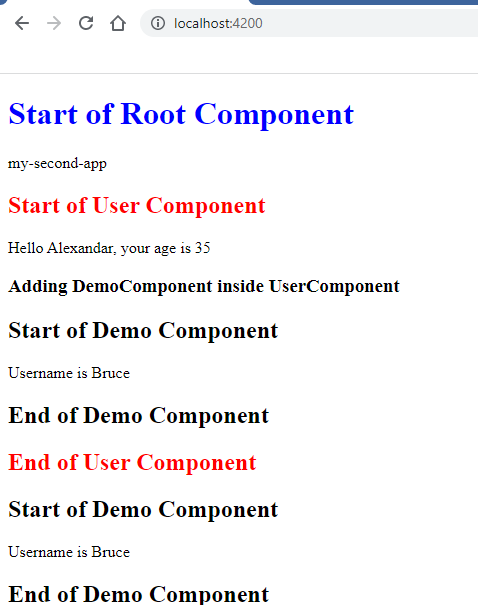
src/app/user/user.component.css



Modify src/app/user/user.component.ts



Output:



Summary:

* ng serve: launches the application by executing main.ts and loading index.html
* AppModule: Root module that represents entire application, this will have all the elements the application should recognize
* AppComponent: Root component which is parent to all the other components
* After creating the component you need to register in the app.module.ts i.e., in the declarations of @NgModule({})
* Inline Template: It is used when the component itself wants to have the content
* styleUrls: It is for adding styles to the particular component, it is a Local CSS
* styles.css: It is for adding styles globally.

Angular CLI commands

Angular provides many inbuilt commands to improve the development speed, these commands are entered with ‘ng’ command

ng generate component component-name (or) ng g c component-name: This is the command which creates .ts, .css, .html files for the components & it also registers the component in the app.module.ts

ex: ng g c demo: This creates 4 files and upates the app.module.ts, those 4 files are

1. demo.component.ts
2. demo.component.css
3. demo.component.html
4. demo.component.spec.ts

Apart from creating these files it will also add some code in some files that developers repeatedly write.

ie..,

@Component({  
 …

})   
export class DemoComponent { }

Apart from ng g c we have many useful commands like

ng generate service service-name (or) ng g s service-name

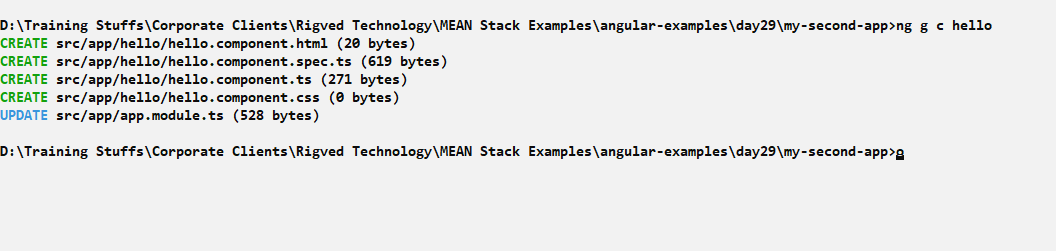
ng generate pipe pipe-name (or) ng g p pipe-name

ng generate guard guard-name (or) ng g g guard-name

All the above commands we are going to see in future

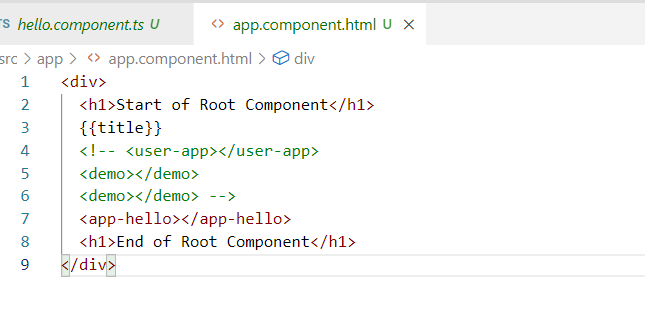
Using ng g c command

To enter this command you must have a separate command prompt that is inside the project folder.

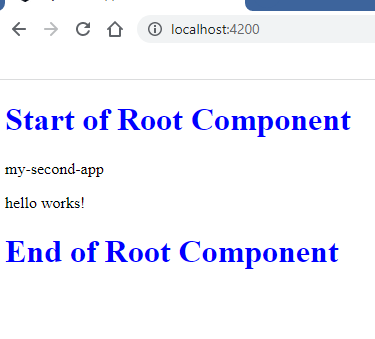


The component is ready to use in other components, you can add in the app.component.html to see the default content i.e., hello works

src/app/app.component.html



Output:



Day 30:

ng serve --port 4201

The above command runs the application in 4201 port instead of 4200

Command to generate component

ng g c component-name

The above command creates templateUrl to represent the view outside the component class

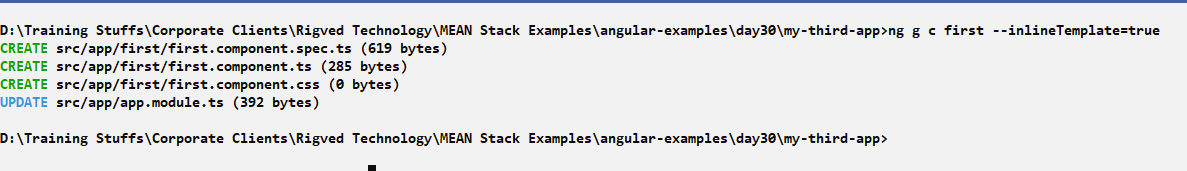
@Component({  
 selector : ‘tag-name’,  
 templateUrl: ‘url’  
})

Command to generate component with inline template

ng g c component-name --inlineTemplate=true

The above command generates the component and view will be part of the component class as below

@Component({  
 selector : ‘tag-name’,   
 template: ‘<p>component wroks</p>’  
 styleUrls: []  
})



first.component.ts

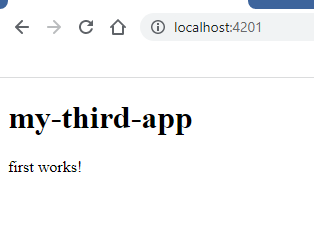


To see the first component content you must add its selector/tag-name in the root component

app.component.html



Output:



Angular Pipes(|)

These are the features in the angular which can format the output so that the content will be transformed to something else instead showing the original content, angular provides lot of inbuilt pipes that can use in the component template using pipe symbol(|), the inbuilt pipes are

uppercase, lowercase, currency, date, json

{{data | uppercase}}: The data will be shown in uppercase

{{data|lowercase}}: The data will be shown in lowercase

{{data|currency}}: This data will show the number in currency format, by default it will be $, but you can pass arguments to the pipes if you need some other currency symbol

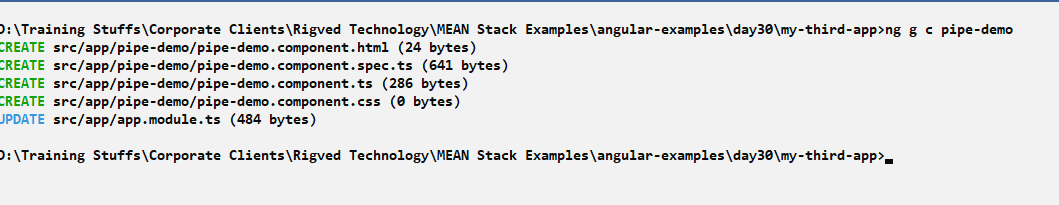
{{data | currency : ‘INR’}}: Here the currency is the pipe name & INR is the argument that specifies the condition what the currency should be in, it will show rupee symbol

{{data | date}}: Shows data in default date format, but it can be formatted according to the requirement

{{data | date: ‘MM-dd-yyyy’}}: Date will be in MM-dd-yyyy format

{{data | json}}: Converts the javascript object to json and shows that json to the user

Note: The data must be of right type when we are formatting, else it doesn’t work, ex: If currency is applied then it must be applied on number type data only.

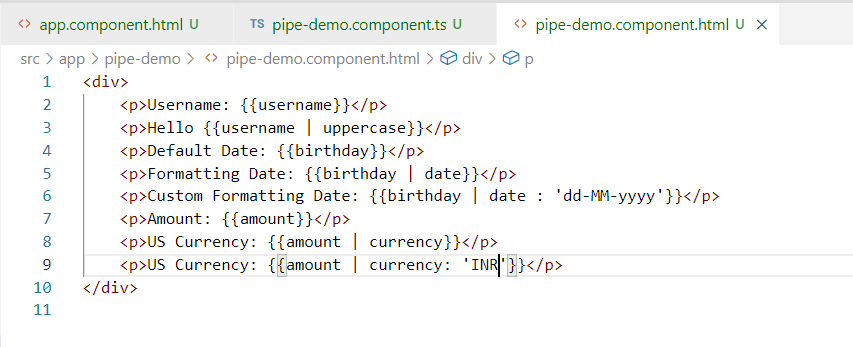


pipe-demo.component.ts

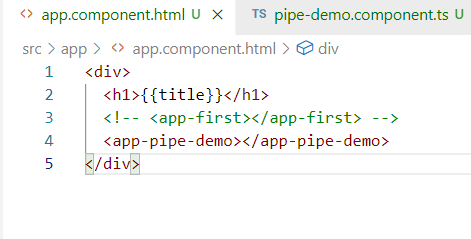


We have username, birthday, amount in the component class that can be formatted using the inbuilt pipes

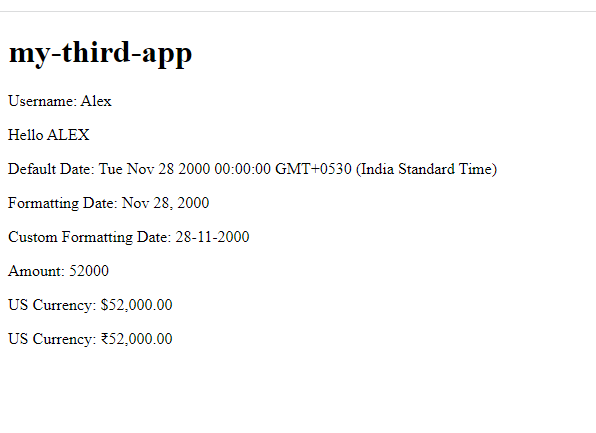
pipe-demo.component.html



Note: Add this component to the root component



Output:



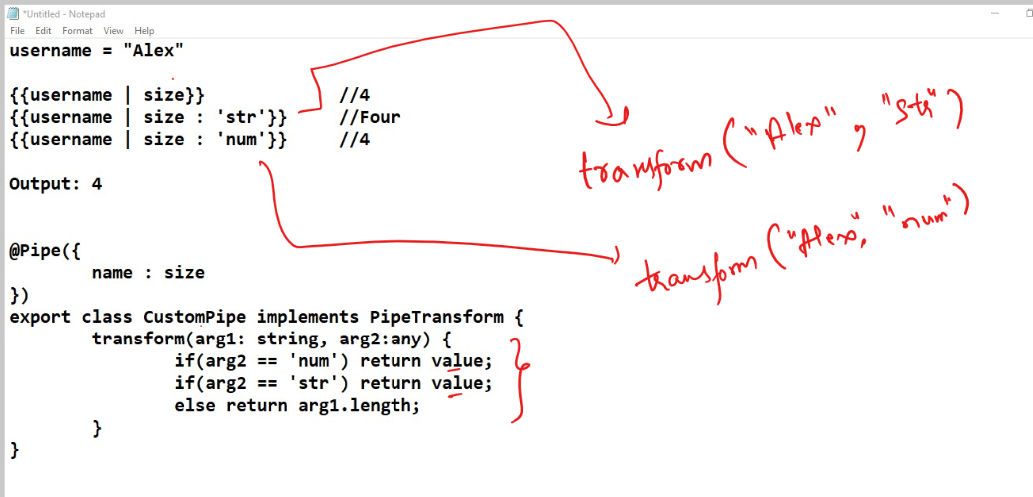
Custom pipes:

These are created when you need your own pipes for transforming the data when the predefined pipes doesn’t help your requirement

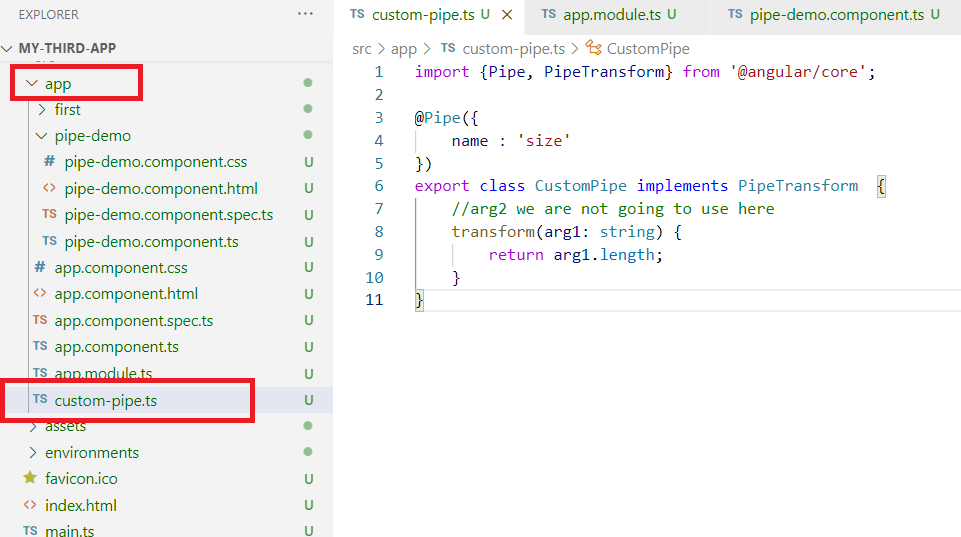
PipeTransform: It is an interface a class must implement while creating the custom pipe, the PipeTransform has a function called transform() that is executed when the custom pipe name is used, the data transform returns would be the transformed data that comes after applying the pipe

@Pipe: It is used to decorate a class to act like a pipe, it will have a name which is used while transforming the output,

Note: A class that implements PipeTransform must have @Pipe decorator, it must be registered in the AppModule so that angular calls the transform() function at runtime automatically by passing arguments to the transform() whenever the pipe is used

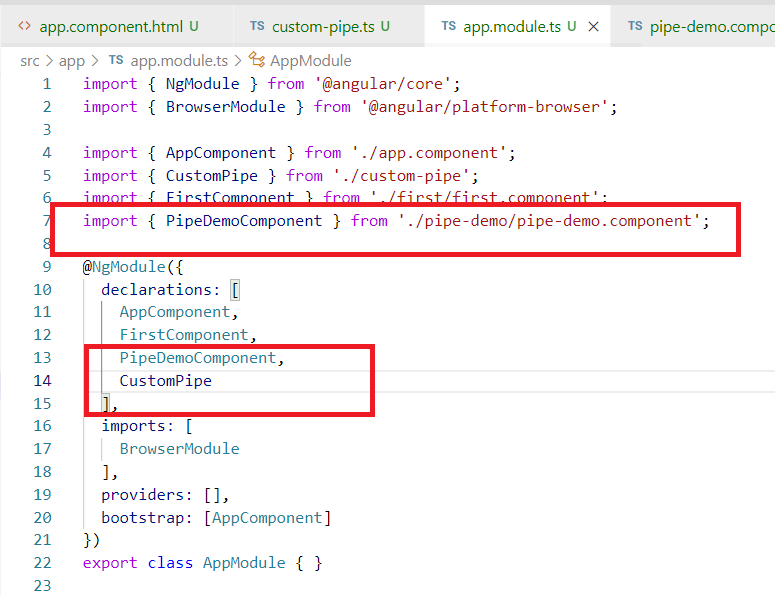


src/app/custom-pipe.ts



The ‘size’ is the pipe name which needs to be used when you want to transform, but angular needs to know this pipe name hence you must register in app.module.ts

app.module.ts

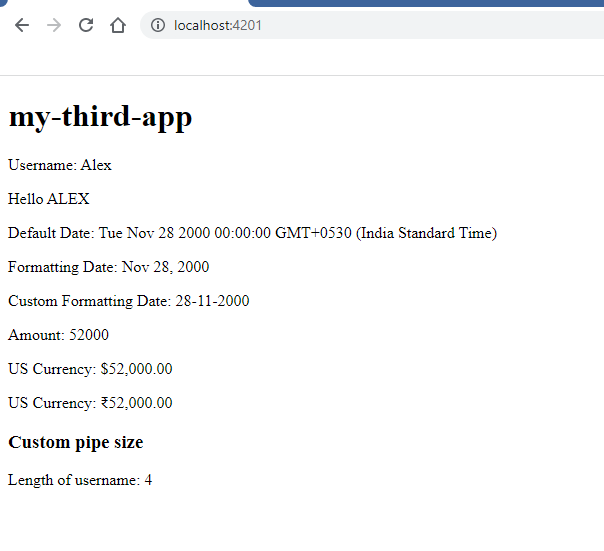


Now you can use the pipe name size anywhere you want and see the size of the string

pipe-demo.component.html



Output:



Structural Directives

These are like attributes that you can use in the view template element, it works with the DOM Structure whether to show or hide the element or to add the elements at runtime

We have 2 structural directives:

1. \*ngIf
2. \*ngFor

\*ngIf: It is used when the element needs to be shown or hidden

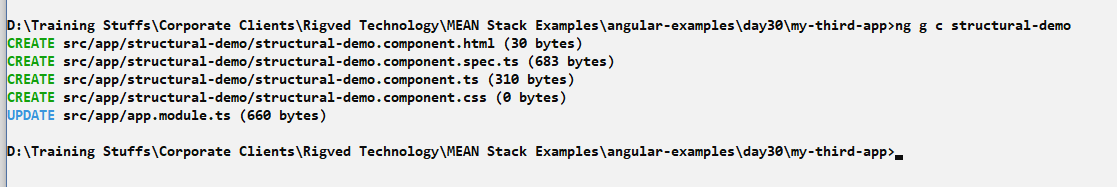
ex: <p \*ngIf = “property”>Some content</p>

Here the content will be shown when the component property value exists/true else it will be hidden when the property is false/undefined

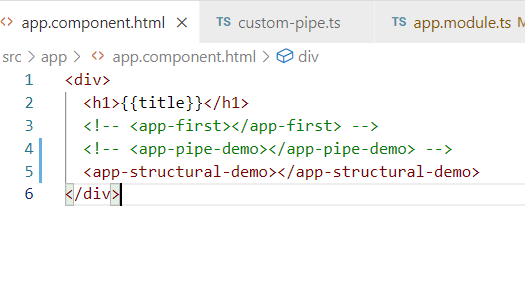
\*ngFor: It is used to iterate the array present in the component class, it allows you to dynamically add the elements in the DOM

ex: <p \*ngFor = “let i of items”>{{i}}</p>

Here ‘i’ is an iterator to iterate the elements present in the ‘items’ array



Add the structural-demo in the root component

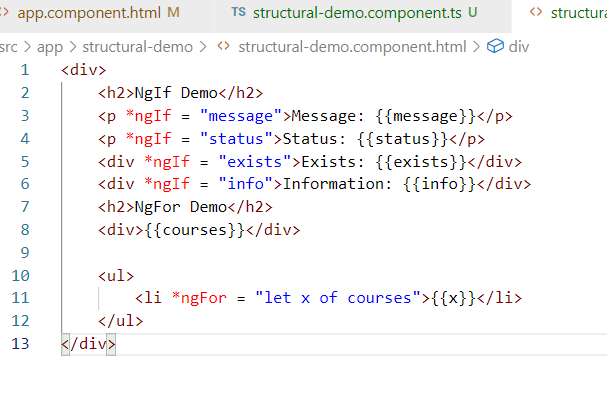


structural-demo.component.ts



You can use \*ngIf for all the properties and \*ngFor on courses as its an array

structural-demo.component.html



Here x iterates each item present in courses i.e., Javascript, Python, Java, C# and <li> is created as many times the iteration happens

Output:



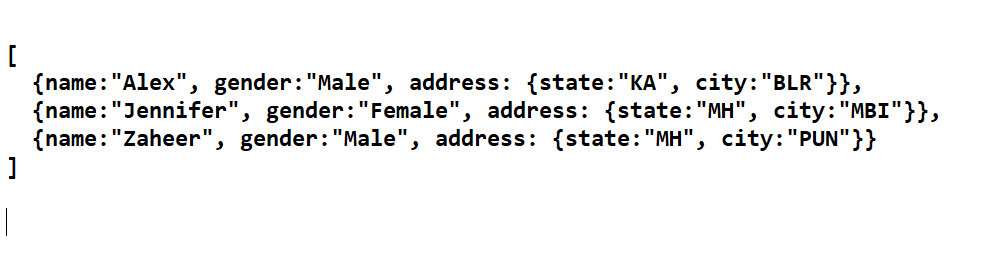
Activity:

1. Try Structural directive activities
2. Display the employee data present in the Javascript object given

employee = {id:100, name:”Ajay”, salary:45200}

Expected Output: Hello Ajay, your id is 100 and salary is 45200

1. Using Structural directive and Custom pipe iterate over an array of users and print users data in a table format, below is the user array



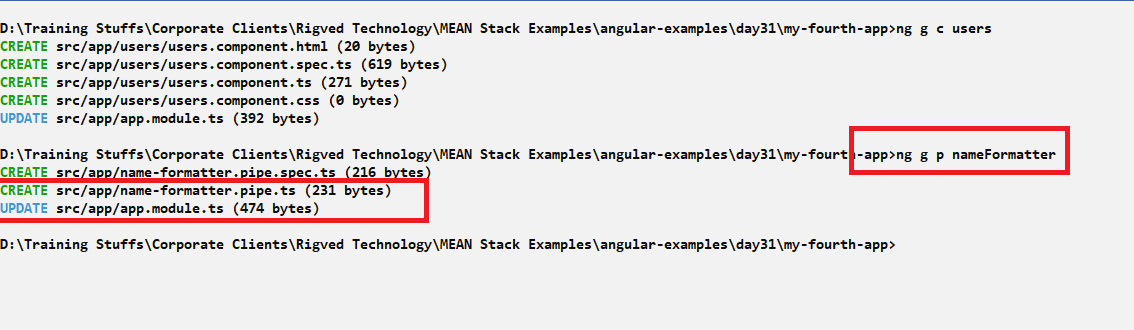
Expected Output:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Gender | State | City |
| Mr. Alex | Male | KA | BLR |
| Ms. Jennifer | Female | MH | MBI |
| Mr.Zaheer | Male | MH | PUN |
|  |  |  |  |

Note: You must use custom pipe that can format the name with the help gender

You can generate the pipe using ng generate pipe <<pipe-name>> or ng g p <<pipe-name>, this will automatically register the pipe in the AppModule and also creates a class with @Pipe({}) and implements PipeTransform

Solution:



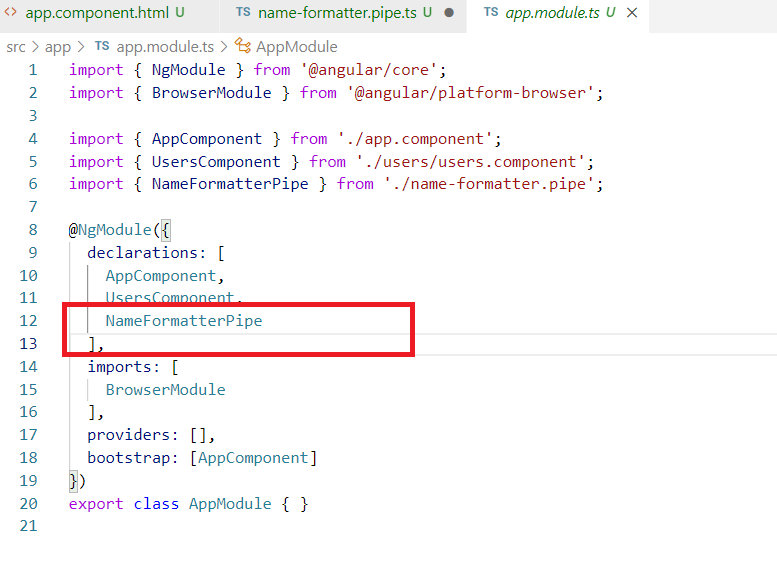
We created users component and added that in the root component, then we created a pipe with the name nameFormatter, the command does 50% of work like creating a class implementing PipeTransform, providing transform() function & registering the pipe in the app.module.ts

name-formatter.pipe.ts looks as below



Note: You have to change the above pipe according to your requirement

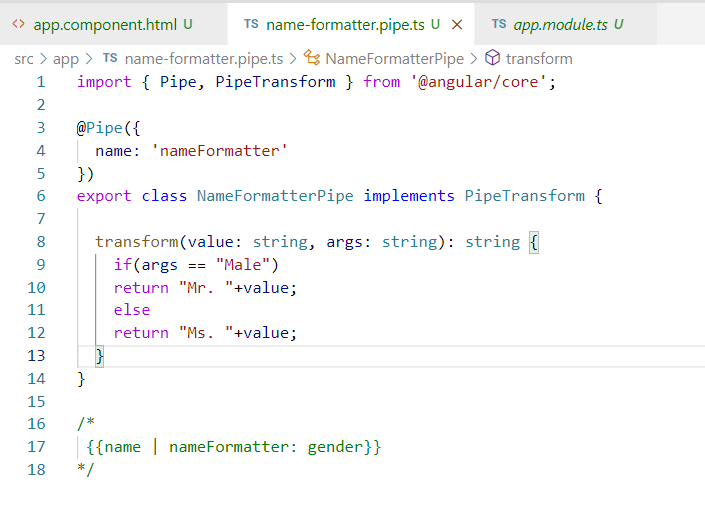
You can see that pipe has been registered in app.module.ts



Changing the nameFormatter to take name and transform it based on the gender, i.e., we are going to using {{name | nameFormatter : gender}}, this nameFormatter is a pipe whose transform() is called by taking name in its 1st argument & gender in 2nd argument.

Here if name is Alex & gender is Male, then it must return Mr. Alex, similarly if name is Jennifer & gender is Female, then it must return Ms. Jennifer

name-formatter.pipe.ts

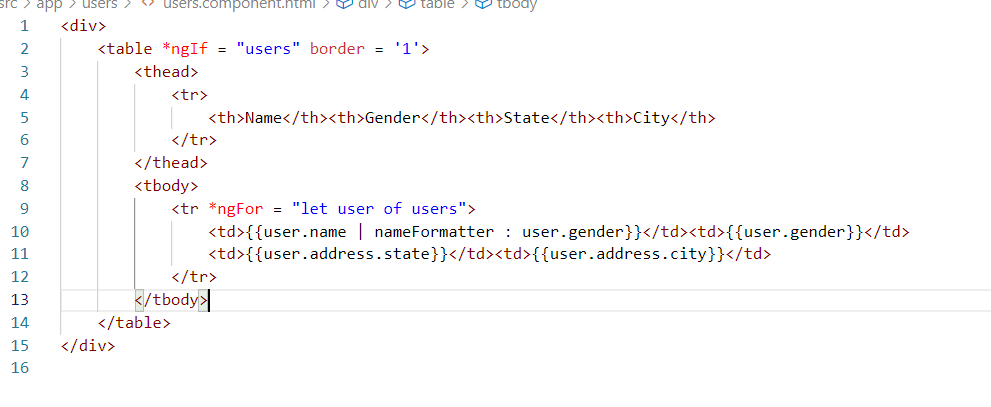


Now you need to use the nameFormatter pipe in users.component as it will have users array.

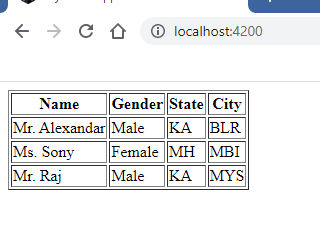
users.component.ts



users.component.html



Output:



Adding bootstrap to our project

Bootstrap provides a good look and feel with the help of its predefined classes, you need to install bootstrap in your project and import it in styles.css so that you can globally use their classes in any component

Steps to add bootstrap in angular project

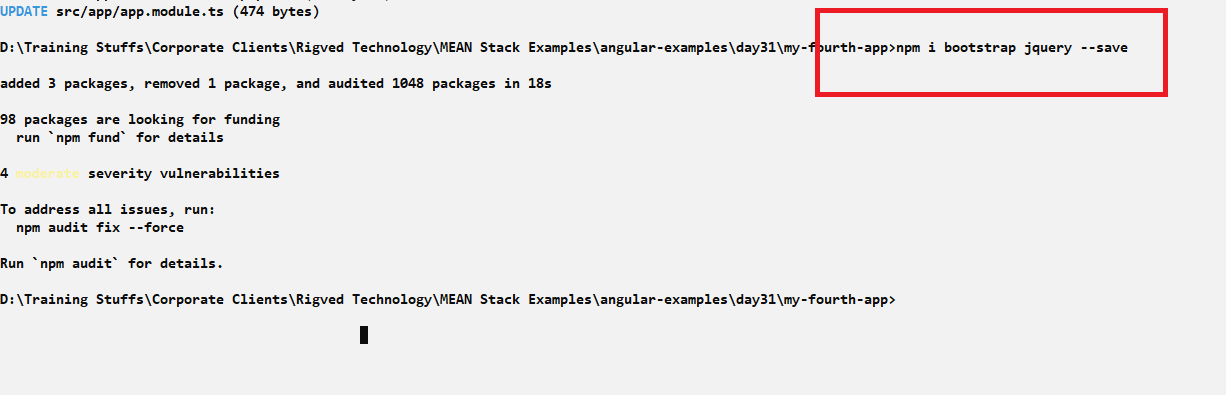
1. Download bootstrap js using npm, since jquery is another library that bootstrap depends on we need to install jquery also

i.e., npm install bootstrap jquery --save

Since some of the bootstrap functions call jquery functions we had to download jquery

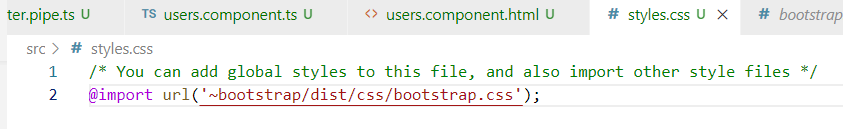
1. Import the bootstrap css file location in the styles.css, the location of bootstrap css will be in node\_modules folder
2. You can use bootstrap classes in your components

Installing bootstrap & jquery



You can locate the bootstrap in node\_modules/bootstrap/dist/css/… folder

Importing that location of css in styles.css



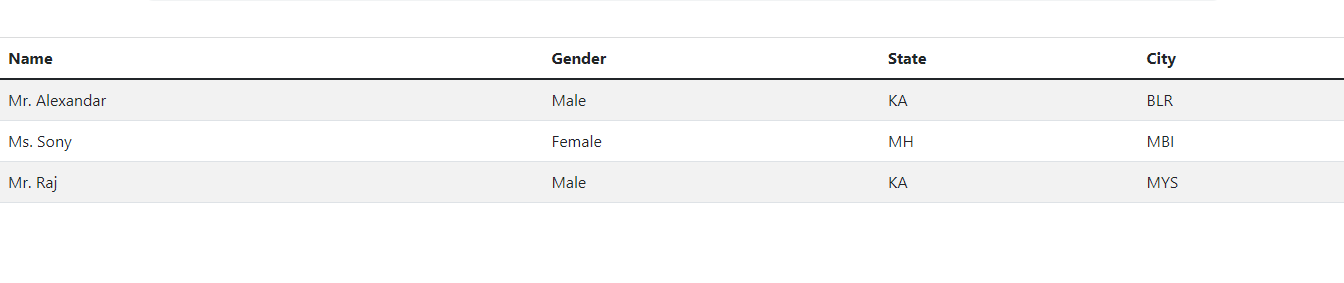
Now you can use all the bootstrap classes in your components like btn-primary, container-fluid, card, table, table-striped, form-group, form-control and etc.

Adding table class our components

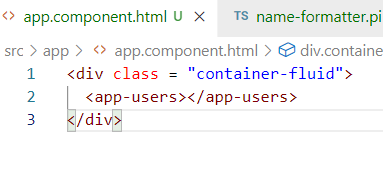
users.component.html



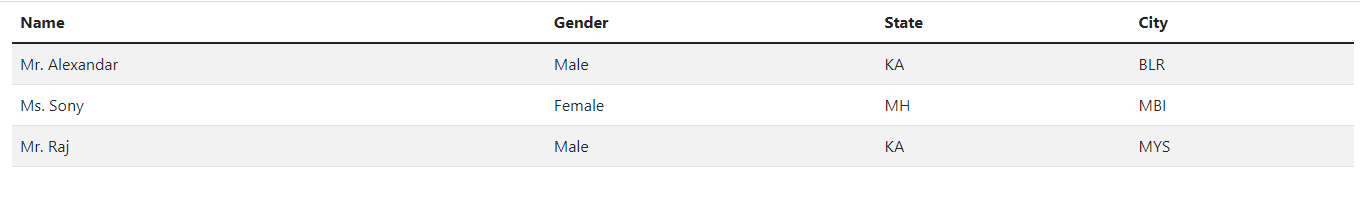
Output



Using container-fluid in the app.component.html



Output:



Apart from bootstrap you can add other third party libraries like

* semantic ui
* angular materials
* prime faces

Databinding in Angular

It is used to share the data between the view and the component and vice versa, here the component class needs data to perform some action and view also needs data to perform some action

There are mainly 4 types of databinding in angular

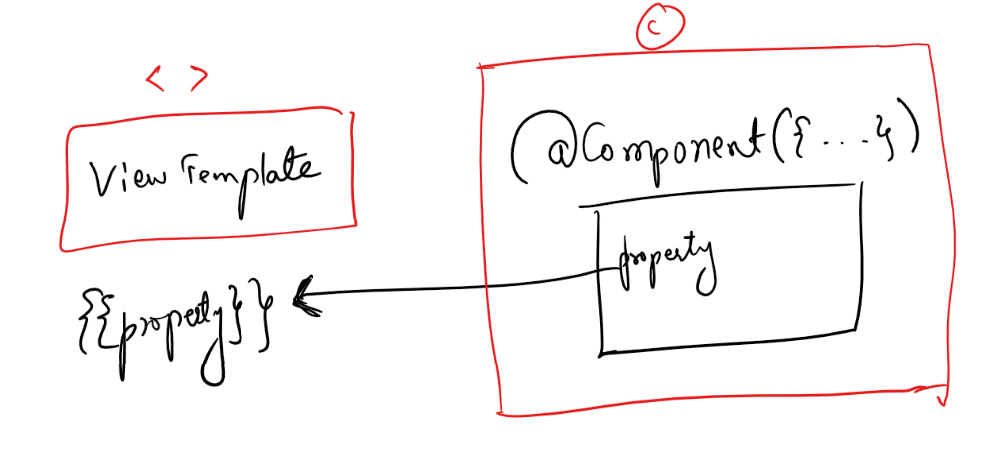
1. Interpolation {{data}}
2. Event Binding (eventName)
3. Property Binding [data]
4. Two way databinding [(data)]

Interpolation: It is used to share the data from component class to view, it is used by {{ }} syntax, it is mainly used to display the data present in the component. Interpolation supports not only accessing the component property it also supports accessing some predefined function of property and also perform some calculations

{{1 + 2}}

{{10 \* 2}}

{{10 / 2}}

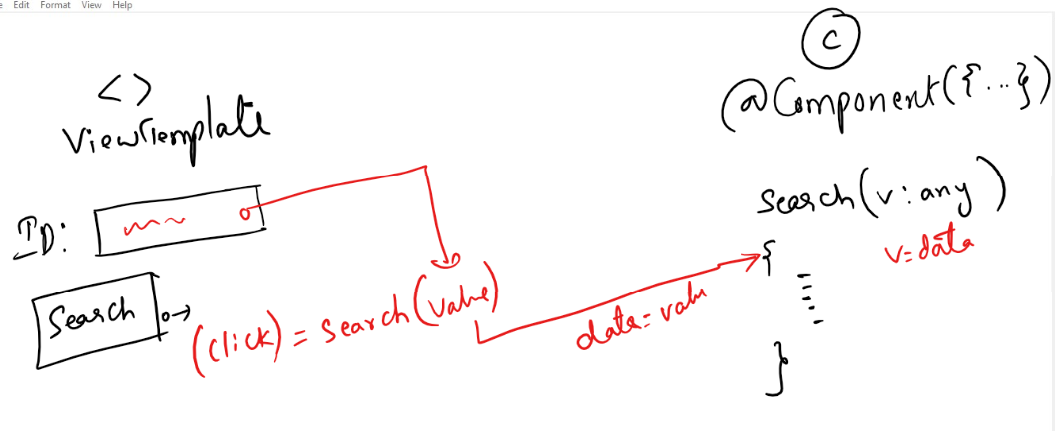


Note: The data flow in Interpolation happens in uni-direction i.e., one way databinding from component class to view template

Event binding: Here the data flow happens from view template to the component class, the data would flow to the component class when any event occurs in the view template, you need to wrap the event name in the parenthesis

(eventName) = action

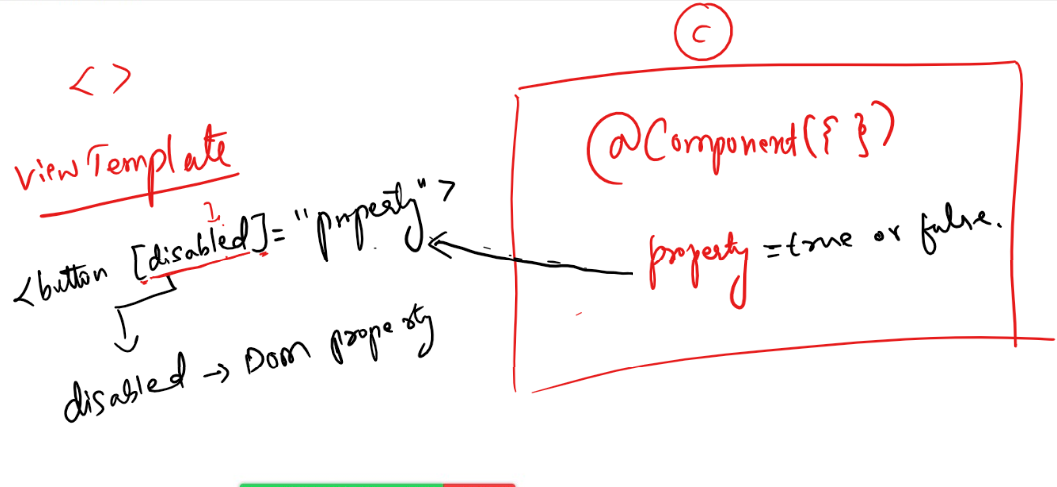
Here the action would be the function call that updates the component data or passes the data to the component class



Note: Event binding is also one way data binding because the data flows in one direction from view to component class

Property Binding: Here the data flows from component class to view template and this is also one way data binding, the use of property binding is it can modify the DOM properties, it is used by adding dom properties in the square bracket

[domProperty] = data



Note: Property binding also a one way data binding, where the data flows from component class to view template, it is mainly used to modify the DOM properties like disabled, innerText, hidden and etc.

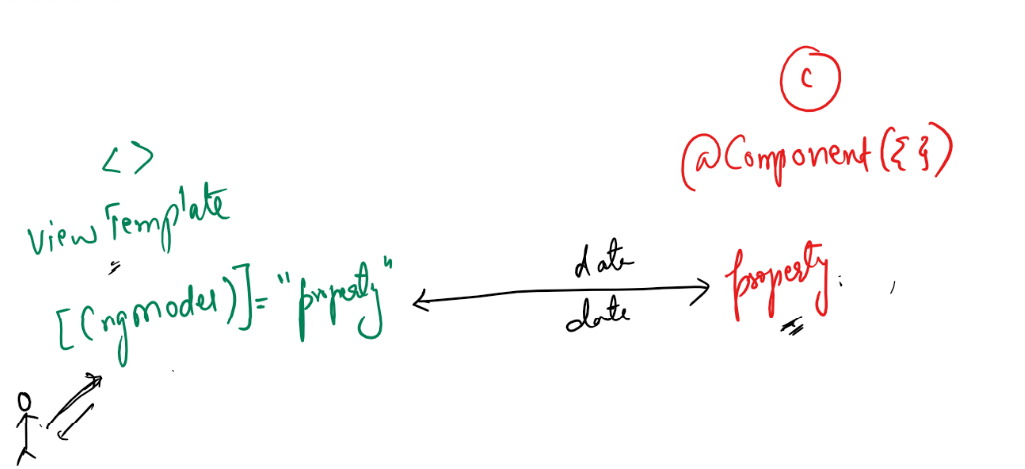
ex: <p [hidden] = “data”>some content</p>

Now if the data is true then hidden = true, so content will be hidden, if data is false then hidden = false, so content will be shown.

Two way data binding: Here the data flows in both the direction from view to component class and vice versa, here the changes you made in view updates the component data and changes happening in component updates the view data, both will be in sync.

[(ngModel]) = “property”

The above expression updates the component data if changes happen in view and also updates the view if changes happen in component class.

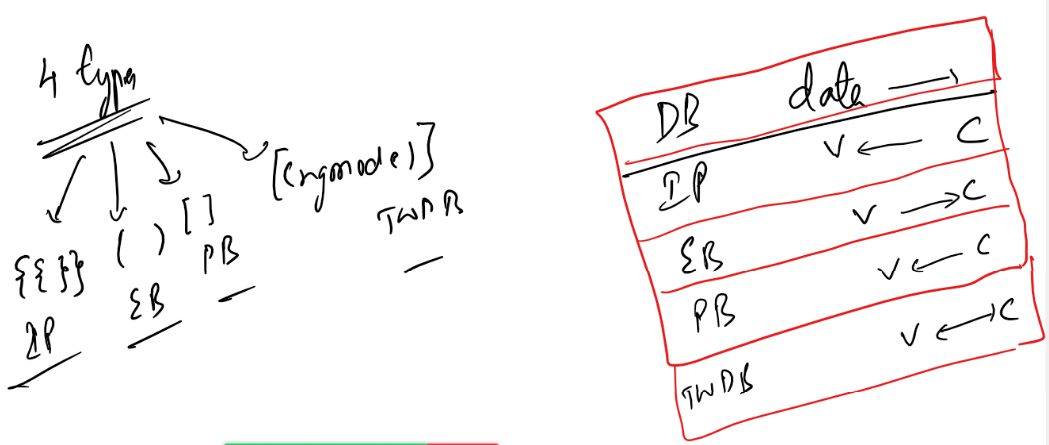


Note: ngModel is a directive which is by default not part of the application module, which needs to be added to the application module i.e., app.module.ts, you need to use FormsModule in the imports of @NgModule to add ngModel to your application, then only you can use [(ngModel)] in your view template

i.e.,

NgModule({  
 imports : […, FormsModule,….]  
})

Now you can use [(ngModel]) in the application.



Event Binding:

Here the data flow happens from view template to component, the view template has to handle the event generated by the DOM elements and update the component property, there are many events that can be handled in the view template, some are

(click): Occurs when you click on DOM element

(input): Occurs when you enter input in the <input> element

(change): Occurs when any changes happen in the checkbox, radio, dropdown

(ngSubmit): Occurs when you submit the form

(focus): Occurs when you focus on a particular input element

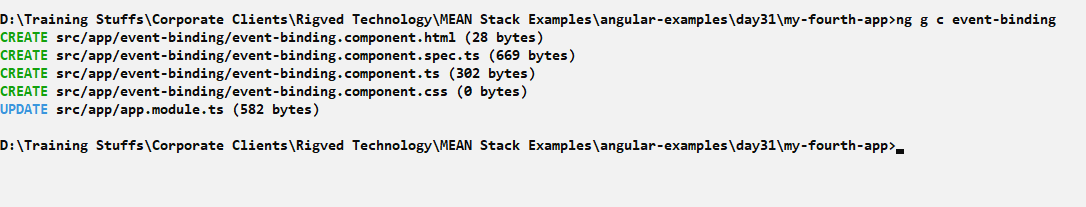
(blur): It is opposite of (focus)

You can call a function present in the component that can modify the component property or pass the data to the component via function call

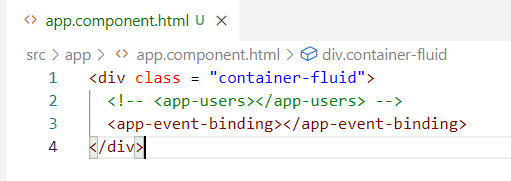
i.e.,

<input type = “button” (click) = “test(data)”>

Generate a component event-binding



Add the event binding component in root component to see its output



event-binding.component.ts



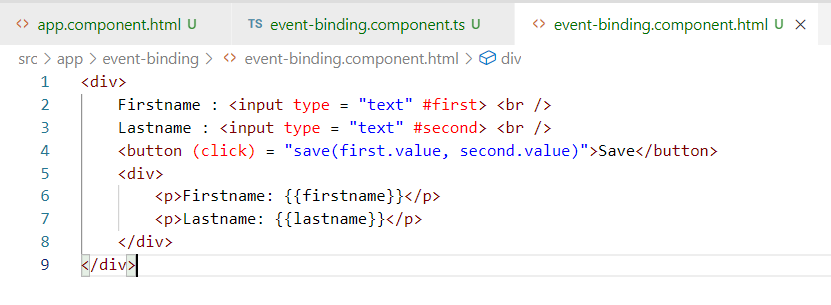
Here the save() function must be called from the view template of event-binding.component that passes value to fn & ln to initialize the firstname & lastname, to ensure there is changes happening in the firstname & lastname we can display it in the view template.

Template Reference Variables:

These are the names given to elements so that they are used by other elements to access element properties, they are used with #name,

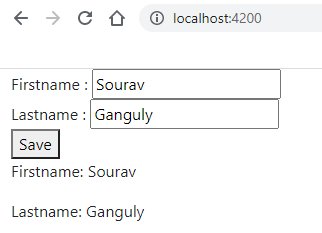
<input #in>, now with the help of ‘in’ you can access the <input> element which has the name #in, here you can read its value by using in.value

event-binding.component.html



Here #first & #second are references to the elements, value is one of the property in the element that returns the value of the element, hence you are using first.value, second.value in the save once you click on the button.

Output:



Here the (click) sends data to the component class and updates the firstname & lastname, however {{firstname}} & {{lastname}} are getting the data from the component class.

Property Binding

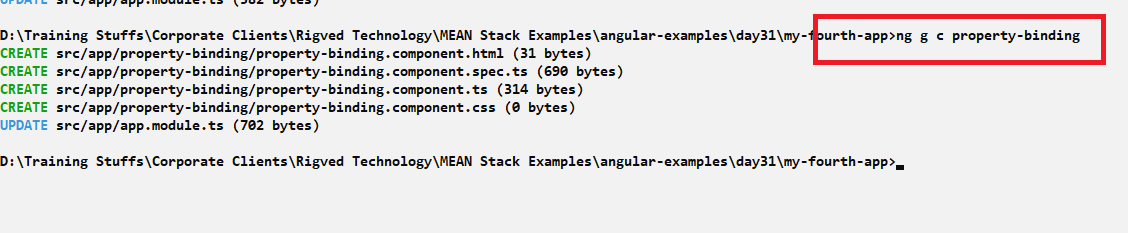
Here the DOM properties can be modified by getting the data from the component class, the view gets the data from component and update the DOM properties.

[domProperty] = “componentProperty”

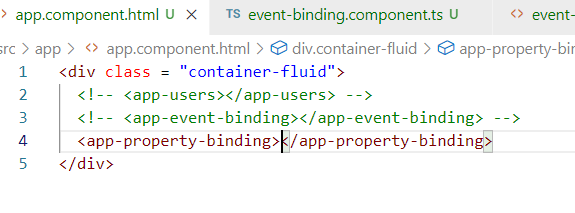
Here domProperty could be hidden, disabled, innerText and so on

componentProperty could be true/false or some content depends on what dom property you are using

Create a component property-binding



Add the component to root component to see the output of property-binding



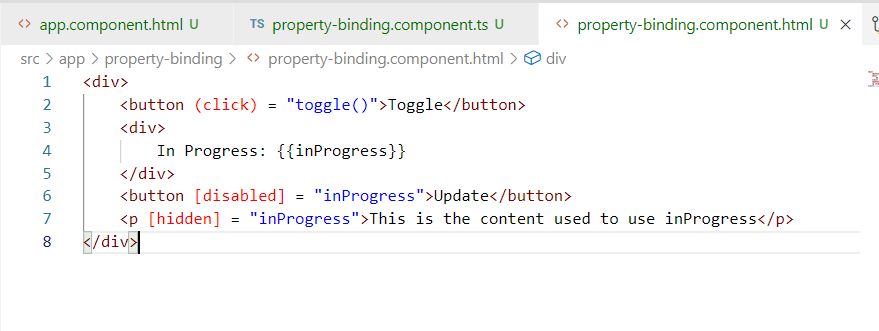
Create a function that changes a property from true to false and vice versa and use that property on any DOM elements to disable or enable the element

property-binding.component.ts



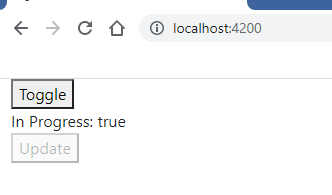
inProgress will toggle from true to false & vice versa, which is used to bind the DOM property however the toggle() is called by event binding method, here the inProgress is shared from component class to view template

property-binding.component.html



Here the data inProgress is shared from component to DOM element present in view template, the button is disabled when the isDisabled = true and also the <p> element is hidden when the hidden = true

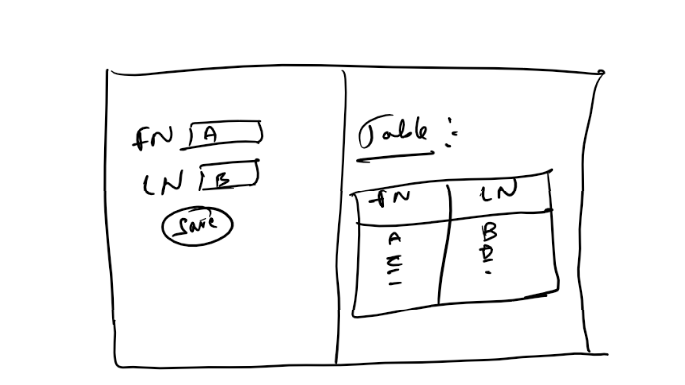
Output:



Activity:

1. Try above 3 types of data bindings & also use bootstrap in your application
2. Create 2 division with the help of bootstrap and in the same component add and display the elements

Output:



Two way data binding:

It is used to share the data in both directions the changes you make in view template is updated in the component class and changes you make in the component class is updated in the view template

[(ngModel)] is the directive used to achieve two way databinding, this directive is not recognized by angular by default, it is part of one inbuilt module FormsModule, which needs to be added in your application i.e., AppModule imports property

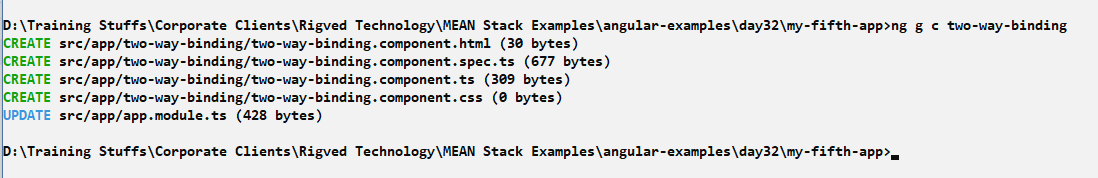
@NgModule({  
 ..

imports: [FromsModule,…]

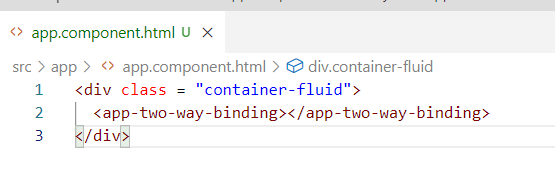
..  
})  
export class AppModule

FormsModule is present in the @angular/forms library, it provides many directives which are useful in case of form handling like ngModel, ngForm

Create a component two-way-binding



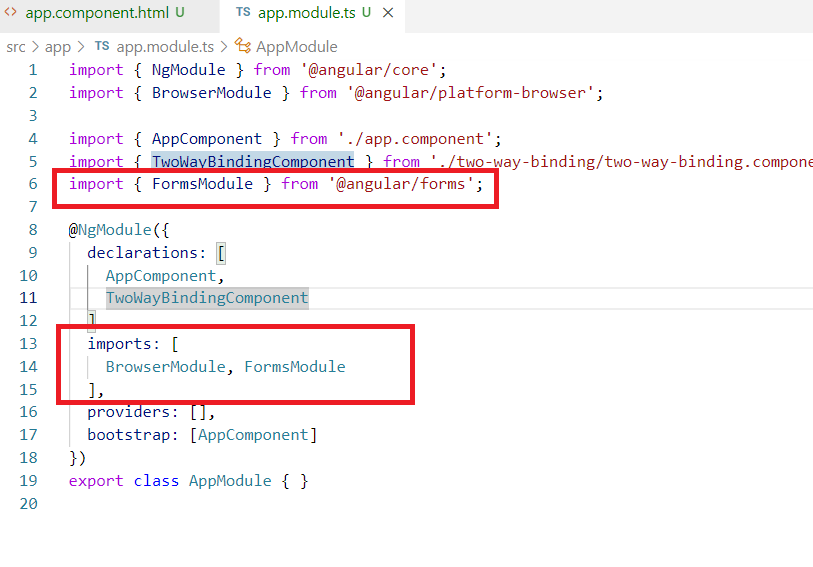
Add this component to the root component



Note: You don’t’ have to add every new component in the root component, because there could be a scenario where you will add the new components to some child component which was part of either root or some other components, but root component will always be your top level component though your new components don’t directly be inside root.

Since you need to use ngModel, you need to add FormsModule in the @NgModule({}) of AppMcoule

app.module.ts



Now we can use the directives of FormsModule like ngForm, ngModel and so on.

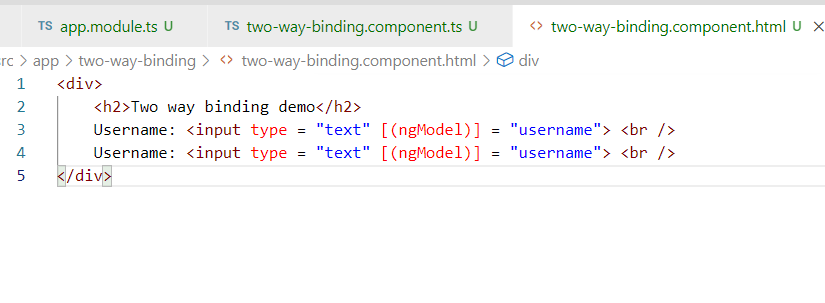
ngModel: Allows you to achieve two way data binding, in both the direction the data will be in sync.

ngForm: Allows you to handle form data in angular

two-way-binding.component.ts

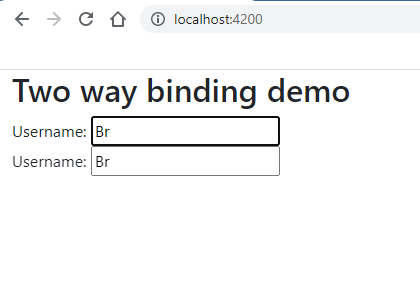


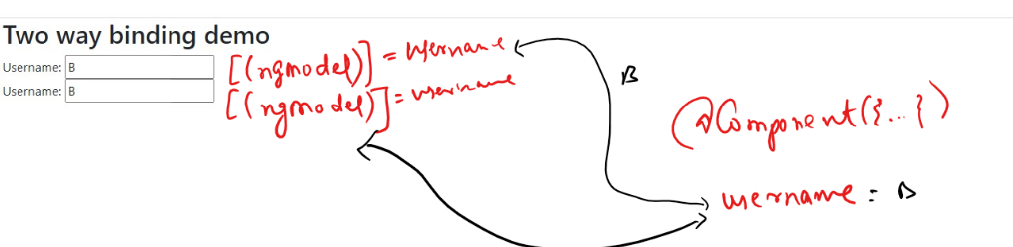
two-way-binding.component.html



Here the [(ngModel)] = username, updates the username present in the component class as well as the view template using username i.e., <input> box itself, the reason 2 input box is created is to see the two way binding direction when one input box is updated another input box also updates, this proves that there’s a change happening in component class from view template

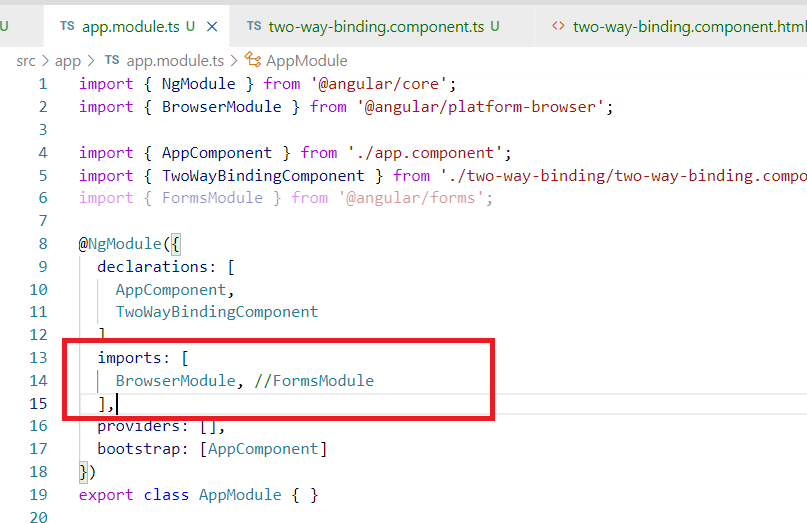
Output:





The code works only if the FormsModule is imported, if its not imported then you will get error in the command prompt & browser saying ngModel is not a known directive.

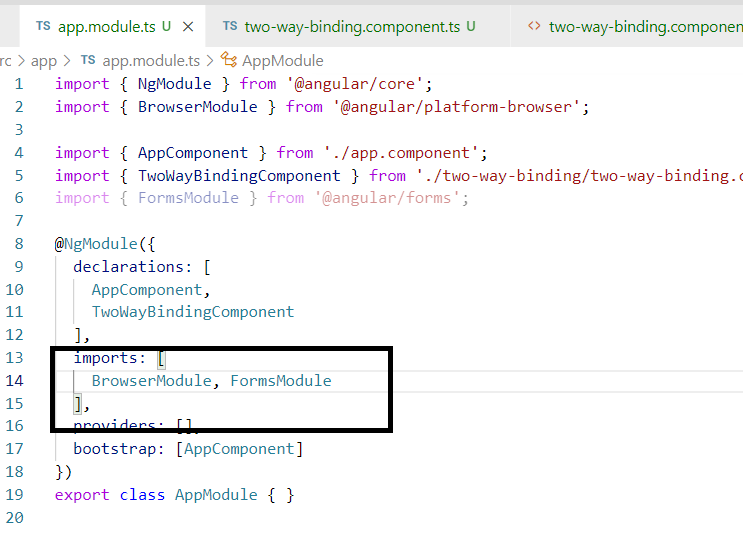
Suppose you removed or commented FormsModule in the app.module.ts -> imports



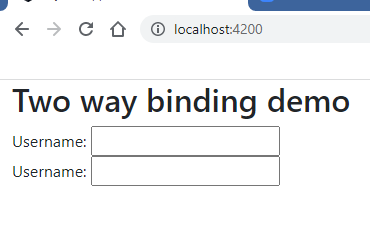
You will see error like below



You can uncomment the FormsModule in the imports of app.module.ts

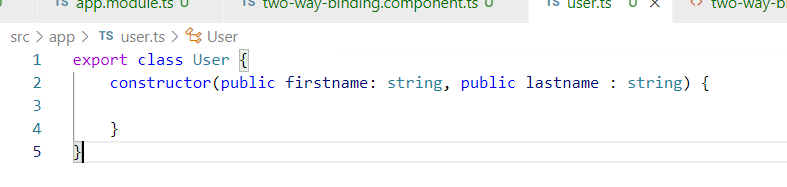


Output:



You can also bind multiple properties or a javascript object to the ngModel

app/user.ts

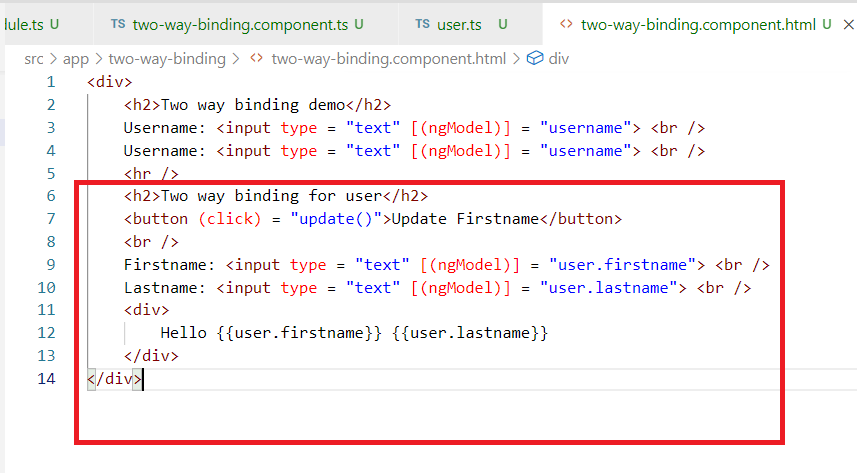


You can create object / variable of User type in two-way-binding.component.ts

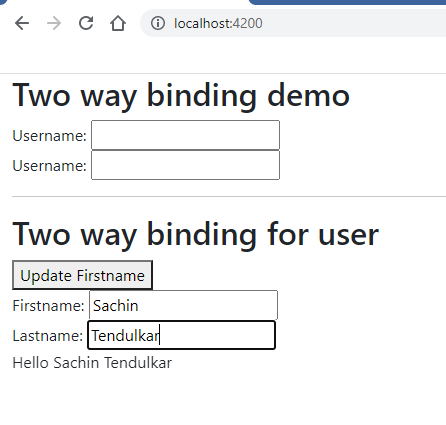


Note: update() is called via event binding that updates the component class property, and that reflects to the view template using [(ngModel)] as it is in sync

two-way-binding.component.html



Output:



Apart from the above 4 types of bindings we also have other types of binding like

1. Attribute binding
2. Class binding
3. Style binding
4. @Input() and @Output() binding

Attribute binding: It is used when you want to bind the HTML attributes in the view template, it is used when there’s no property binding possible, probably because there wouldn’t be DOM property for the attribute

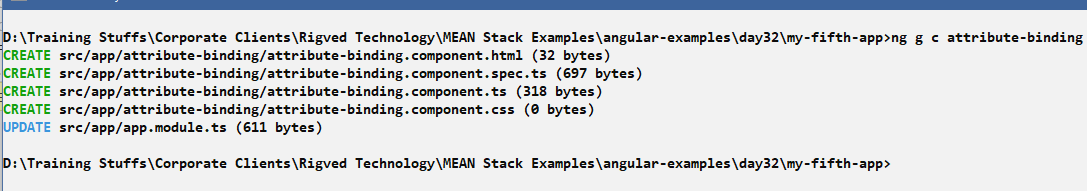
ex: <p [hidden]>: Here the hidden is a DOM property that angular uses to bind, same way you have <button [disabled] where disabled is also a DOM property but we don’t have a DOM property in certain elements like colspan, rowspan of <th>, <td> of <table>, the colspan & rowspan are not having any DOM properties in such cases we can bind to these attributes

<table [colspan] = “value”>: This doesn’t work because colspan is not a DOM property, but we can use <table [attr.colspan] = “value”> this is binding the attributes

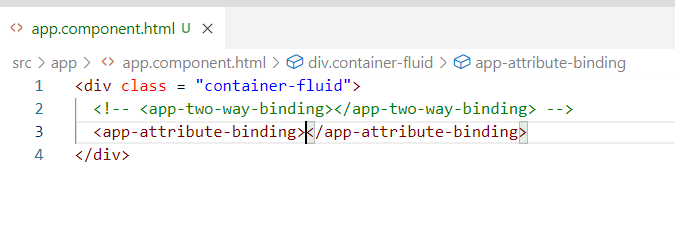
We need to use attr.attribute-name to bind the attributes

Note: Use attributes binding for the elements when there’s no DOM property available for the elements

Create component attribute-binding



Add this to the root component

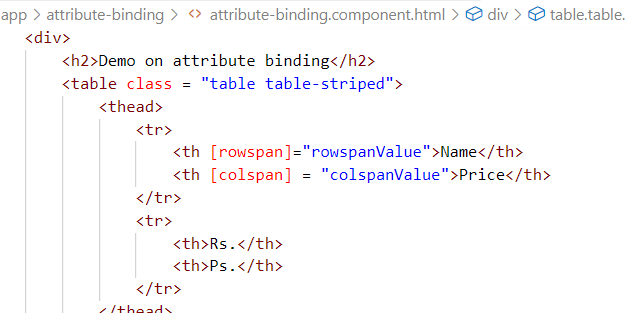


attribute-binding.component.ts

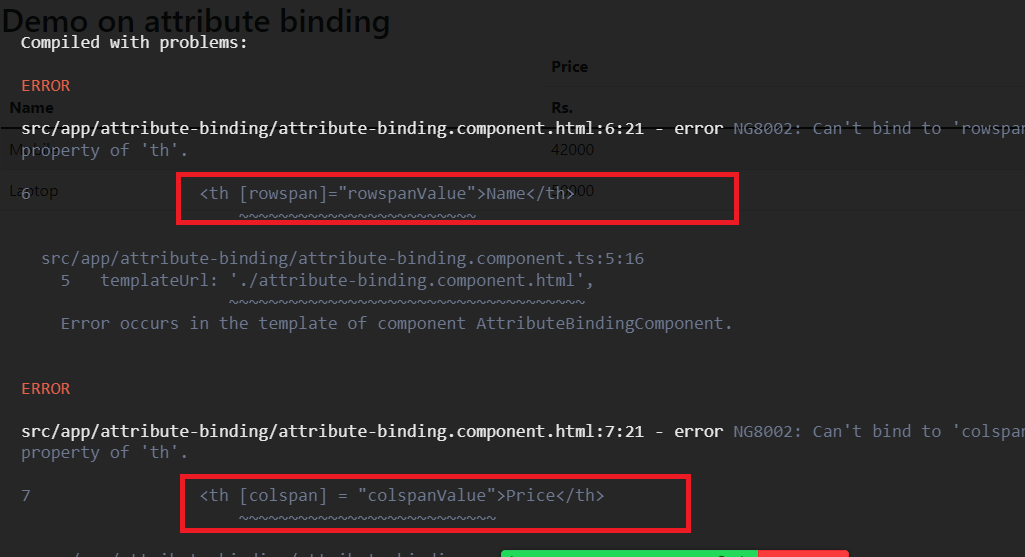


If you try to bind the colspanValue & rowspanValue to the <th> or <td> through Property binding you will get error.

i.e.,



The error will be

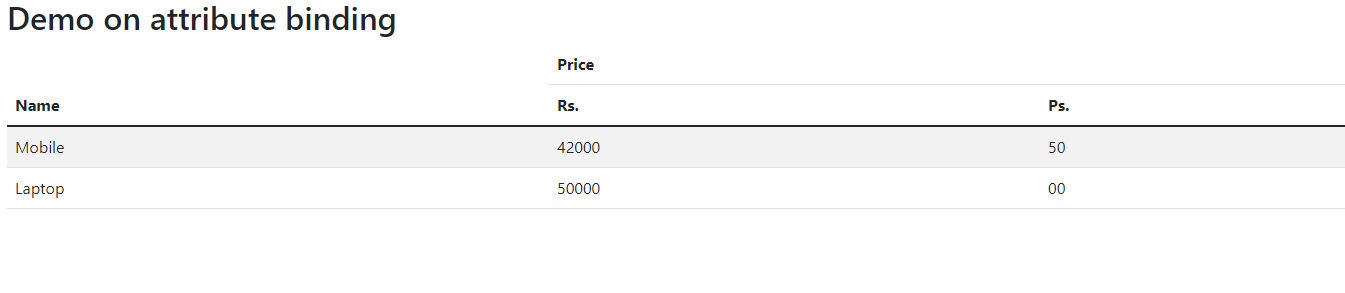


This error is because there’s not corresponding DOM properties for these attributes which some times works in the cases where you use attributes like hidden, disabled, style, class, because for all these attributes there are DOM properties available with the same name, but you don’t get DOM properties for the colspan & rowspan, hence you can’t bind with property binding syntax i.e, [colspan], [rowspan] which is a property binding means DOM property it binds.

To avoid error change attribute-binding.component.html

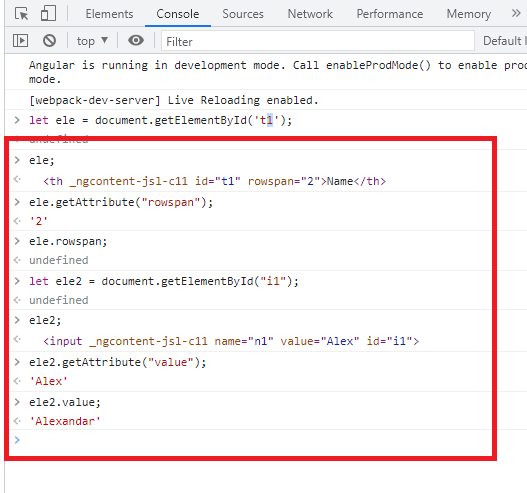


Output:



Attributes vs DOM properties

Attributes can’t be changed it is fixed however DOM properties can be changed at runtime



Here, ele is a reference to the <th id = “t1” rowspan = “2”>, since rowspan is an attribute you could see 2 in the ele.getAttribute(“rowspan”), but ele.rowspan gives undefined, because rowspan is not available as a DOM property

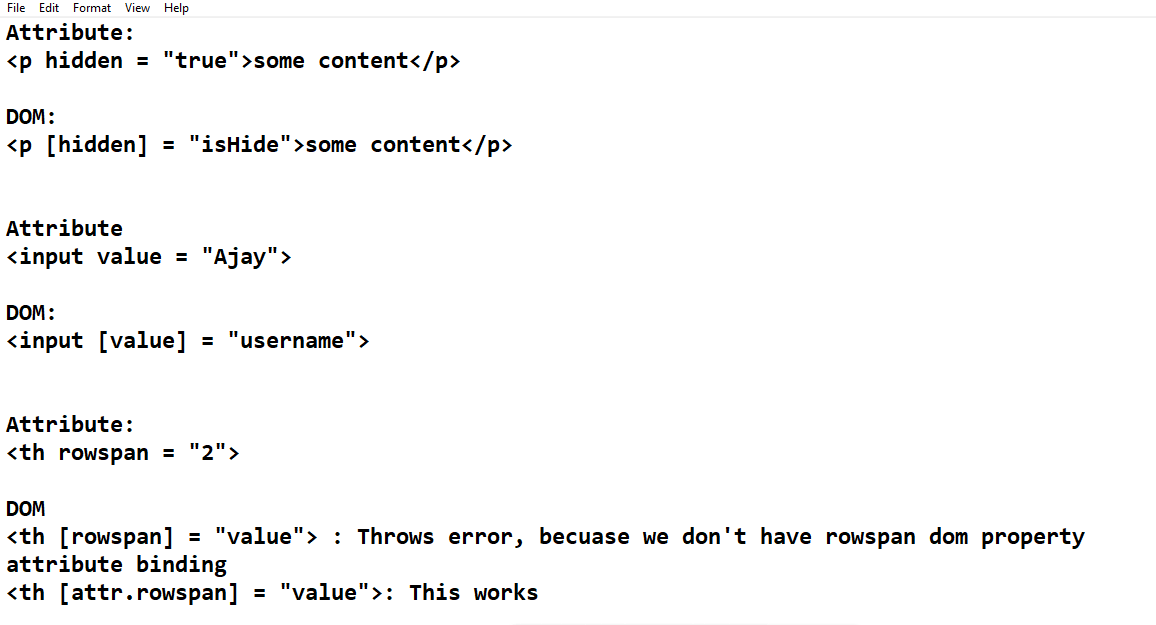
Similarly

ele2 is a reference to <input type = “text” id = “1” value = “Alex”>

ele2.getAttribute(“value”): will always give Alex

However ele2.value is a DOM property it gives the value you entered in the DOM ie.., in browser, since Alexandar is entered ele2.value gives Alexandar, we have DOM property for value attribute hence we got its value.

We can do a Property binding only for the DOM properties, but if the DOM properties are not available we can’t bind we get the error

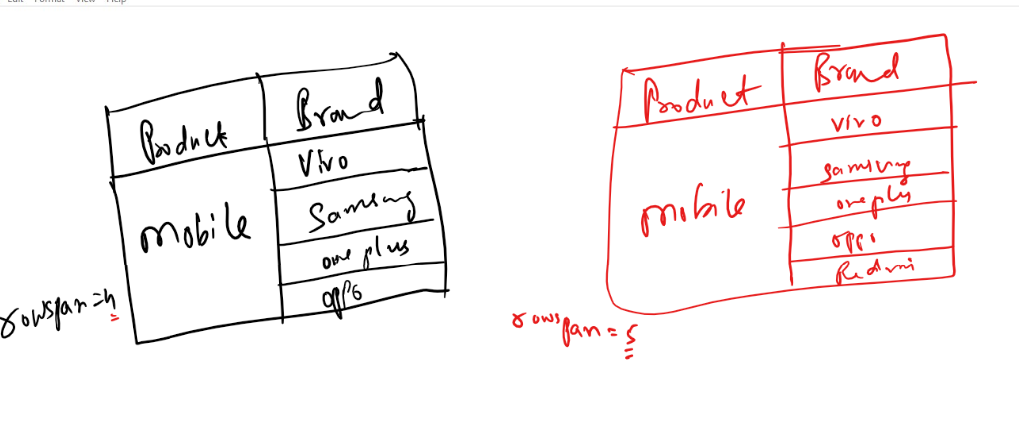


When to go for rowspan & colspan attribute binding

When the table rows & cols are decided at runtime and the spanning of some rows and cols need to be dynamic in that case you can use rowspan & colspan attribute binding

ex: Suppose you have a Mobile with different brands and all the brands should span to the Mobile then the Mobile must span according to the number of brands.

i.e.,



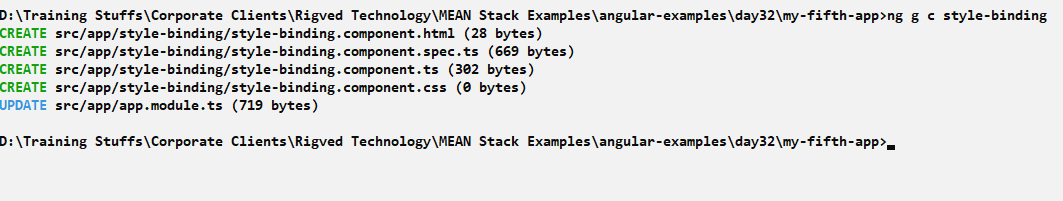
Style binding

It is also a type of property binding where you can add CSS to the style property

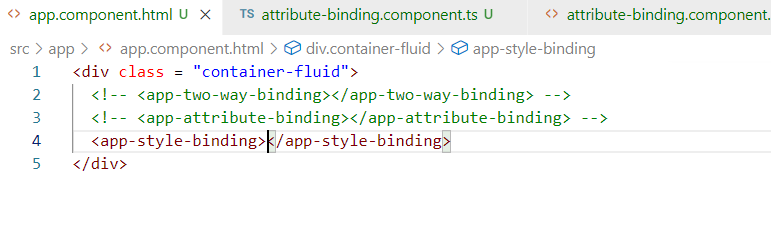
<p [style] = “myStyle”>

Here [style] is used to bind the style property and myStyle will have a value of CSS for the style like {“color”:”blue”, …}

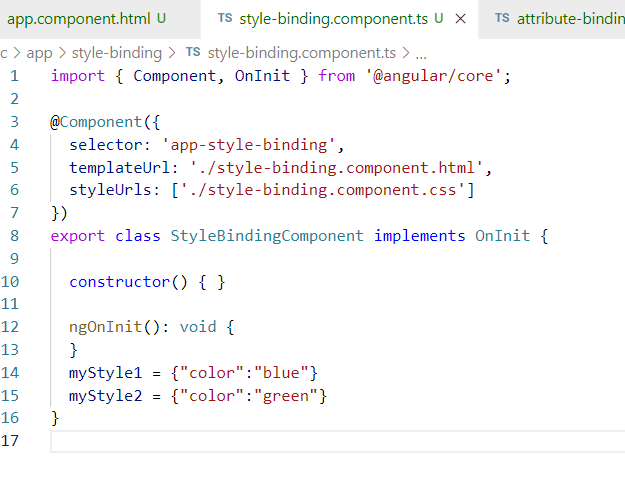
Generate style-binding component



Add it to the root component



style-binding.component.ts

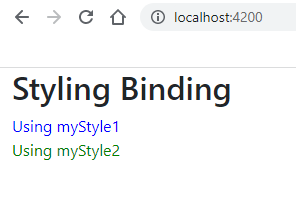


myStyle1 & myStyle2 can be used to bind the [style] property of the view template

style-binding.component.html



Output:



Class binding: It is used to add/remove the CSS classes defined in the CSS file, here you would set of css classes in css files that can be added or removed in the view template,

In case style binding the value will be provided by the component class, however in case of class binding the value is provided by CSS file with the help of true/false value from the component class, true means add the CSS & false means remove the CSS

CSS file

.rc { color : red }

Component file

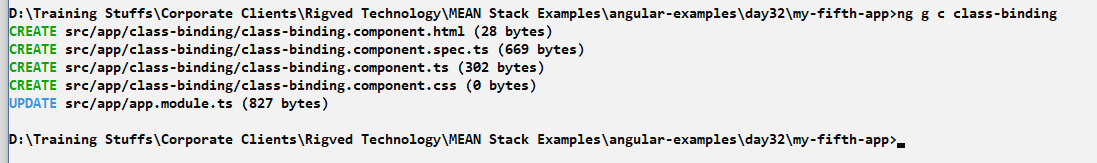
apply = false or true

HTML file

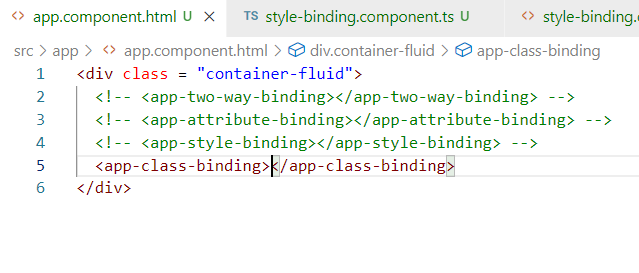
<div [class.rc] = “apply”>

If apply is true then rc class is added else rc class will be removed

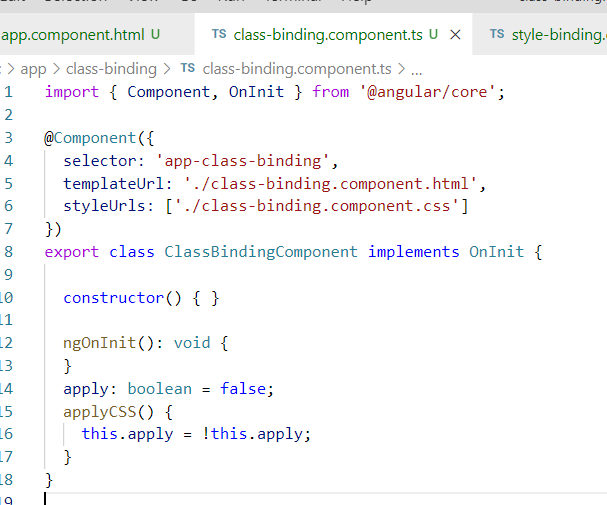
Generate the class-binding



Add the class-binding to the root component



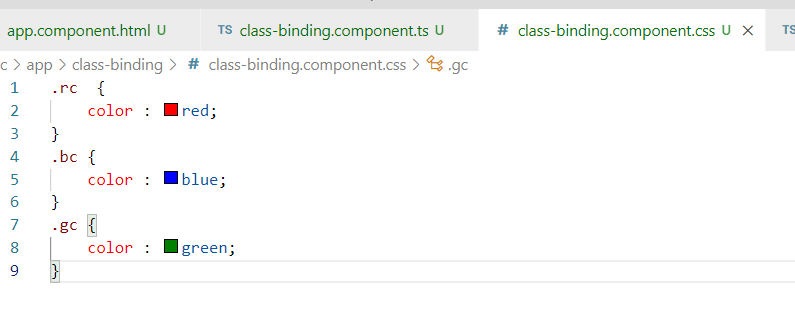
class-binding.component.ts



Here call on applyCSS changes value of apply to true/false alternatively.

Now you can add a class in the CSS file

class-binding.component.css



class-binding.component.html



Here !apply reverse the value of apply

Output:



Activity:

1. Try all the above activities
2. Create a component to show the students record with marks, the student record must be shown in a table along with rollno, name, marks, the marks data must be have css style with red or green based on the value of marks, below is the student data you can use

students = [{rollNo:1, name: Ajay, marks: 40},{rollNo:2, name: Vijay, marks: 30},{rollNo:3, name: Manu, marks: 60},{rollNo:4, name: Sachin, marks: 20},{rollNo:5, name: Amit, marks: 70},{rollNo:6, name: Zaheer, marks: 80}]

Consider above student data and show the marks in red color when the marks is less than 40 and in green color when marks >= 40

1. Create a component that shows the product information with product name, brand name and spans the product name based on the number of brands you have, you can consider below products array to show the product name & brand name in the table format

products = [

{name: "Mobile", brand: ["Vivo", "Oppo", "Oneplus"]},

{name: "Laptop", brand: ["HP", "Dell", "Lenovo", "Acer"]}

]

The table must show name and brand by allocating the rows to product names as below

|  |  |
| --- | --- |
| Product Name | Product Brand |
| Mobile | Vivo |
| Oppo |
| Oneplus |
| Laptop | HP |
| Dell |
| Lenovo |
| Acer |

Day 33:

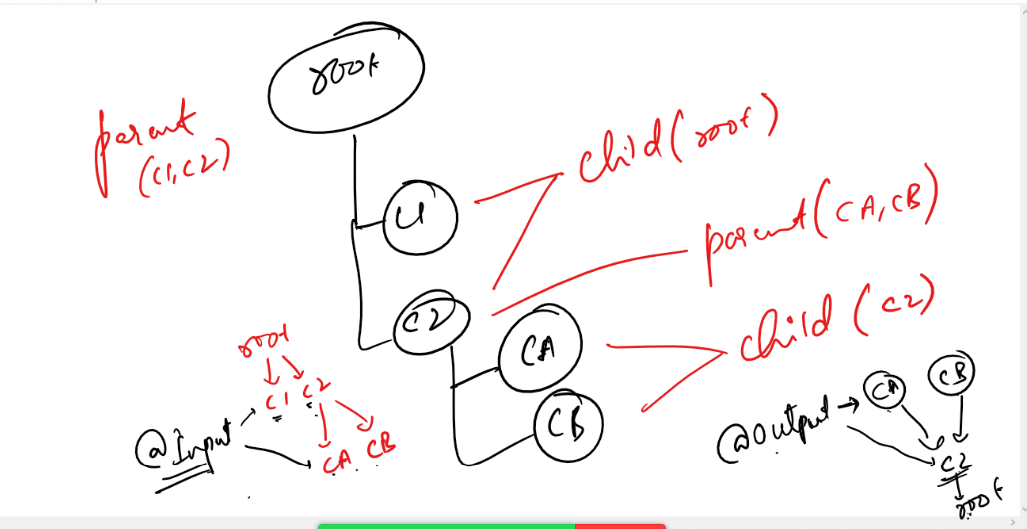
@Input() & @Output()

These are the decorators used to share data between the components i.e., from one component to another component like parent to child and child to parent components

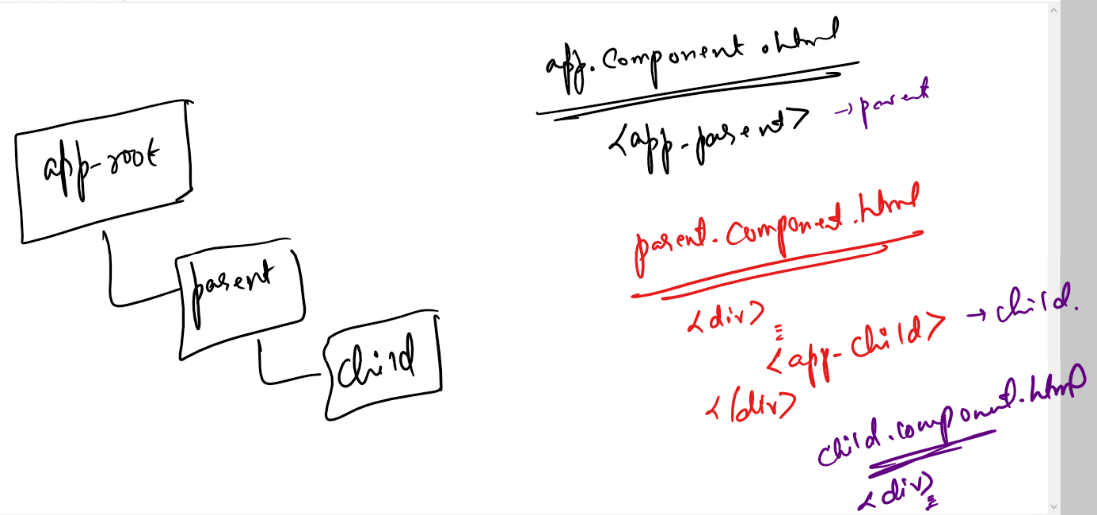
These two decorators helps a component enclosed in another component to share the data and vice versa

@Input(): It shares the data from parent to child component

@Output(): It shares the data from child to parent component

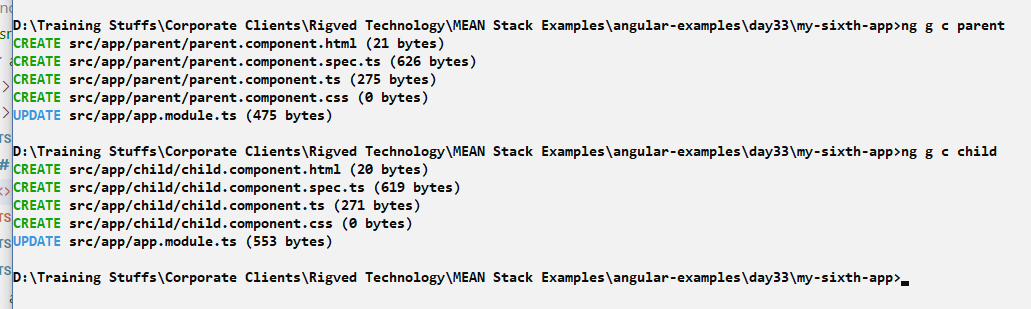


You can nest the child to parent in the view template as below

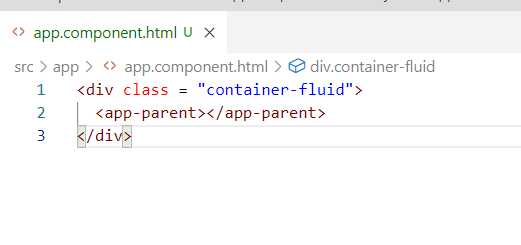


Generate parent & child components

Add the child component to parent component & parent component to root component



Add parent to root



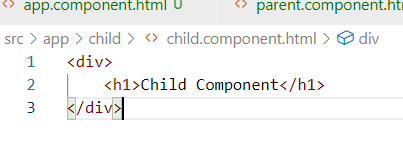
Add child component to parent component

parent.component.html

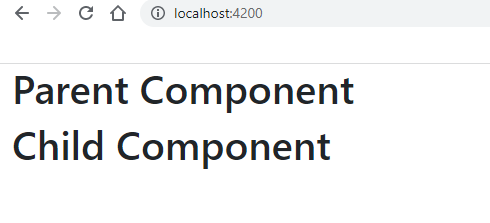


Modify the child component to have some heading

child.component.html



Now the output looks like this



@Input(): It is a decorator used in the child component to receive the data from the parent component, it must be written on a variable present in the child component

@Output(): It is a decorator used in the child component to send the data to the parent component, it must be written on a variable present in the child component.

Note: Both decorators are written in the nested component

Using @Input

parent.component.ts



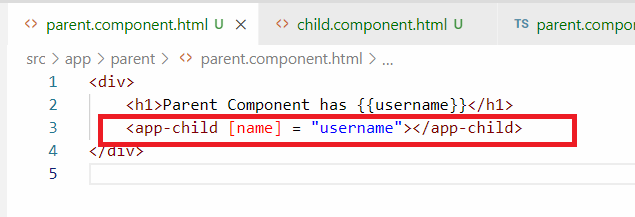
The parent component has a username property, this username can be passed to child component property i.e., name via <app-child [name] = “username”>

child.component.ts



Here the @Input() name would receive the value from its enclosing component

parent.component.html

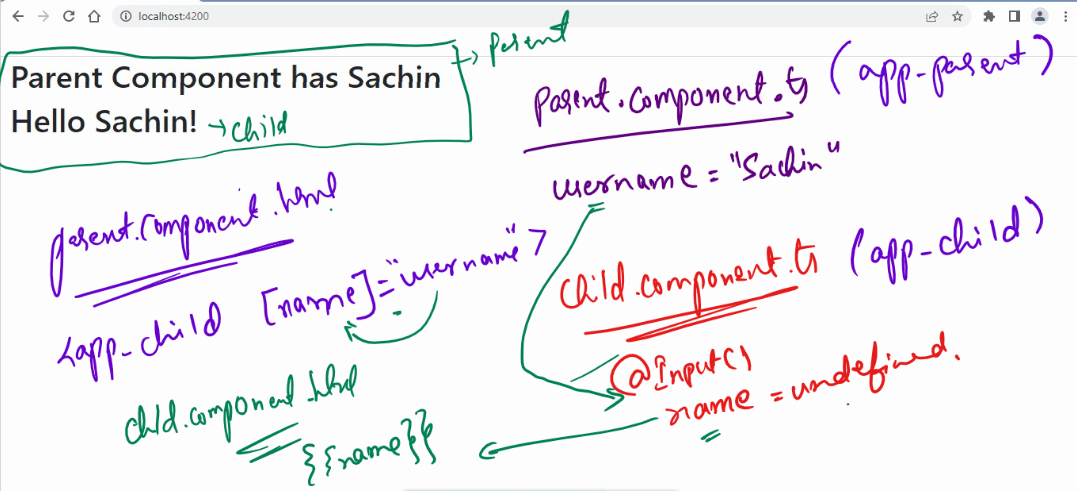


The username is passed to [name] of the child component, the parent component also displays child component content i.e., Hello {{name}}

child.component.html



Output:



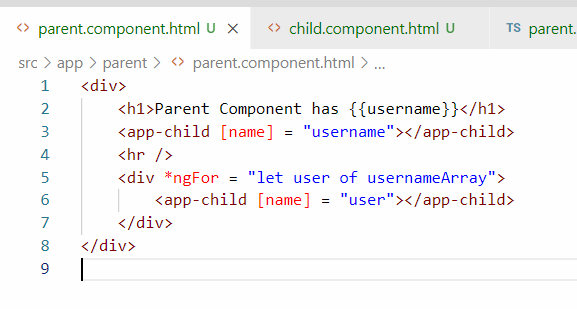
Here the child component is taking care of printing the name, so from parent you can also supply any number of names to the child component, using arrays

parent.component.ts



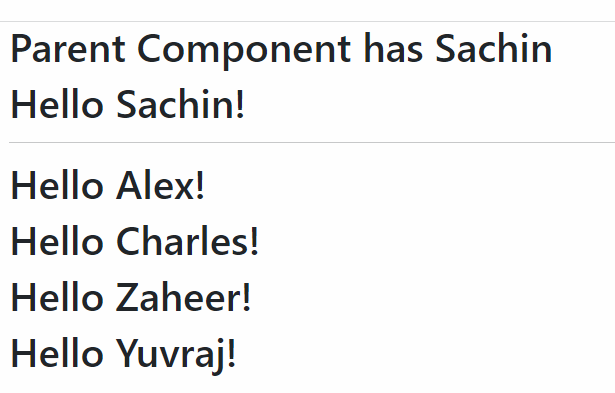
Now you can iterate usernameArray and supply iterated item to child component

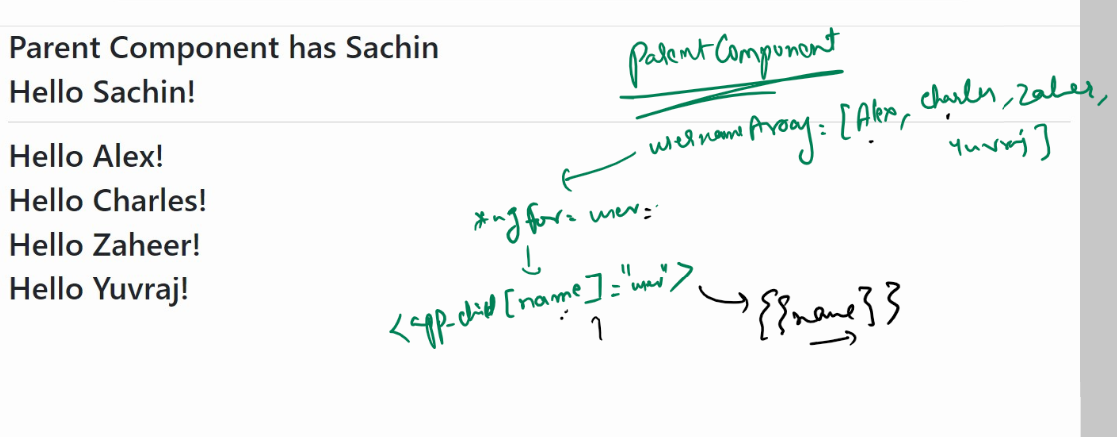
parent.component.html



The <app-child [name] = “user”> is part of \*ngFor so it supplies each item to the [name] of child component and prints that name, since there 4 items in the array it prints 4 different names

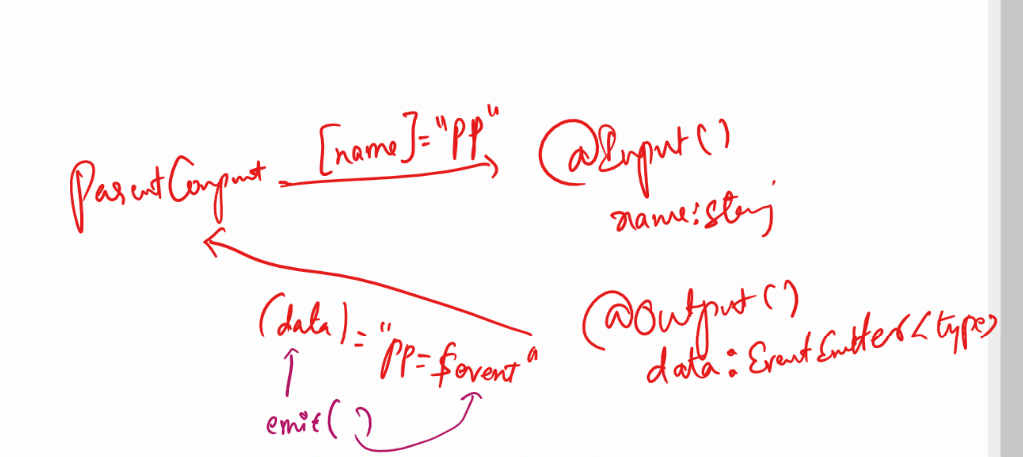
Output:





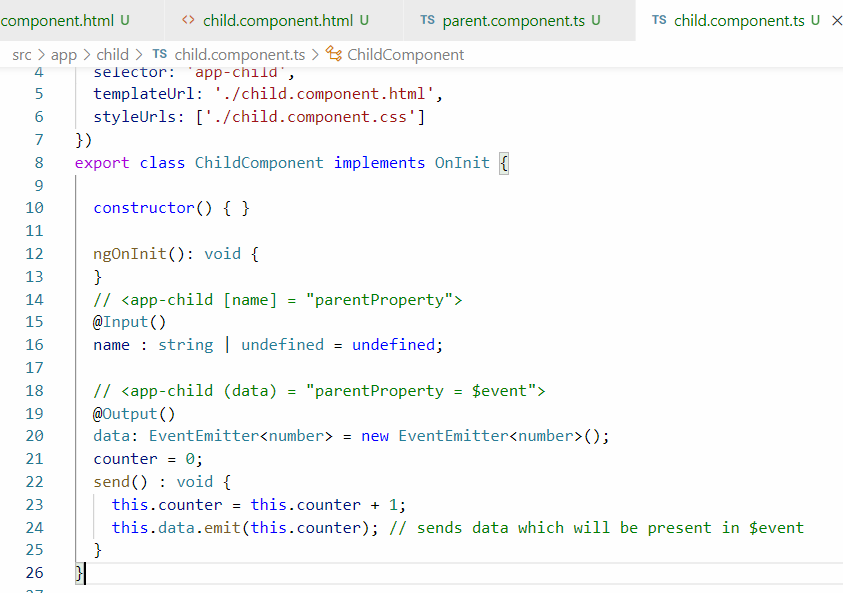
@Output:

It is used to share the data from child component to parent component, but it has to be shared by emitting the event object



The child component can share data to the parent component with the help of EventEmitter, the EventEmitter will mention what type of value it can emit, the value will be stored in $event which can be used in parent component

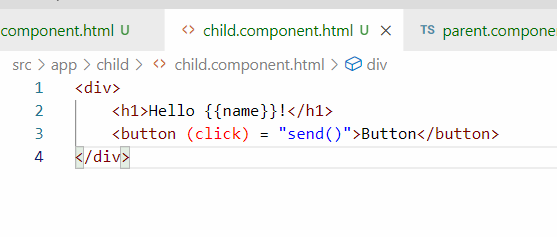
child.component.ts



Here the send() must be called to generate an event in the data property, the EventEmitter<number> suggests that data.emit() must emit a number type value, hence the counter is used in data.emit().

Now the child component must call send() function then only the event in data will be emitted.

child.component.html



Here the child component only updates its property but to send value to another component we need EventEmitter because that EventEmitter variable will be used in the parent component.

parent.component.ts

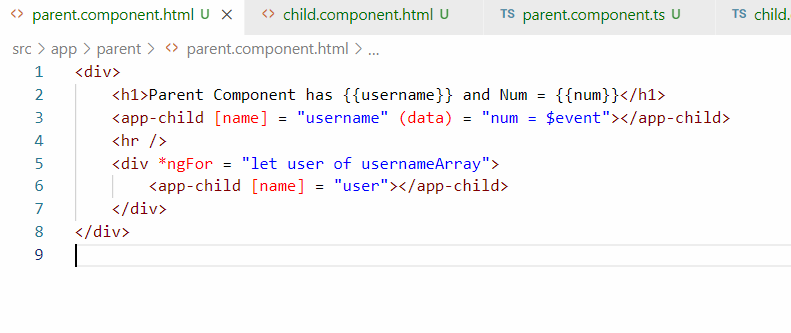


Here the variable num would get the value from the child component, since child component is nested in parent component, the parent component will use the EventEmitter variable of child component i.e., data as

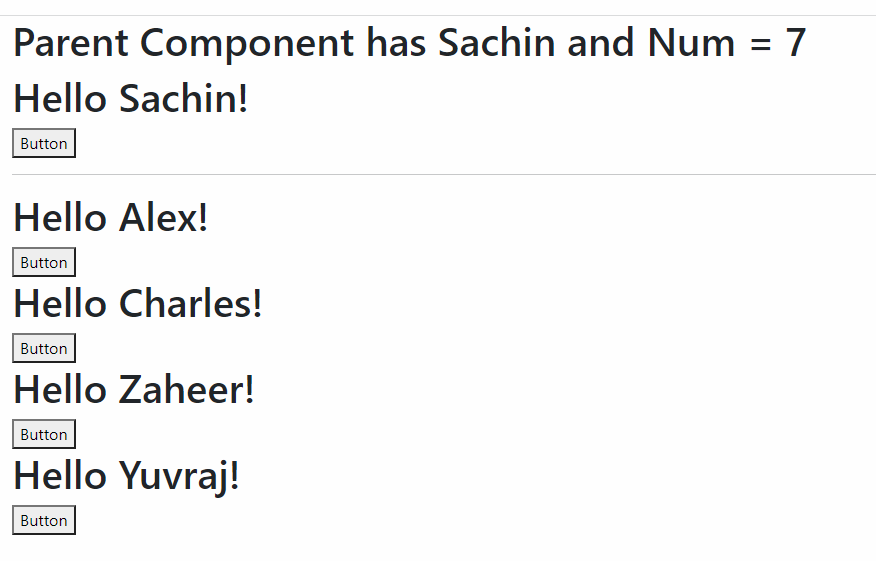
<app-child (data) = “num = $event”>

The data is an EventEmitter type present in child component this generates an event when emit() is called through send() function, the value will be stored in $event that is assigned to parent property to finally pass data from child to parent.

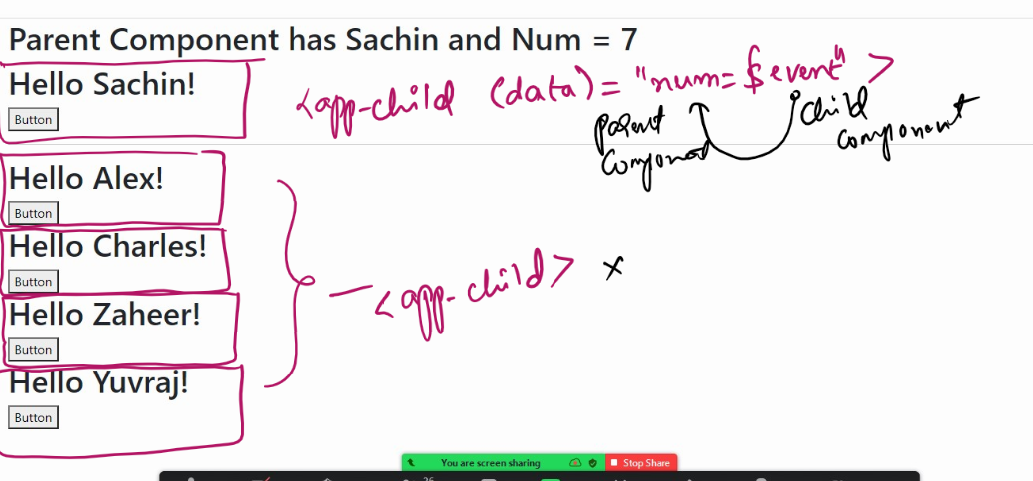
parent.component.html



Output:



Since the parent component does event binding to the (data) only in one child component not in other child components created in the \*ngFor the $event will supply value only from one child component and other child components created \*ngFor doesn’t pass data to parent.



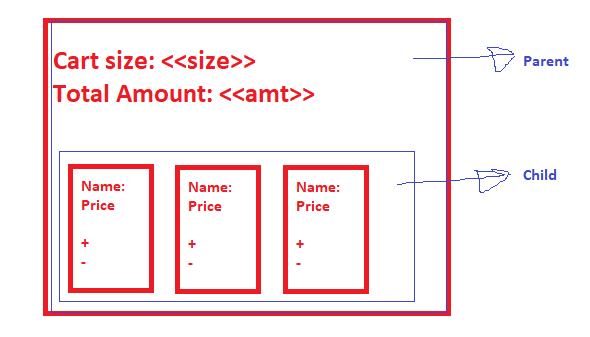
When @Output() is useful

Suppose you have a parent component that nests multiple components of product, id each product you select wants to add in the cart, then you can send those product to the cart which is present in the parent, when you see the cart you must see all the products selected.

EventEmitter: It is a type that can send data of any type through emit(), you can create EventEmitter for number, string, user defined class, any, [], so on

Activity:

1. Try the above @Input() & @Output() example
2. Create a simple order tracking application, where you will have two components one shows the cart size and total amount and another shows the list of products.



The child component must have predefined products in an array that you can display with name, price and two buttons to add or remove, the predefined products could be:

*products = [{name: ‘Laptop’, price: 52000}, {name: ‘Mobile’, price: 30000}, {name: ‘Gyser’, price: 50000}]*

The child component will have the above products array & displays each product name, price with two buttons + & -, the + must send the product to the parent component that will show the cart size & total amount by calculating the products the cart has.

i.e., if you select two products from the child component then the cart size must be 2 and total price should be sum of both the product price, you can also select same product more than once.

Angular Forms

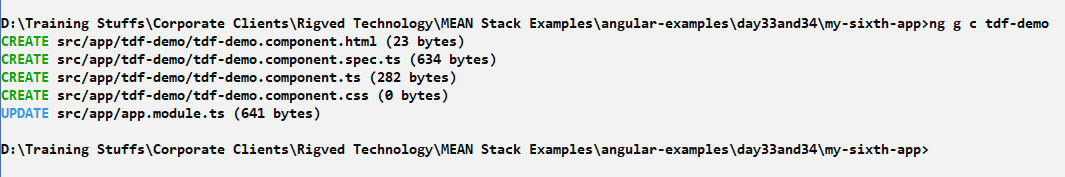
Forms allows user to enter the data that can be used to perform many operations like storing, retrieving, updating & deleting the resources in the backend, angular provides two types of forms

1. Template Driven Form: Form data is controlled in view template
2. Reactive Forms / Model Driven Form: Form data is controlled in component class

Template Driven Form:

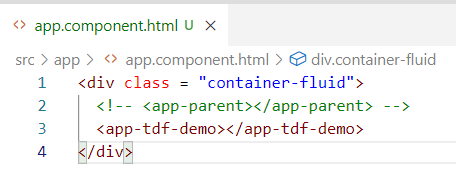
It uses [(ngModel)] to control the form data along with [(ngModel)] it uses some directives of FromsModule like ngForm.

Generate a component tdf-demo



This component we will use to create template driven form.

Add this component to the root component



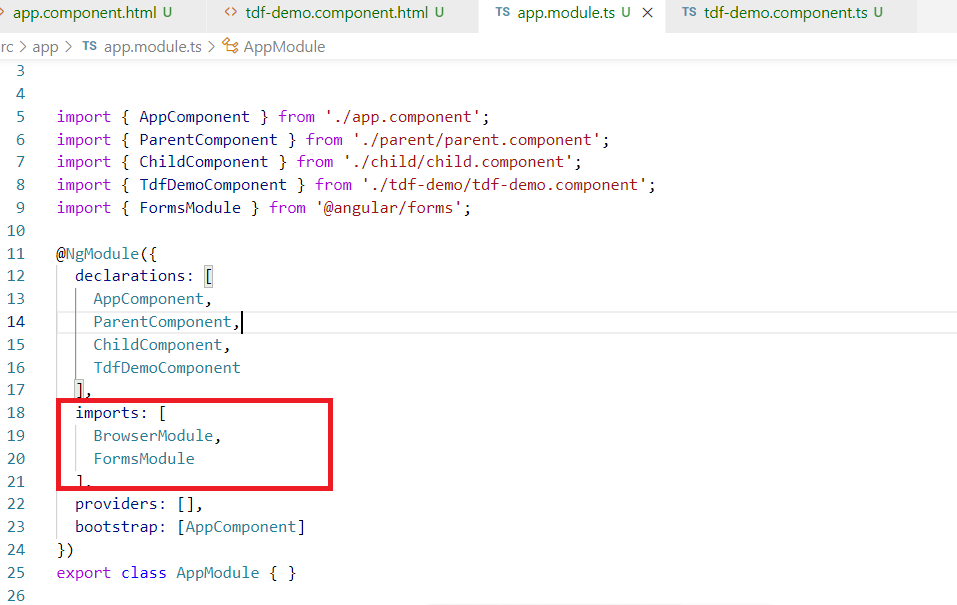
tdf-demo.component.ts



This component class must take the data from the view and send them to the backend, but as the backend services are not available currently, we will just print the data in the console.

ngForm is part of FormsModule that needs to be added in AppModule

app.module.ts



ngForm creates a NgForm object that initializes the form data with the help ngModel, so that you don’t have to track each input controls

tdf-demo.component.html



Here #user = “ngForm” and ngModel in each <input> element takes care of initializing the NgForm object



Now on submit you can use user.value so that the component class gets {fn:value, ln:value, ph:value}

Output:

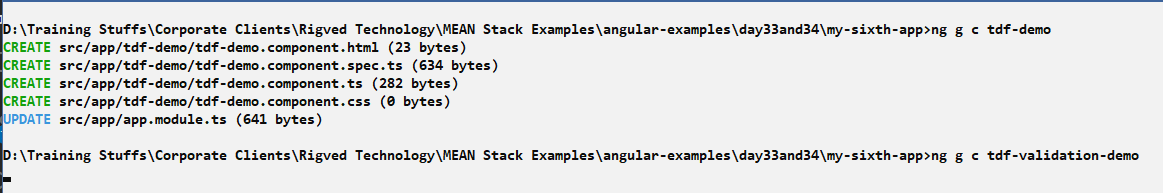


The console output is because of formSubmit(formValue:any) printing in the console.

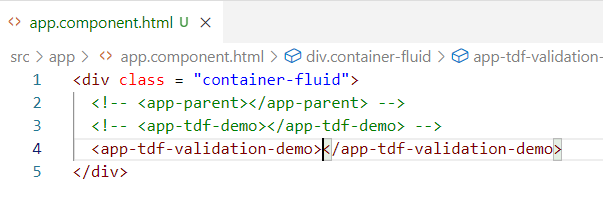
Form Validations

Form Validation allows user to prevent from invalid data could be empty input or the size of the input doesn’t meet certain criteria

Generate a tdf-validation-demo component



Add this component to the root component



tdf-validation-demo.component.ts



Here the handleSubmit() must be called only after the form is validated, i.e., only on valid form data the form must be submitted.

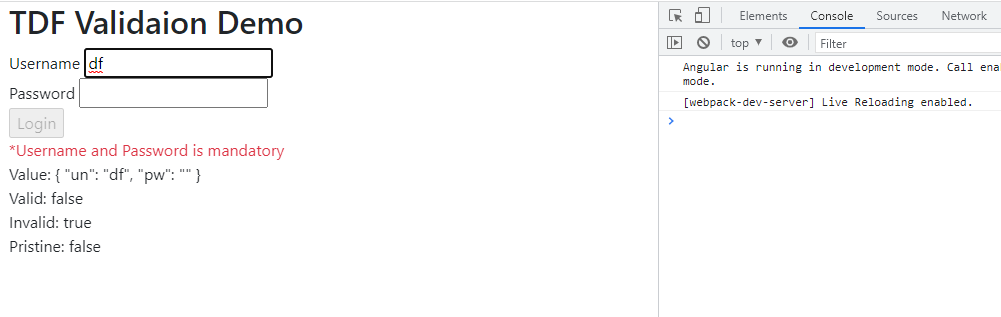
You can use NgForm valid, invalid, pristine properties to validate the form data

pristine: It will be true when the controls are not modified, if modified it will be false

tdf-validaiton-demo.component.html



Output:



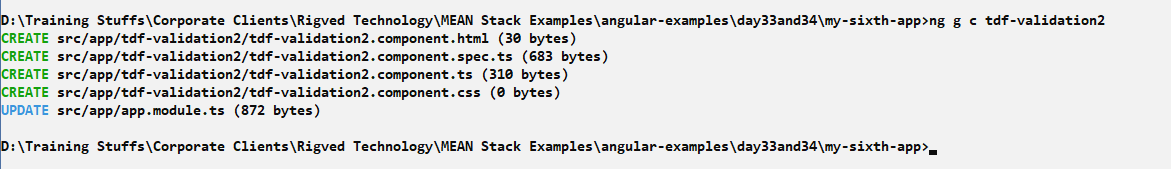
The above code shows error messages whenever any one of the input control is invalid, but you can create a template where for each input you can have a validation error messages

To handle each input element we need to use template reference that must be initialized with ngModel

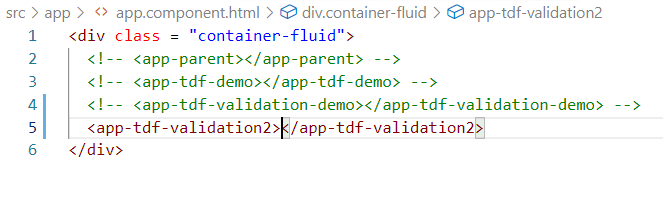
<input #tf = “ngModel”>

This above code will create an object of template reference from which you will get various properties of an object like

valid, invalid, value, pristine and so on



Add this component in the root component



Create a function that is called only when form is valid

tdf-validation2.component.ts

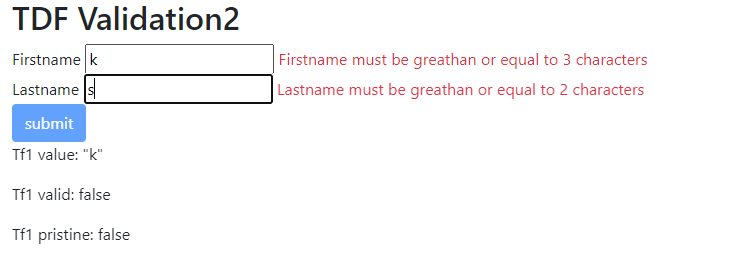


Now you can validate each input and show the error message for every invalid input element

tdf-validation2.component.html

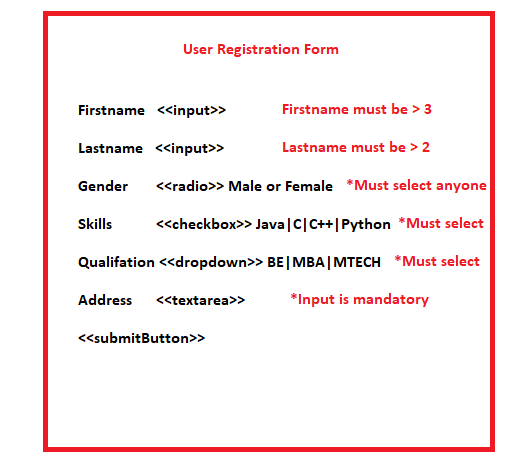


Output:



Activity:

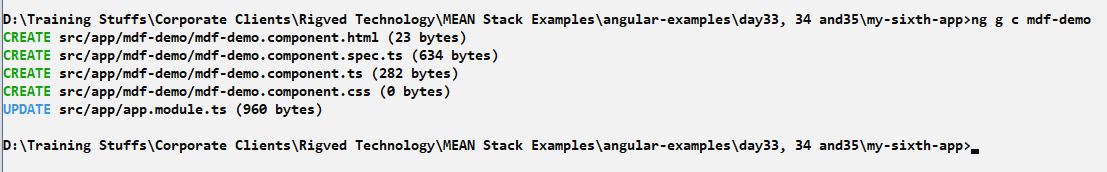
1. Try the above activity
2. Create a Registration form which will have some user information’s and once all the inputs are valid display those data in the view template.



The submit is disabled by default until all the input is entered and once all the input is valid, you can click on submit and display those information’s below the form

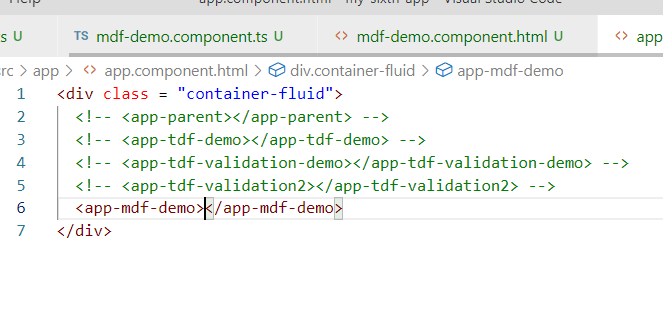
Model Driven Form or Reactive Forms  
This handles form data in a typescript code instead of view template, this can have more control over the form, it doesn’t use any two way binding feature, but it will keep view and component class in sync, to use Reactive forms you need to use a module called ReactiveFormsModule of @angular/forms

The ReactiveFormsModule gives us some form control objects and form groups using which you can handle form data, with ReactiveForms the view template will have very less code in terms of handling data, so you can concentrate more on the designing



Add the mdf-demo to the root component

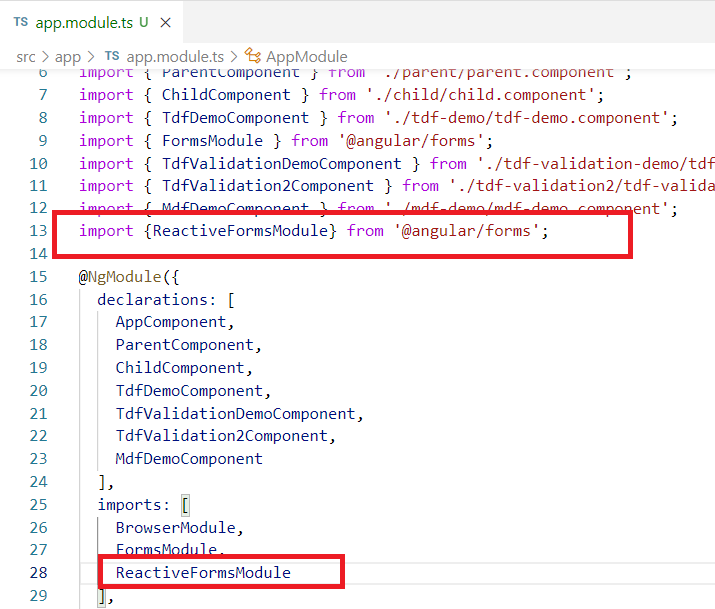
app.component.html



We will use FormControl to handle the form data, FormControl is an object that refers to the form control in the view template

Note: To use FormControl we need to add ReactiveFormsModule in the AppModule.

app.module.ts



The ReactiveFormsModule will handle keep view and component class in sync without two way data binding syntax i.e., [(ngModel)]

Now you need to use the instance of FormControl(‘’) that is referenced by the view template form elements, so that without any extra code in the view template the data will be available in the component class.

In the component class we need a form control object

username = new FormControl(‘’);

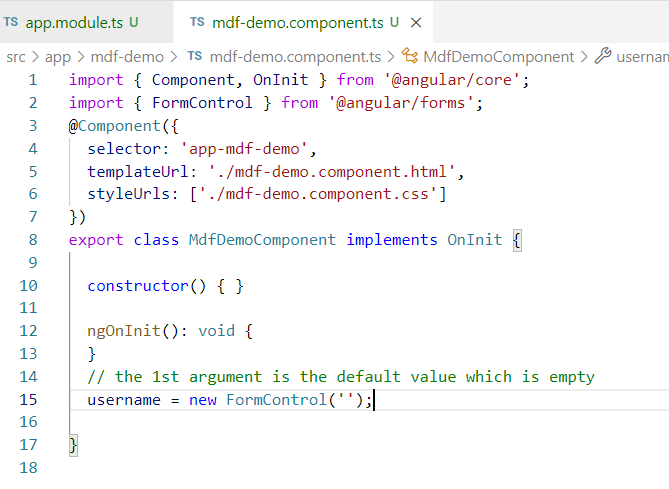
In the view template we can bind this username through [formControl] = “username” in the form <input> element.

<input [formControl] = “username”>

Now the View and Component class will be in sync without any two way binding syntax

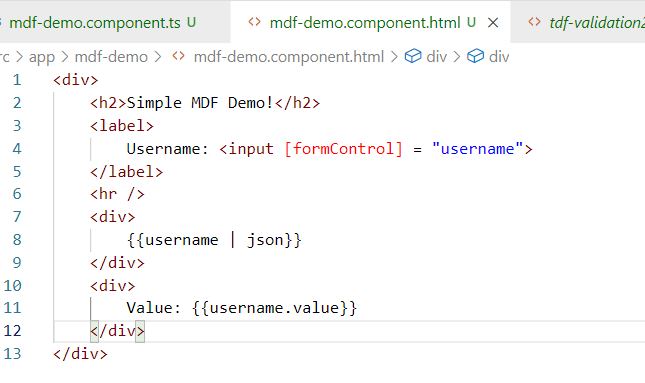
i.e., <input name = “un” [(ngModel)] = “username”>, here if you need any kind of validation then you need to add them in the <input> and also to validate the <input> you need to use a template reference i.e., <input name = “un” [(ngModel)] = ‘username” #tf = “ngModel”

mdf-demo.component.ts



Now you can use username in the html with [formContro] = “username” so that it is bound to the FormControl

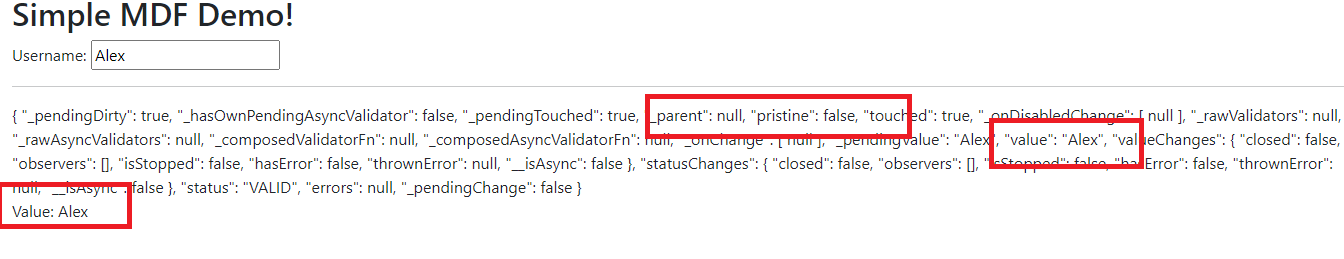
mdf-demo.component.html



Output:



Output:



Since username is FormControl, we can use {{username | json}} that shows all the FormControl properties, like pristine, value, hasError, status and so on.

This reduces you to write template reference, ngModel attributes in the form that you used in template driven form.

Activity:

1. Try the above FormControl activity
2. Create two FormControl and show their values in the view template, 1st Form Control use firstname & 2nd Form Control user lastname, and display their values in the view template (i.e., HTML)