Angular Framework

Angular is a framework used to develop single page applications using HTML & Typescript.

Single Page Applications are applications that makes user to see everything in one page, it only loads part of the page instead of reloading the complete page whenever user interacts with the application.

Ex: Gmail, Facebook, GitHub, Angular.io, ...

Angular uses component to render the data, components are UI’s which can be independently built and reused in other components. They are build without knowing where it will be displayed

Ex: Facebook has Profile component, that can be reused in comments, likes, newsfeed and so on.

Angular uses HTML & Typescript to develop the application.

HTML: It is used display the content

Typescript: It is used to write logics in angular, like fetching the data from server and passing the data to the HTML, deciding what components to be loaded and so on.

Typescript is a super-set of Javascript, it is less error prone compare to javascript, Typescripts are compiled to javascript.

* Typescript more safe compare to javascript
* It supports all the features of Javascript, including the new features of Javascript (released by ES6 in 2015)
* ES means EcmaScript which is a standard or specification for Javascript, ES is not a programming language, however Javascript implements the ES standard

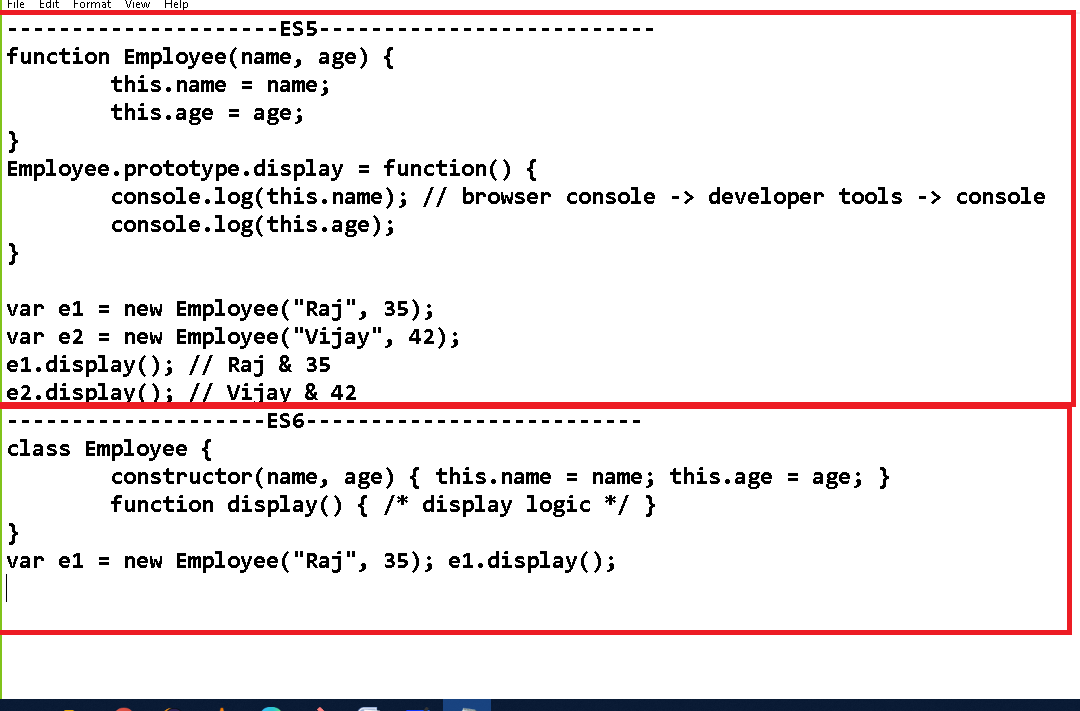
Softwares required for Angular application development

* Visual Studio Code - Editor
* Node.js - Runtime environment to run javascript code & also angular code
* Angular CLI - toolkit for angular applications

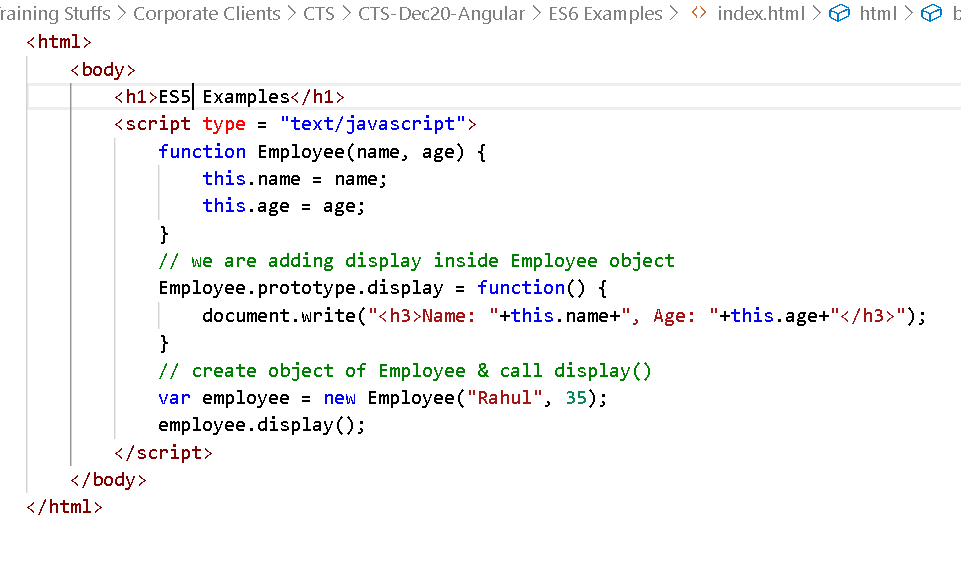
ES6 features:

There are few new features added that simplifies javascript coding

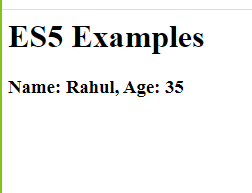
* let, const, class, constructor keywords
* template strings
* arrow functions
* rest & spread operators



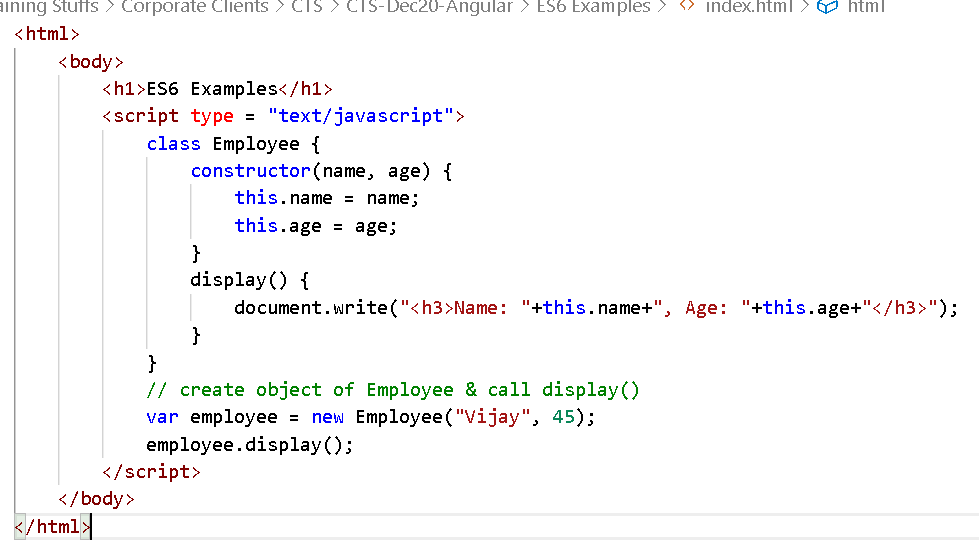
Creating function & object in ES5



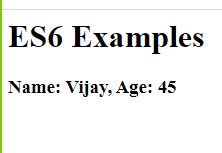
Output:



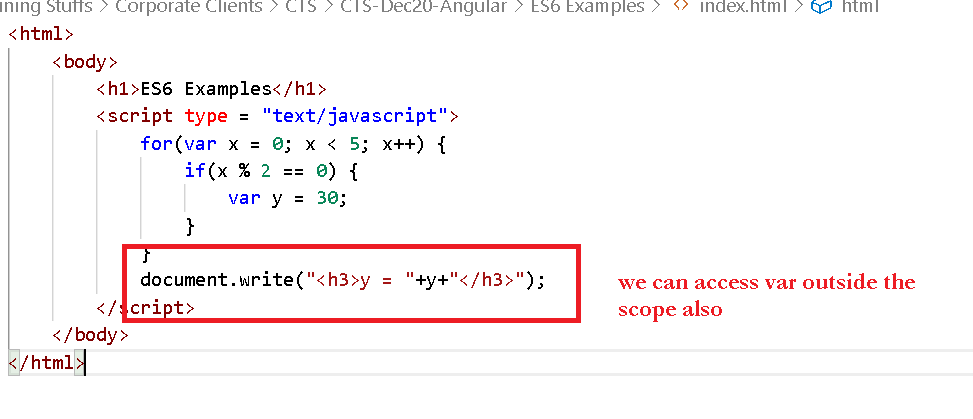
ES6 uses class & constructor keywords



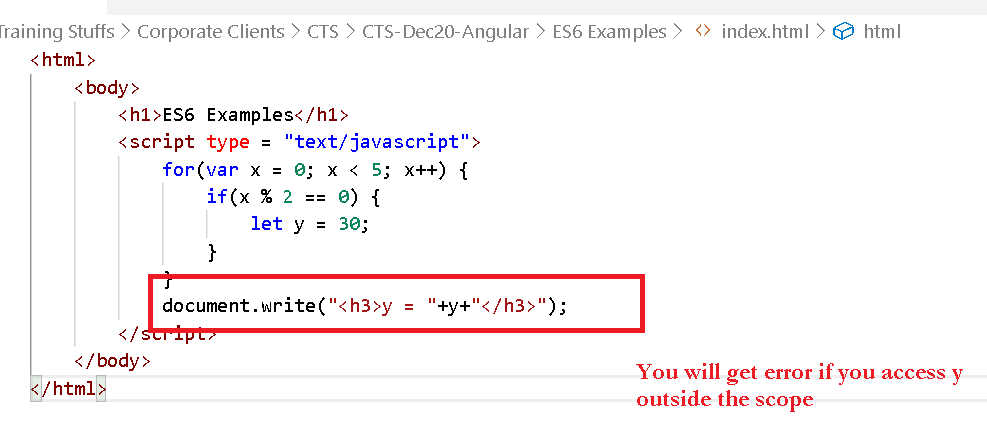
Output:



let, const keywords: these are used to create block scoped variables, because in javascript you were creating variable with ‘var’ which is not part of any scope.



But with let & const you can create block scope variables



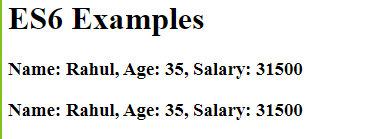
Template String

It allows you to create a string & access the data without breaking the strings using + operator.

Template strings uses a backtick(`) i.e., a quote below the Esc key.



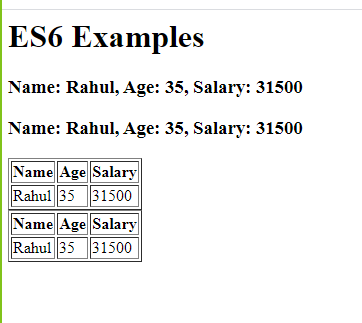
Output:



You can’t give line break without + in “ or ‘ quotes but you can give line break in ` without + operator

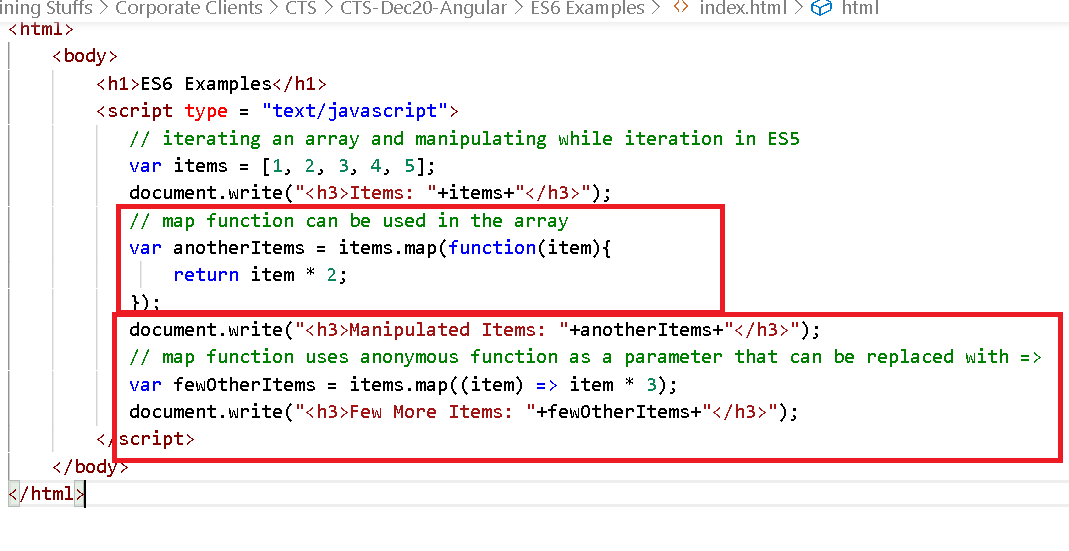


Output:

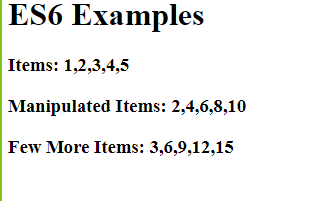


Arrow functions(=>):

These simplifies writing anonymous functions in javascript where they are having few lines of expressions.



Output:



Rest & Spread operators

When a function doesn’t have an idea how many parameters are supplied by a caller you can have rest operator, and also when an input has to be spread across multiple parameters of a function you can use spread operator.

add(4, 3, 1, 2);  
add(4, 3);  
add();

Rest operator:

function add(...x) { .... }

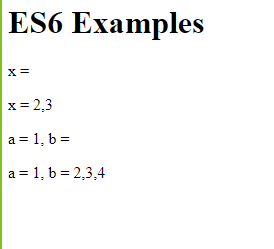
The add function takes test parameter means can accept any number of parameters from 0 to many

function test(a, ...b) { ...}

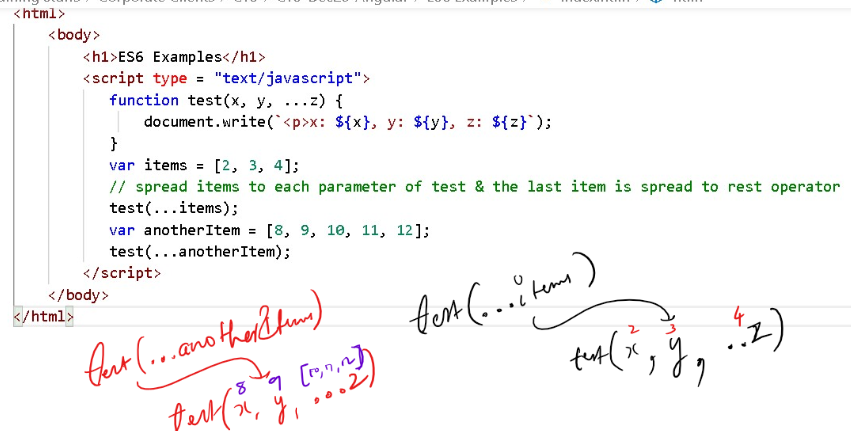
The test function takes 1 parameter minimum followed by 0 or many parameters



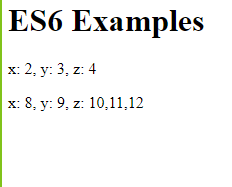
Output:



Spread operator



Output:



Typescript: It is a super-set of javascript, which supports all the javascript features including that it has some additional features

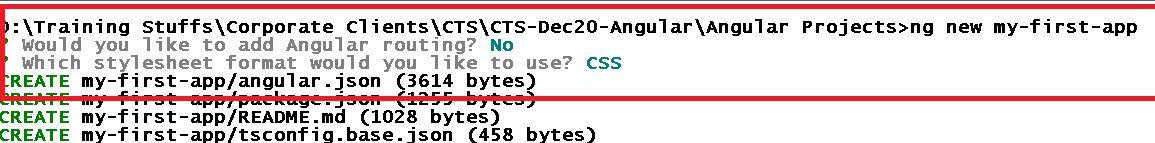
1. types for variables
2. return types for methods
3. short-cut initialization of constructor
4. import & exporting classes

Installing Angular Toolkit:

Command: *npm install -g @angular/cli*

How to create angular project

Command: *ng new my-first-app*



How to launch the application

Command: *ng serve*

How to add a component in another component

Angular gives you a command to generate the component, if you use this command it will automatically create 4 files

1. component class ts file
2. component html file
3. component css file
4. component spec file (testing purpose)

Updates the Application that component is created

Command: *ng generate component <<component-name>> (or)*

*ng g c component-name*

How to change the port

*ng serve --port=4201*

Angular Building Blocks:

These forms the angular application, the are:

1. Components: @Component({})
2. Modules: @NgModule({})
3. Pipes: |
4. Data-binding
5. Directives: ngFor, ngIf, ngSwitch
6. Services: @Injectable()
7. Routers: Navigations

Inline template

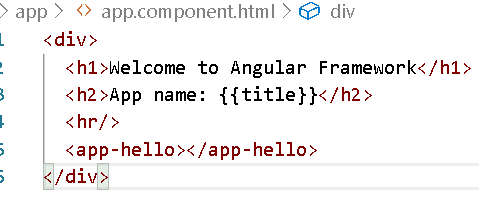
It is going to provide content in the @Component() decorator itself instead of a separate html file i.e, templateUrl.

@Component({  
 template: ‘content’  
})

hello.component.ts



Add app-hello in root component



Output:



Different way of maintaining the data

Since you are using typescript, you can mention types

title : string = ‘string value’

employee : any = {“id”:1000, “name”:”Raj”}

employee : Employee = new Employee(1000, “Raj”);

employee : Employee[] = [array of employee objects]

hello/hello.ts



You should import Hello from hello.ts using

import { Hello } from ‘./hello’

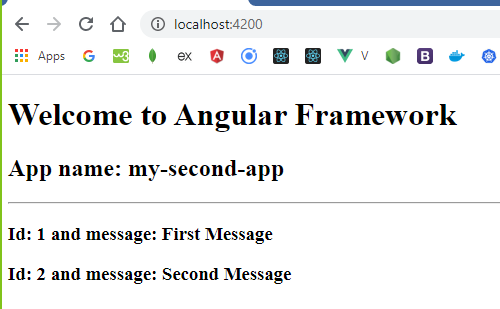
hello/hello.component.ts



hello/hello.component.html



Output:

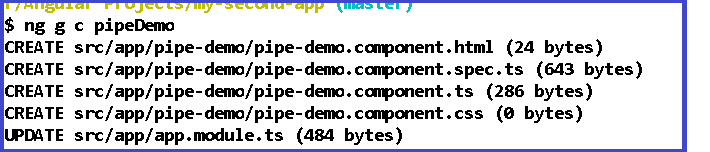


Angular Pipes(|):

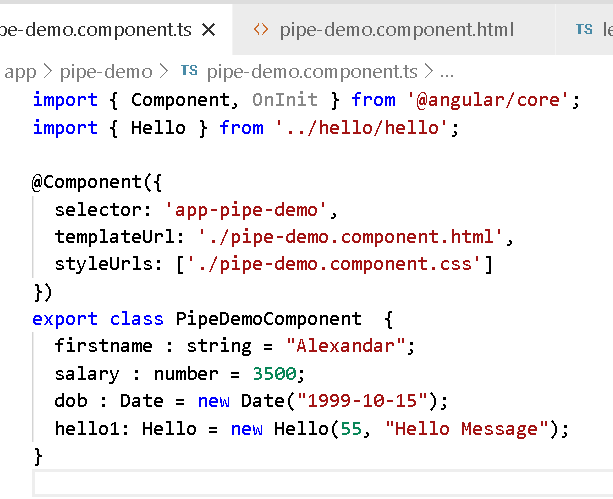
It is used to format the output, we have many inbuilt pipes like

* uppercase
* lowercase
* date
* json
* currency

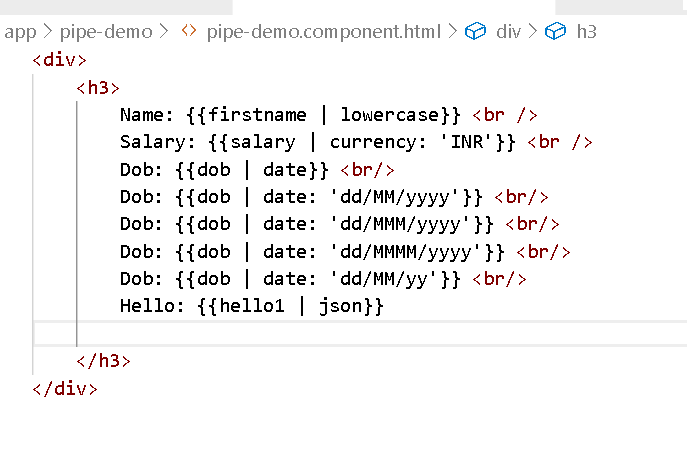
We can also create custom pipes by implementing PipeTransform interface & decorating it with @Pipe({})



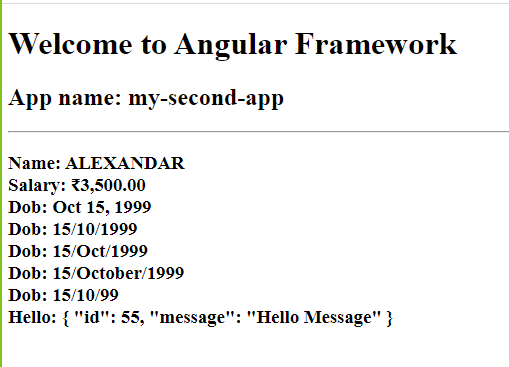
pipe-demo.component.ts



pipe-demo.component.html



Output:

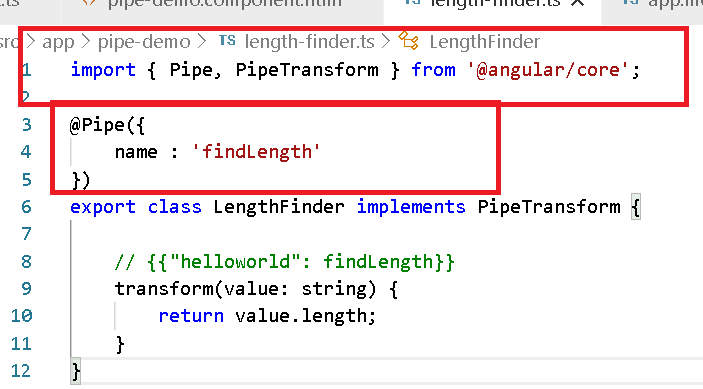


You can create custom pipe by using *ng g p pipe-name*

The above command creates a class that implements PipeTransform & decorate it with @Pipe({})

This will update the app.module.ts

We can also create pipe without using command as below



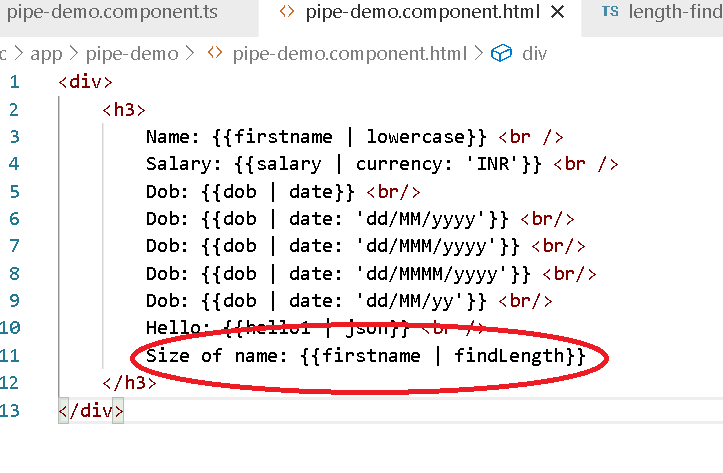
Since we created custom pipe manually we must update it in app.module.ts -> declarations property.

app.module.ts

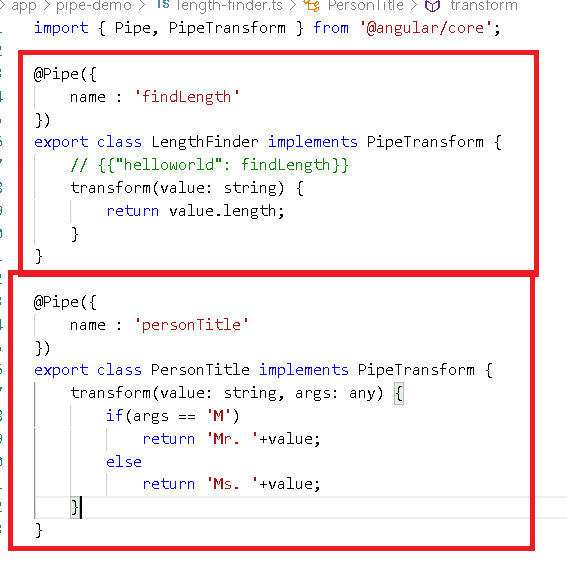


Now you can use this pipe name called ‘findLength’ in any html template

pipe-demo.component.html

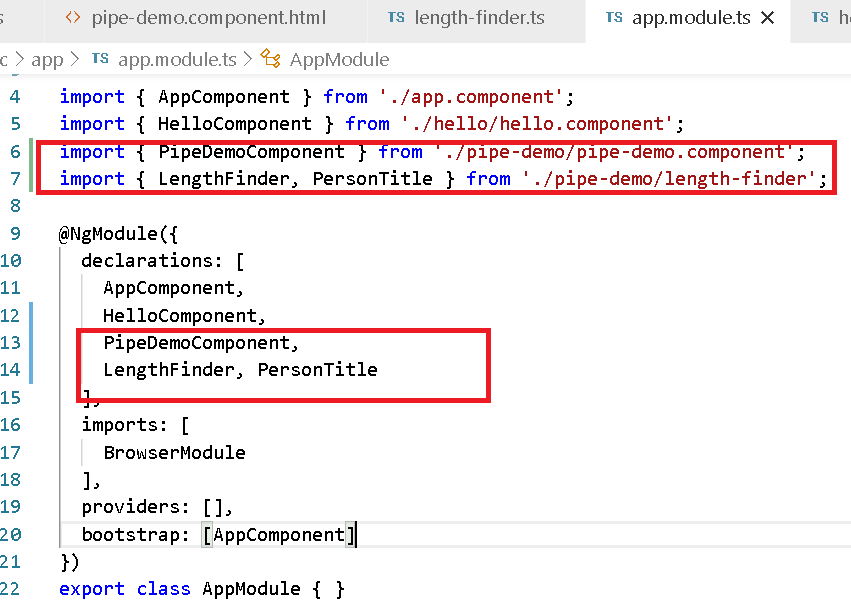


You can create any number of pipes in the same file but you need to have @Pipe for each class & mention them in the app.module.ts

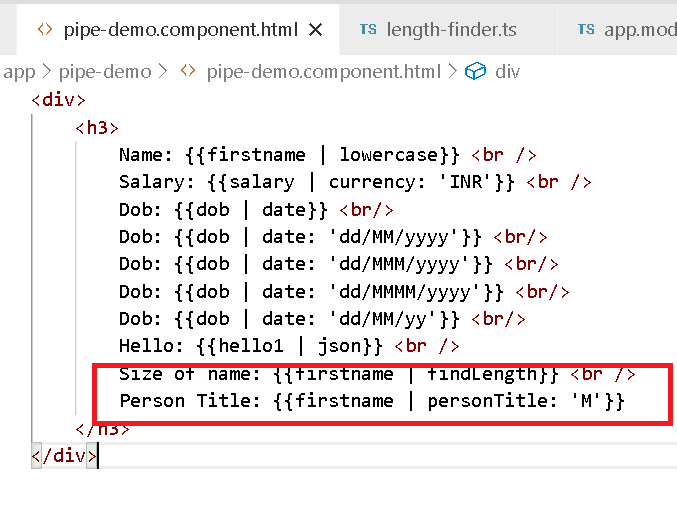


Note: transform(value, args), means {{value | pipeName : args}}.

app.module.ts



pipe-demo.component.html



Note: Here ‘M’ is passed in html directly which is not proper way of mentioning the firstname is a male, so we must render Mr. or Ms. based on the data coming from component

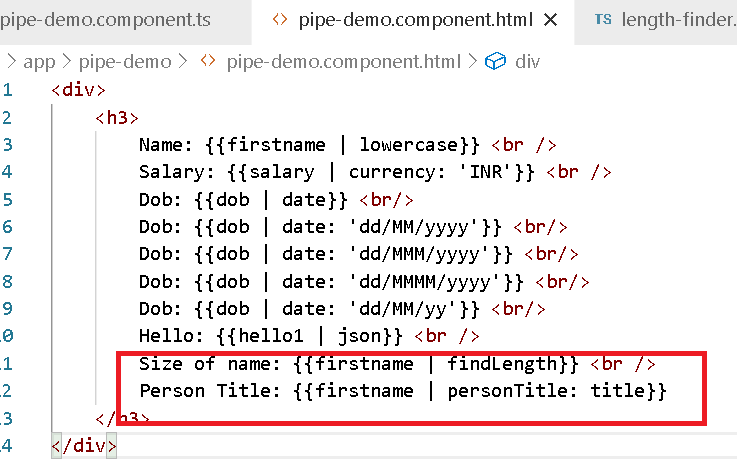
Output:



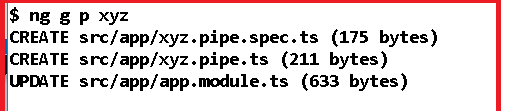
pipe-demo.component.ts



pipe-demo.component.html



Note: You can also generate the pipes using ng g p pipe-name



The above command gives you all the necessity to use pipes and directly you can implement transform().

Exercise:

Create a StudentComponent that will have Student data like rollno, name, marks.

* create 3 students with different marks like 30, 50, 80
* show all the 3 student data with marks, but use a pipe called result that will calculate the marks and displays pass, fail and firstclass

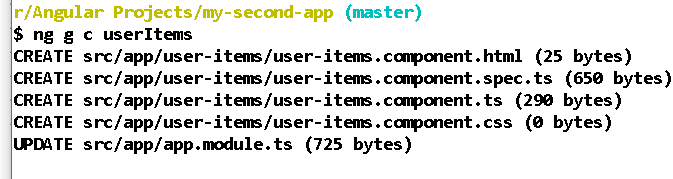
Structural Directives:

* \*ngFor
* \*ngIf

These directives is used to modify the DOM structures

\*ngFor: It is used to iterate over the arrays and display items

\*ngIf: It is used to apply conditions to display the content

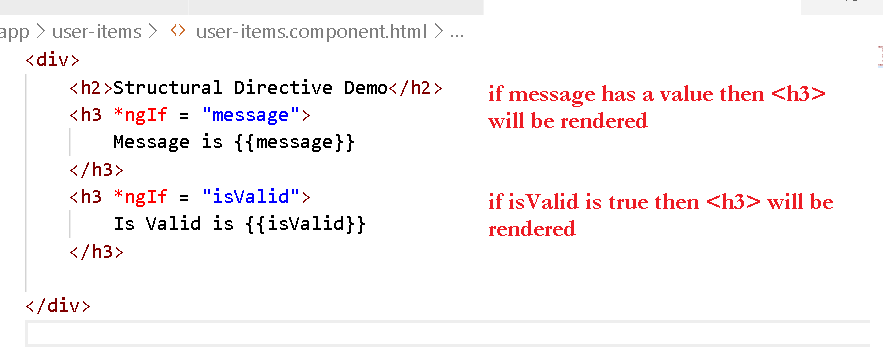


NgIf Demo

user-items.component.ts



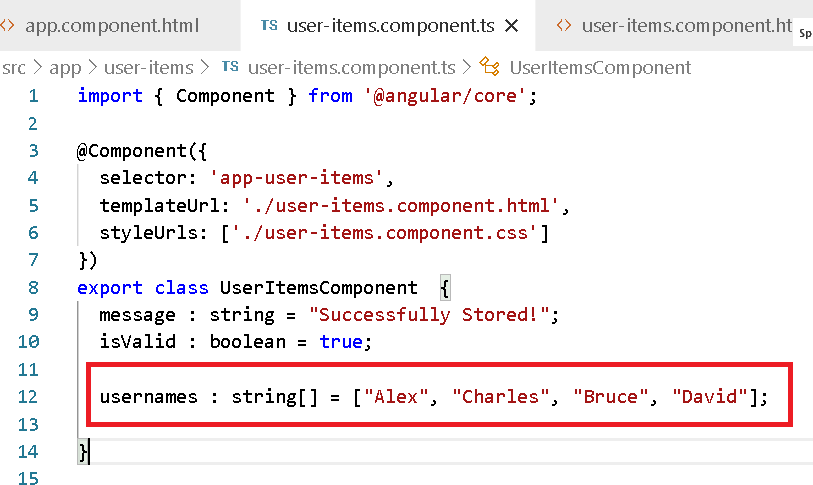
user-items.component.html



Output:

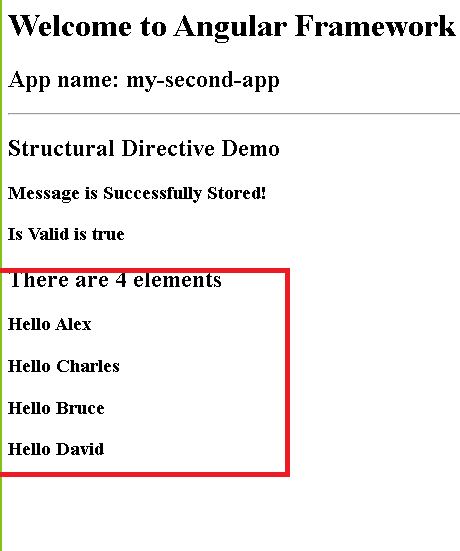


NgFor: This is used to iterate over the array and dynamically structure the DOM.



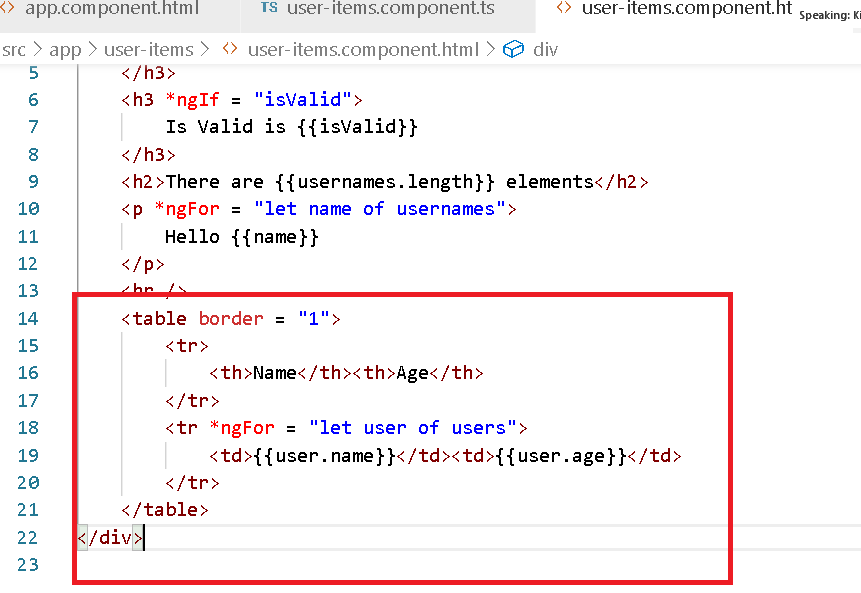


Output:

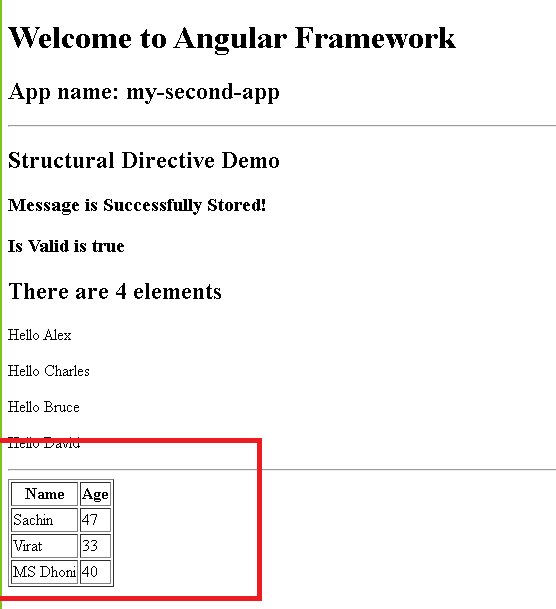


With NgFor you can create table rows dynamically



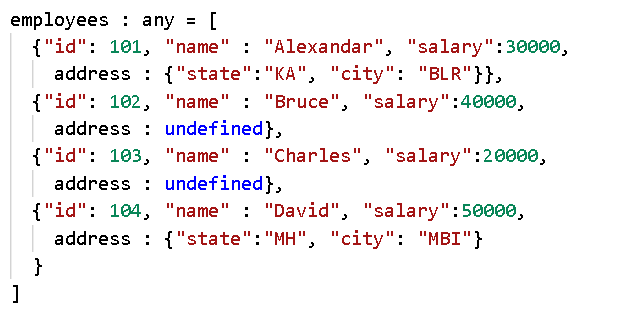


Output:



Exercise:

Try to display the below data in a table



You have to have a table with totally 5 columms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Name | Salary | State | City |
| 101 | Alexandar | 30000 | KA | BLR |
| 102 | Bruce | 40000 |  |  |
| 103 | Charles | 20000 |  |  |
| 104 | David | 50000 | MH | MBI |

Databinding:

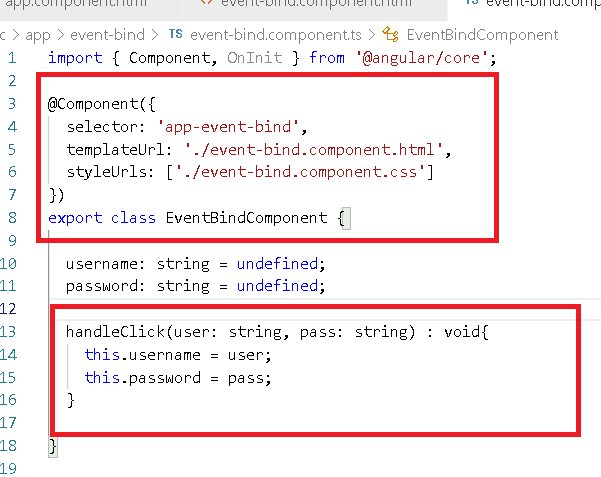
It binds the data between the view & the component to enable the data to flow between them.

We have four types data binding to make the data to flow between the view & the component

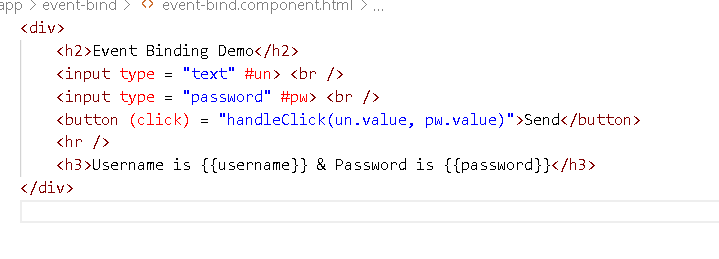
1. Interpolation: Component -> View {{}}
2. Event Binding: View -> Component (eventName)
3. Property Binding: Component -> View [property]
4. Two way databinding: Component <-> View [(ngModel)] -> ngModel by default wouldn’t be available it is part of FormsModule that must be mentioned in your AppModule imports section

Event Binding

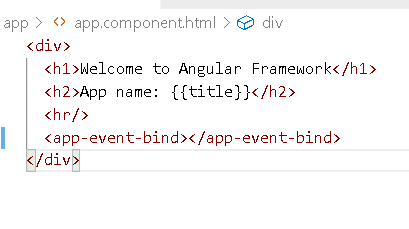
  
event-bind.component.ts



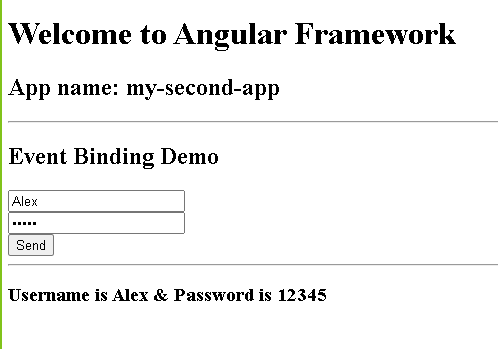
event-binding.component.html



app.component.html



Output:

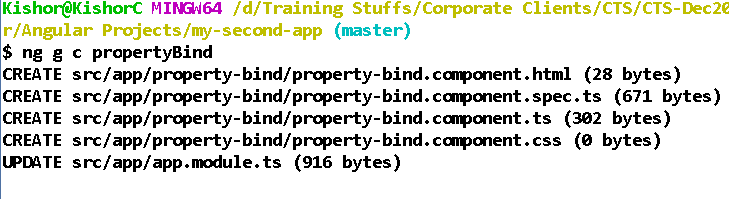


There are different types of events you can handle

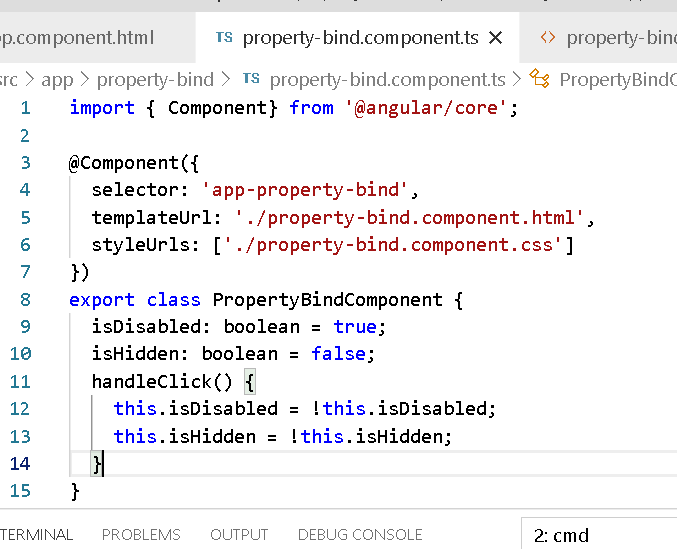
1. (click): when you click on an element
2. (input): when you enter input to the input element
3. (ngSubmit): when you submit the form
4. (mouseEnter) & (mouseLeave): when you place cursor over the element
5. (keyUp) & (keyDown): when key is pressed/released

Property Binding: [propertyName]: C -> V

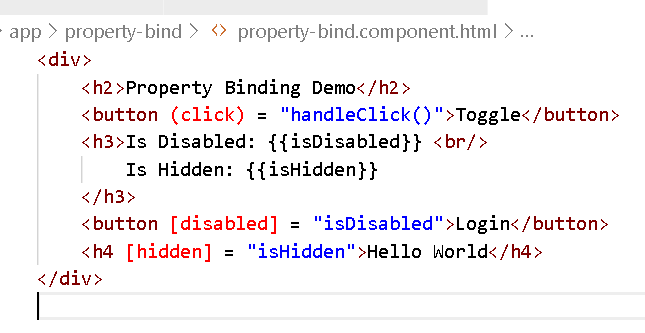
This is related to DOM properties, where you can hide/show the dom element or enable or disable the dome elements, here the data flows from component to view.



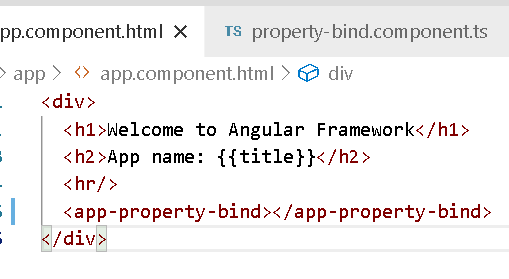
property-bind.component.ts



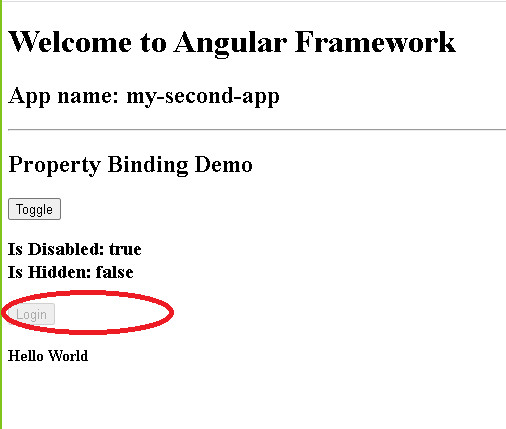
property-bind.component.html



app.component.html



Output:

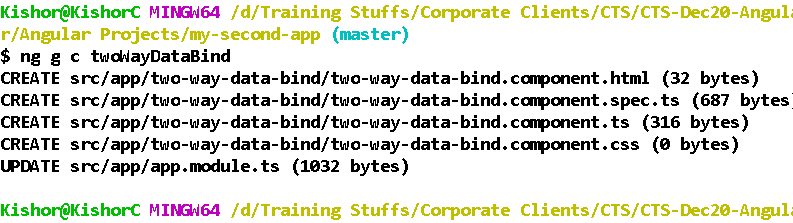


Two way databinding [(ngModel)]:

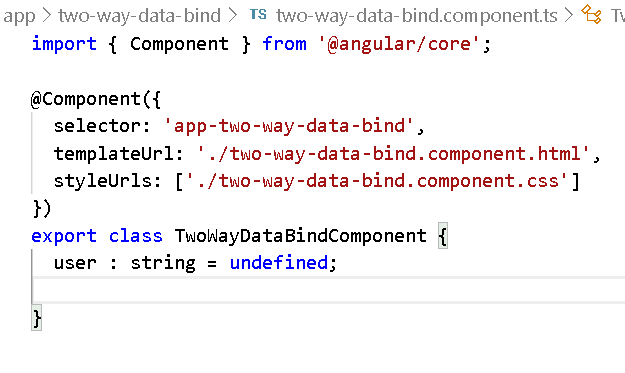
Here the data flow happens in both the directions i.e., from view to component & vice versa.

Here it automatically binds the data without explicitly binding each variable with event binding

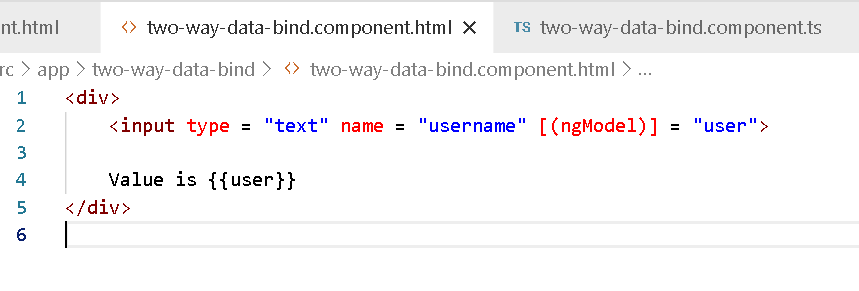
By default ngModel is not added in your application hence you must use FormsModule in your application (AppModule) to use the ngModel



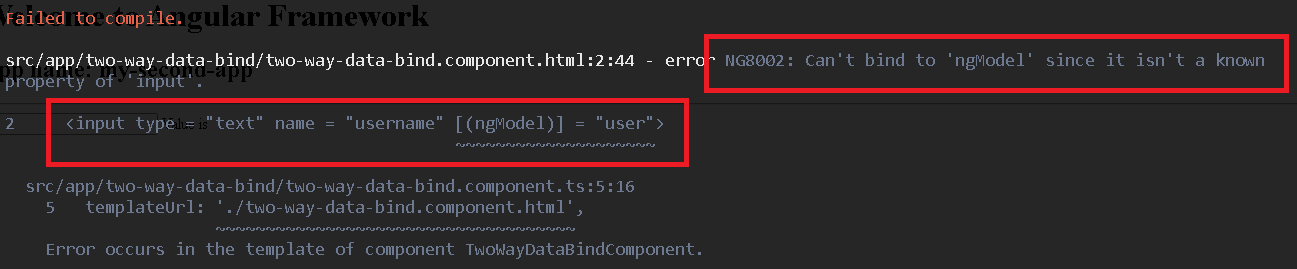
two-way-data-bind.component.ts



two-way-data-bind.component.html

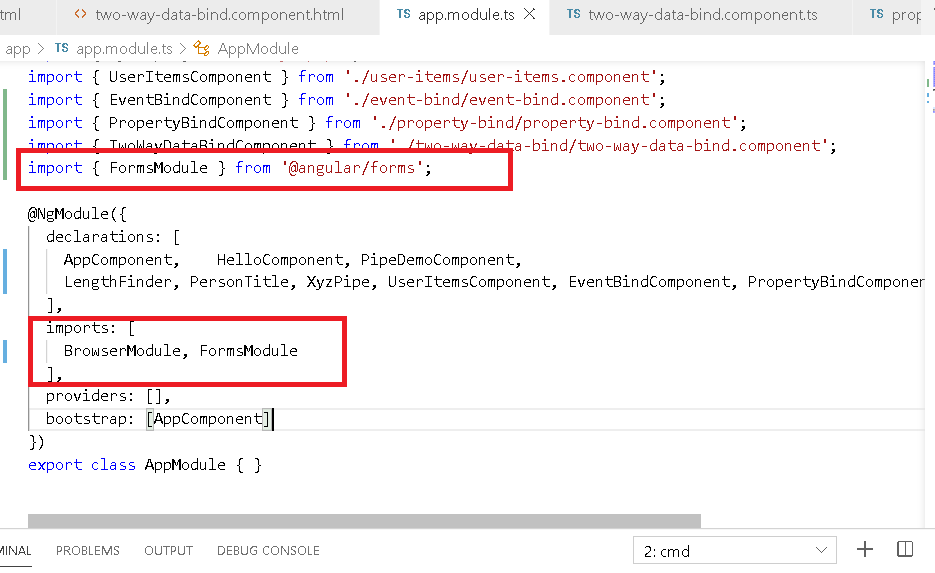


You will get below error when use [(ngModel)]

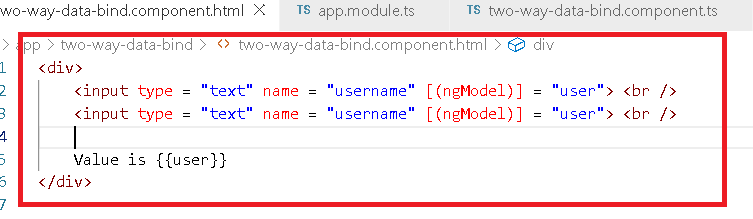


You must add *FormsModule* in imports of @NgModule({})

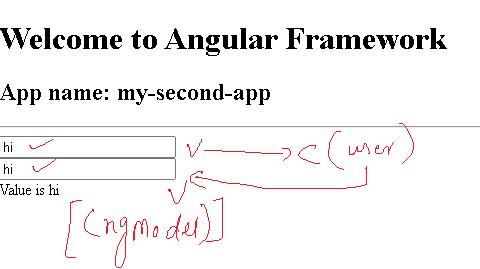
app.module.ts



two-way-data-bind.component.html



Output:



Advantage of using two-way-databinding is

1. You can directly update the property of the component from view so that you don’t need to have event bound to each data of view to the component through function.

i.e., <input (input) = “fun1(value)”>   
 <input (input) = “fun2(value)”>

2. You use two way databinding in case of form handling and also it helps in updating the view automatically when some data in components are updated the dependent property will be auto-updated because in two way databinding the data flows in both direction.

3. Form validations can be done where when use sends the data to the component the same data can be validated in the view

All these types of databinding makes data to flow either from view to component or component to view, but there are two more types of databiding where data can flow between the components.

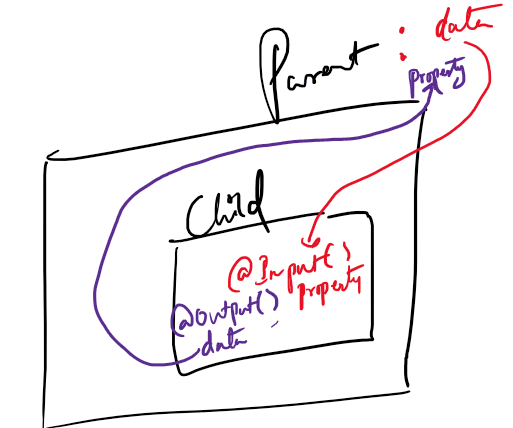
Component interacting/communication

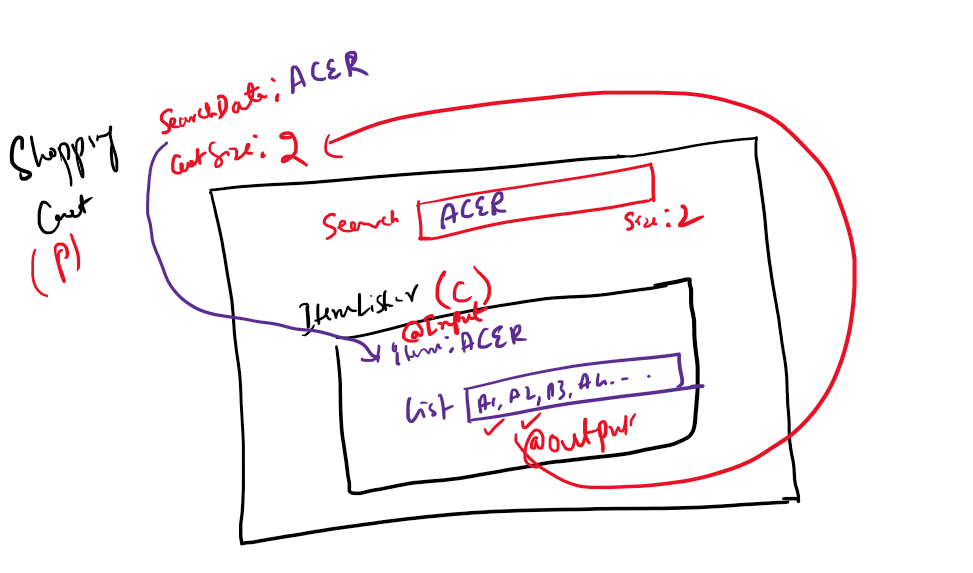
Here you can pass the data from one component to another component, we have two types of component here

1. parent component: enclosing component of another component
2. child component: a component nested inside another component.

You can use two types of decorators here

1. @Input(): Where data flows from parent to child
2. @Output(): Where data flows from child to parent



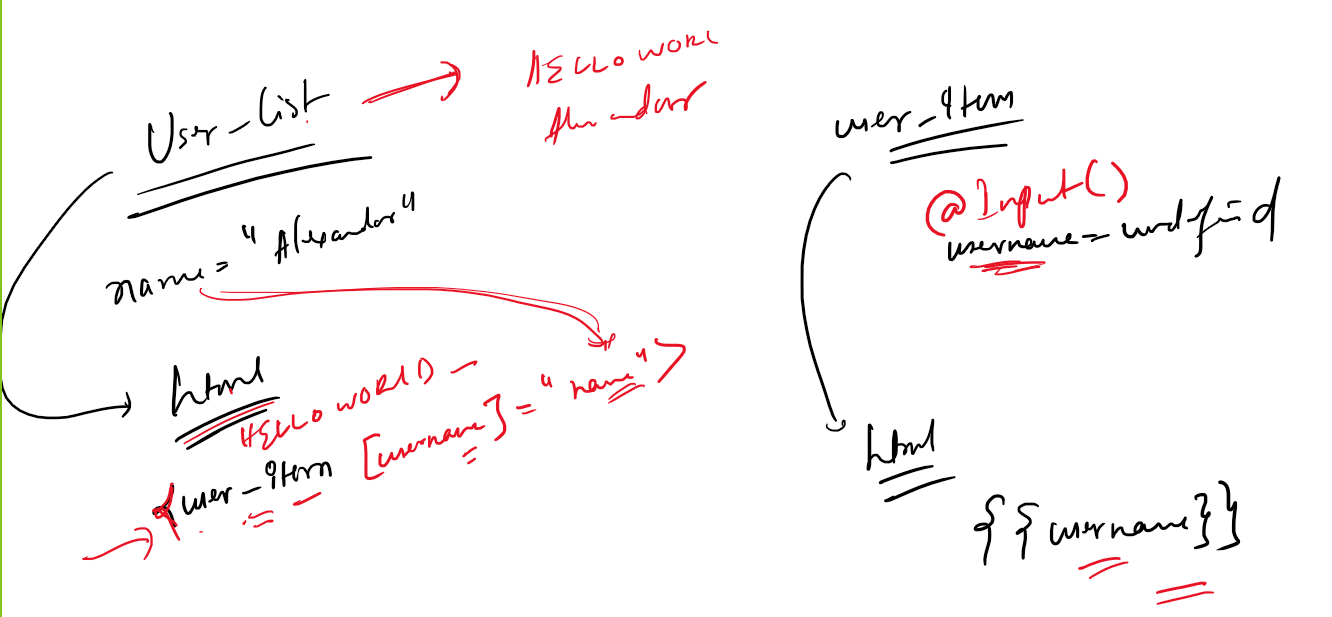


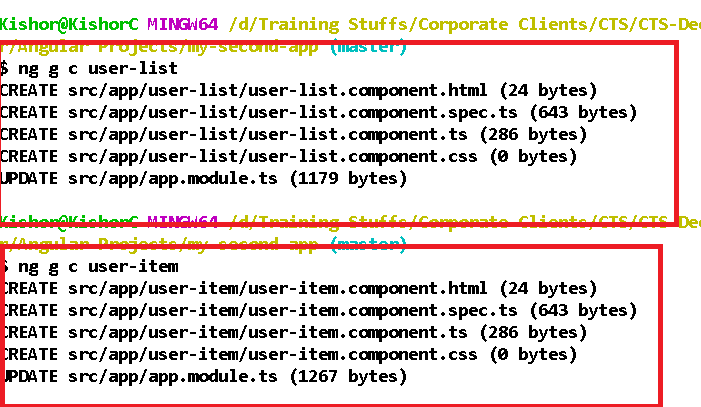
In a shopping care search box will be in parent component that receives the item to search and parent component will also have size to show the number of items selected, another component ItemLister will be a child that receives the value from parent to search and lists those items, when you select the item that will be updated to the size of the cart.

@Input() Demo

Here we will create a parent component called user-list(P) and a child component user-item(C) which displays the user details passed from user-list.

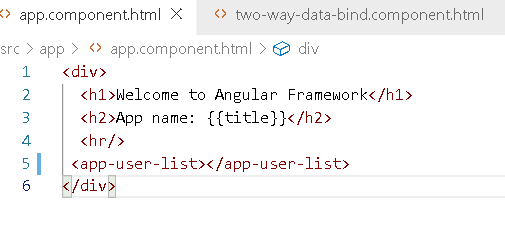
Note: child components are nested inside parent component.





Add user list to the root component

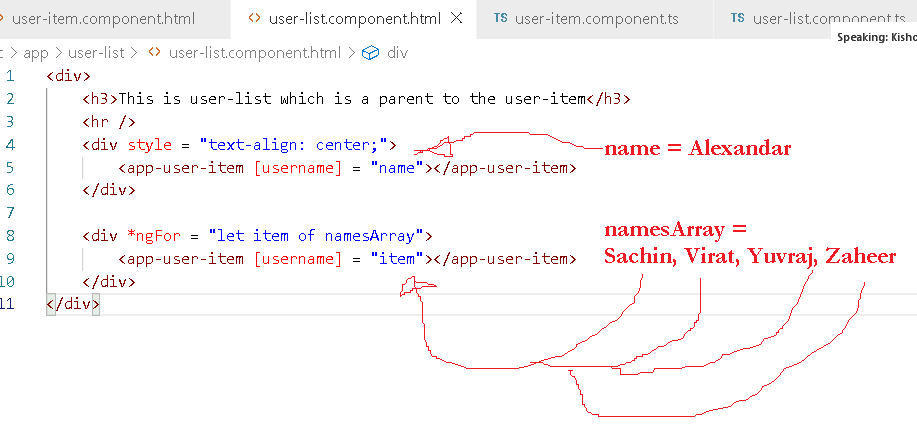
app.component.html



user-list.component.ts



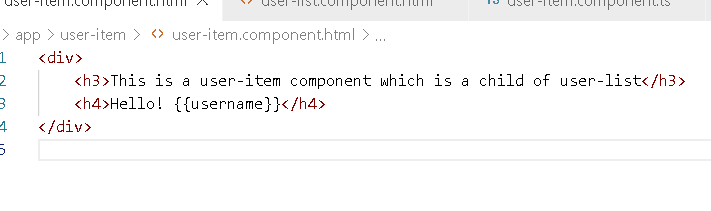
user-list.component.html



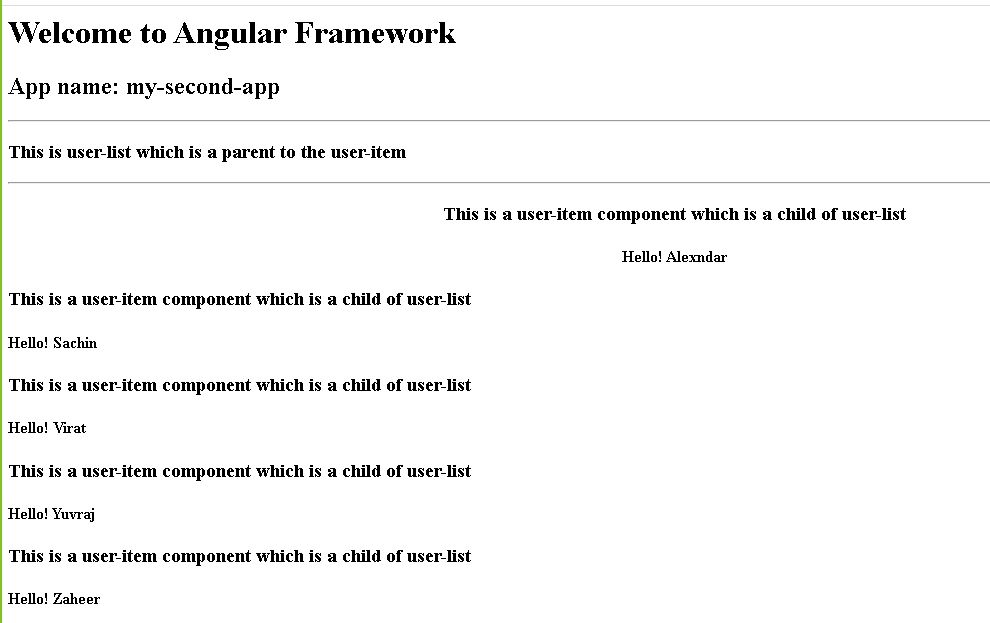
user-item.component.ts



user-item.component.html



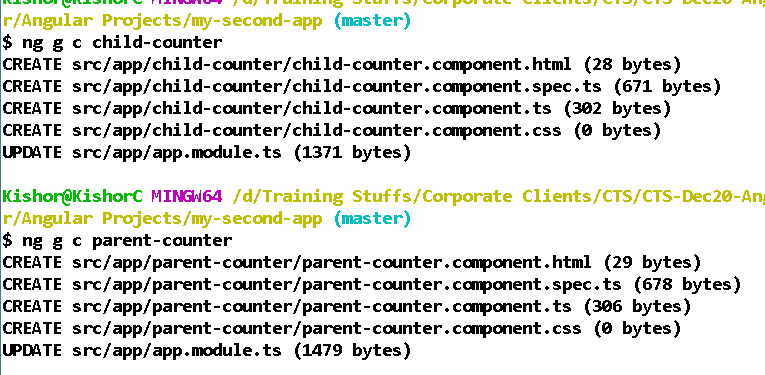
Output:



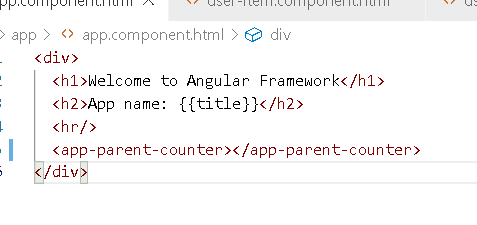
Note: [property]: it is used for DOM as well as component property binding, in DOM we use DOM properties like disabled, hidden, innerHTML and so on. Whereas in component property binding we use property of the child component, i.e., username if user-item

@Output():

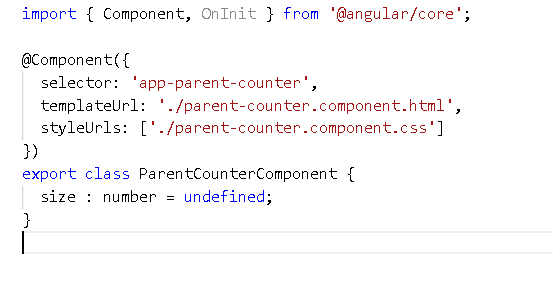
It is used to pass child component values to the parent component, this is done as a event binding where you bind property of child component that is of type EventEmitter



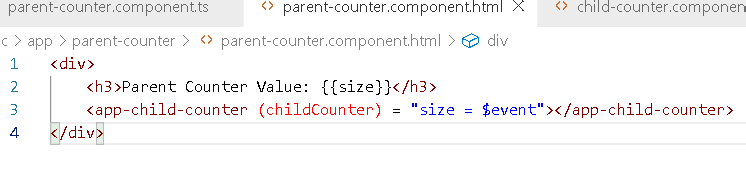
Add parent counter component to the root component



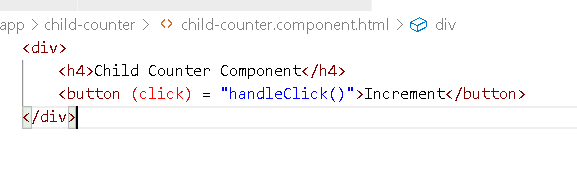
parent-counter.component.ts



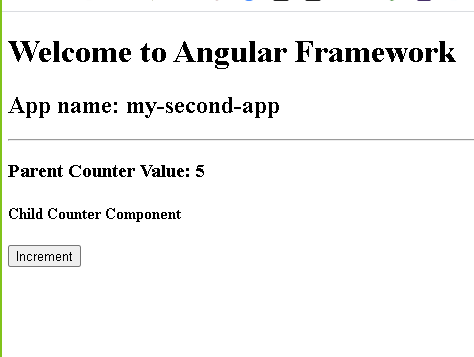
parent-counter.component.html

child-counter.component.ts

child-counter.component.html



Output:



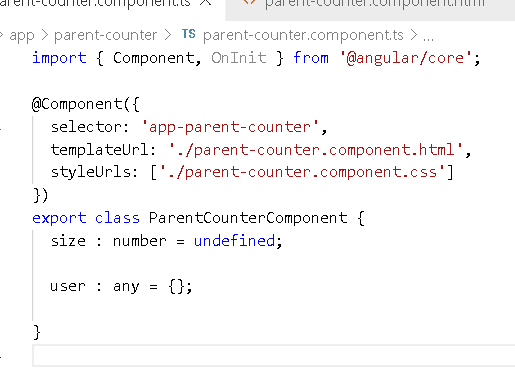
Note: @Output is mentioned on EventEmitter variable, that generates an event, $event is an event object that will have the value emitted by the event emitter property, so you must use child component event emitter property like a event binding.

<child (eventEmitterProperty) = “parentProperty = $event”>

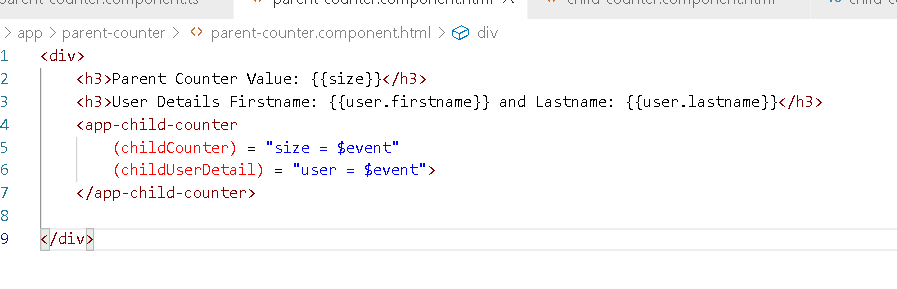
Note: You can emit any complex object instead of number, string, boolean.

Let us try to emit a complex data from child and pass to parent

parent.component.ts



parent.component.html



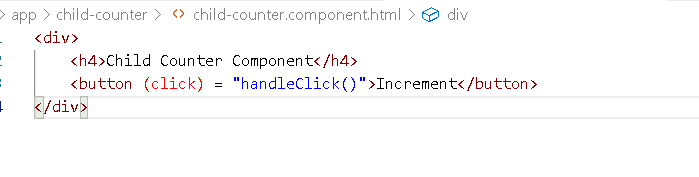
Note: childUserDetails generates user object & childCounter generates number

child-counter.component.ts

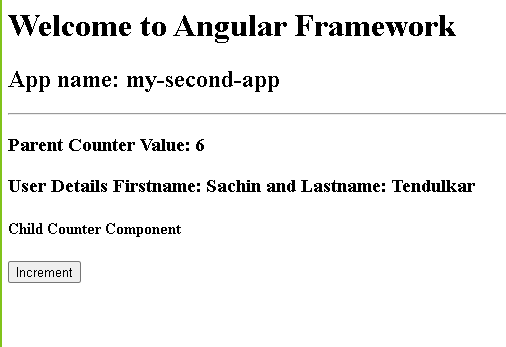


In the above code same handleClick generates event through childCounter & childUserDetails, event generated from childCounter is assigned to size & childUserDetails is assigned to user property of parent-counter

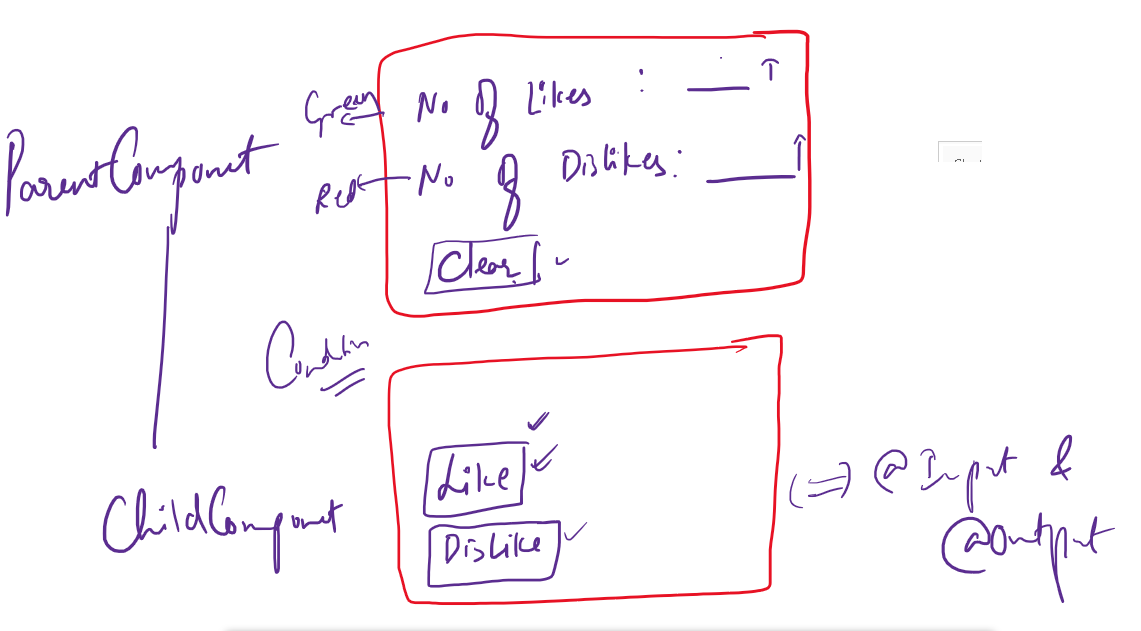
child-counter.component.html



Output:



Exercise:

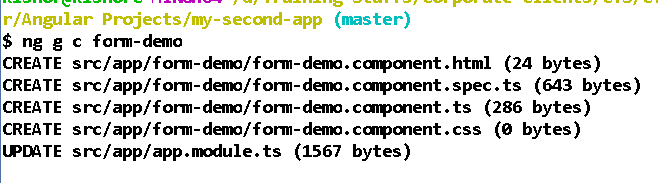


Condition: After clicking clear the like and dislike should start from the beginning

Angular Forms:

It uses ngForm to handle the data entered in HTML and submit to the components.

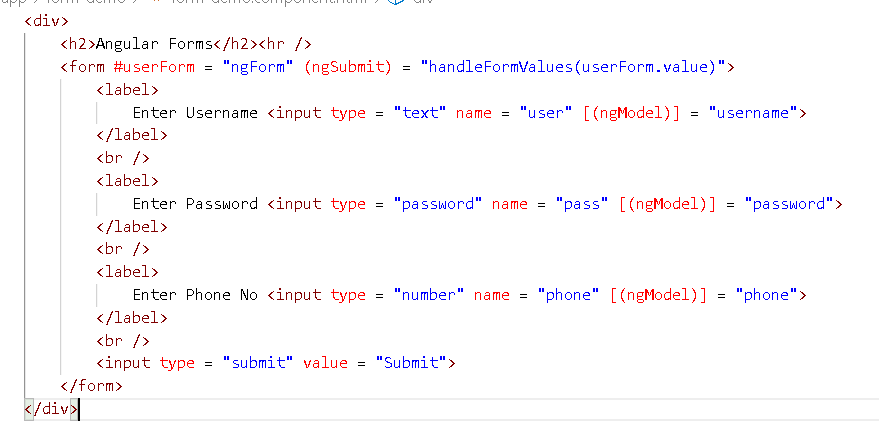
ngForm is an angular directive part of FormsModule this can hold all the form data in a single name



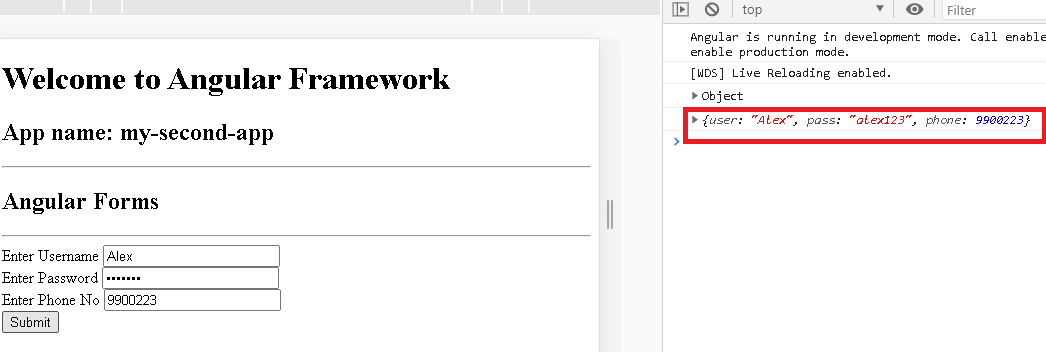
form-demo.component.ts



form-demo.component.html



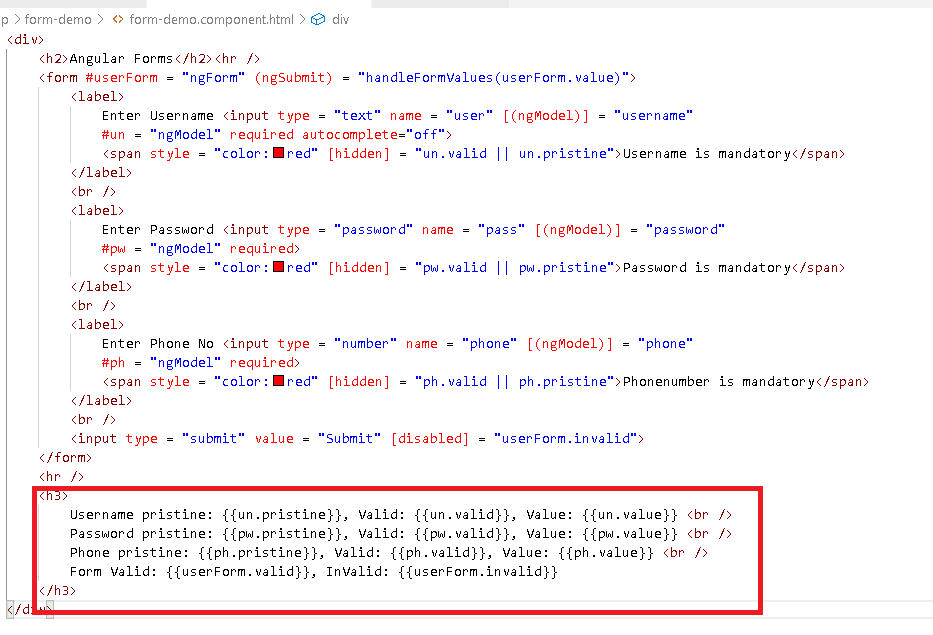
Output:



Form Validation:

ngForm gives some inbuilt properties where you can extract the form values, validate the form inputs with some properties.

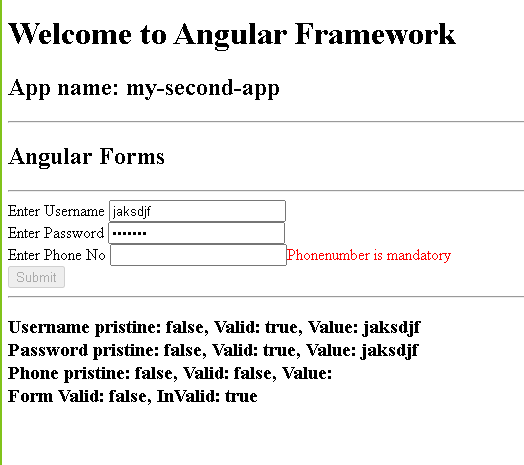
form-demo.component.html



Pristine will be true if control is not modified

No changes in the form-demo.component.ts

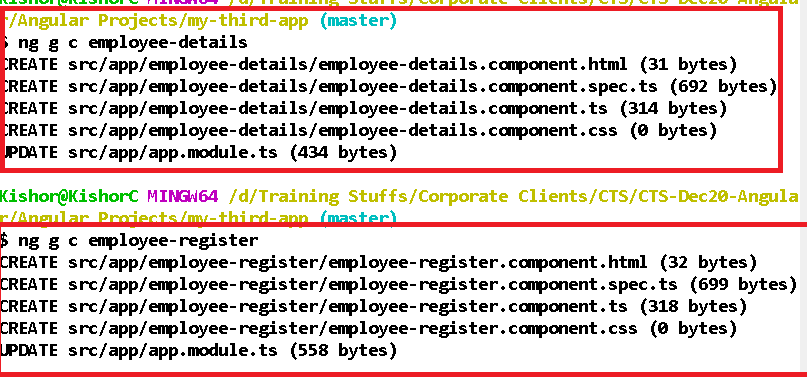
Output:



Services in Angular

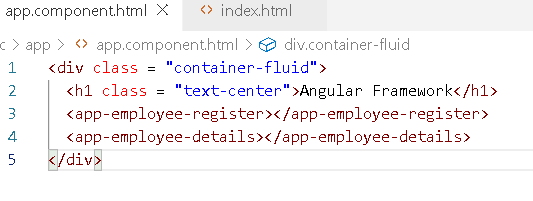
These are classes that will have a reusable code which usually access remote services (REST webservices) to access the backend data, these are reusable across multiple components, i.e., components will access logics present in services.

* Services are created with @Injectable() decorator
* @Injectable() decorator enables the service instance to be auto-created by angular & supply to the components via constructor parameter, you don’t have to create service instance

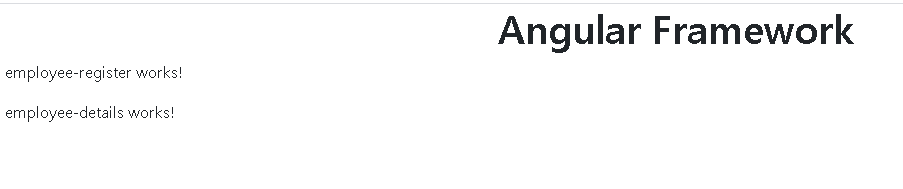


We can show both the component in the root component

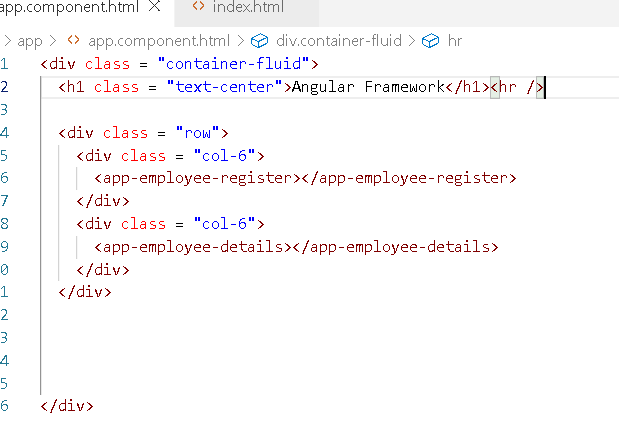
app.component.html



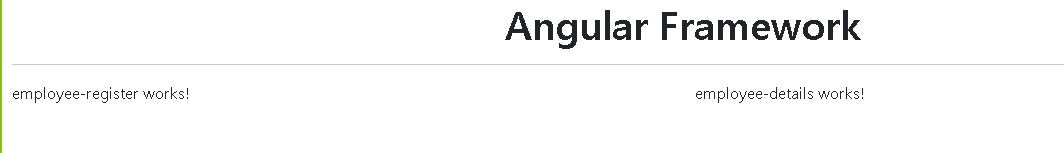
Output:



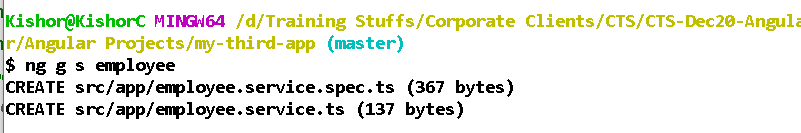
We can use bootstrap row & col class to keep both the components in the same row



Output:



Now we will create a service that can be supplied to both the components



employee.service.ts



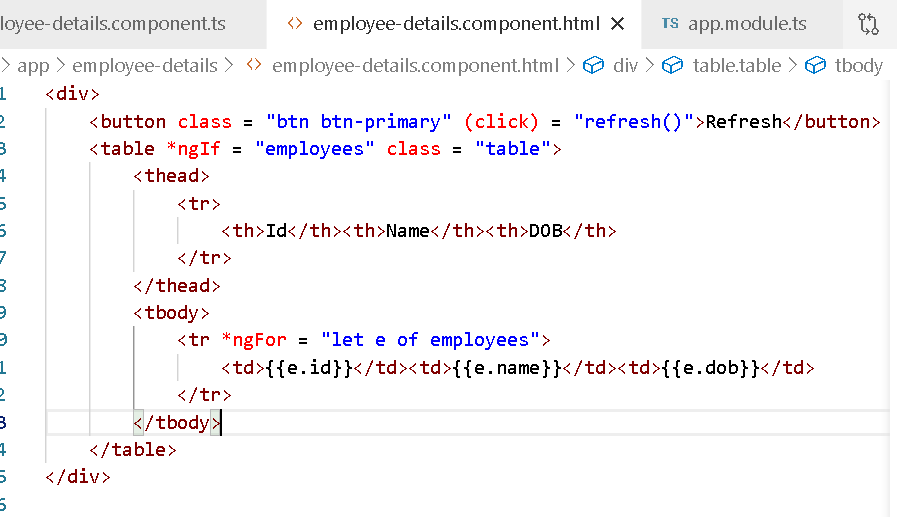
Now multiple components can use this service instance via constructor parameter.

employee-details.component.ts

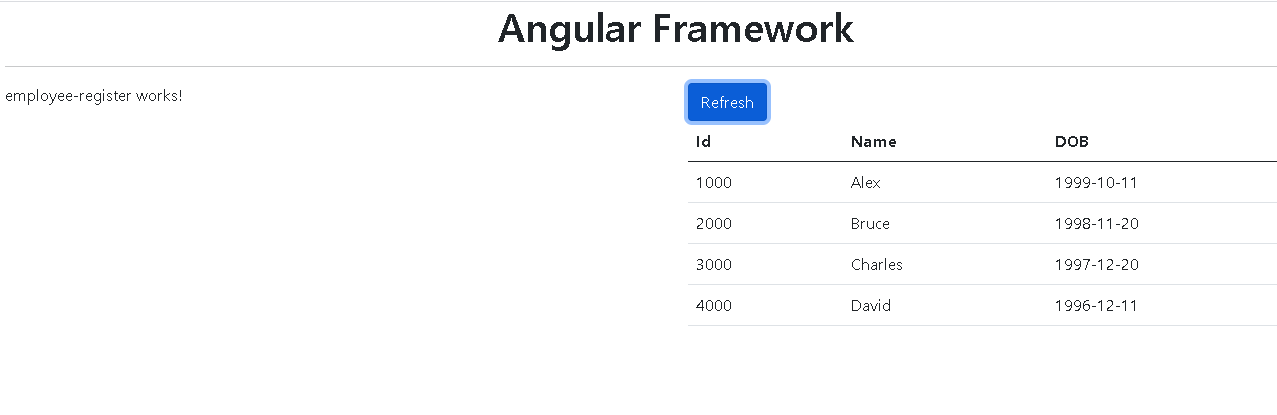


Here refresh() function is called on button click so that you can see employee details.

employee-details.component.html

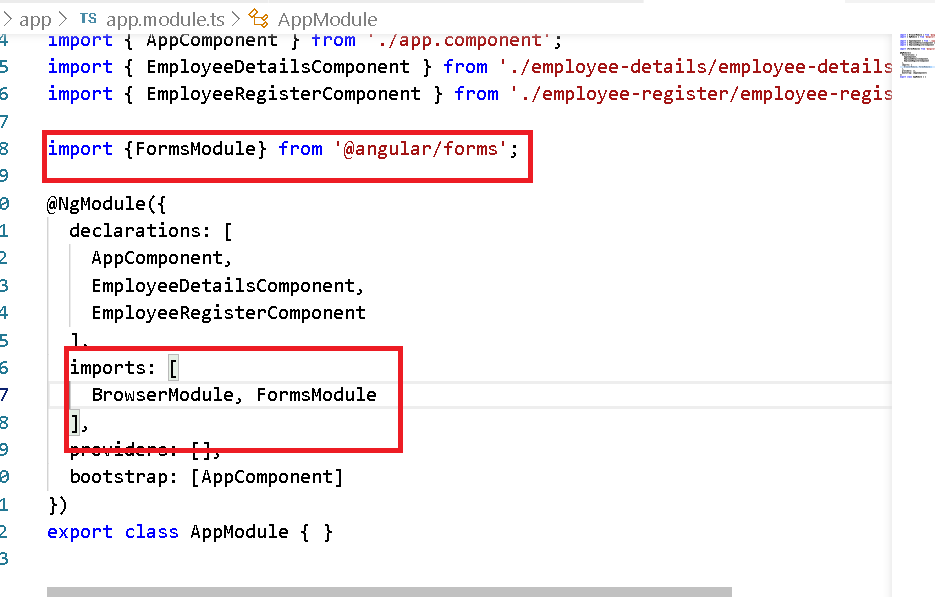


Output:



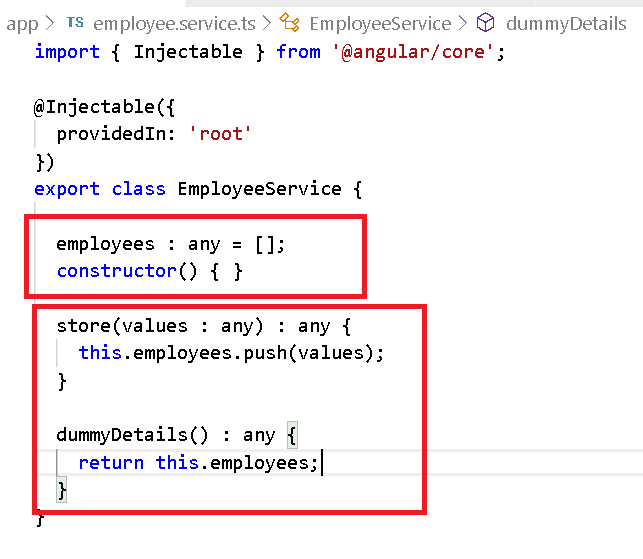
In Employee Register you can create a simple form

app.module.ts

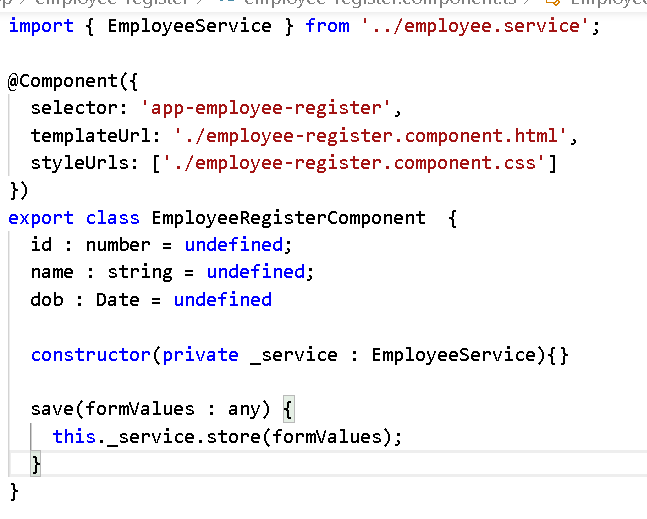


Change the service to access the employee details from form & return the stored employee

employee.service.ts



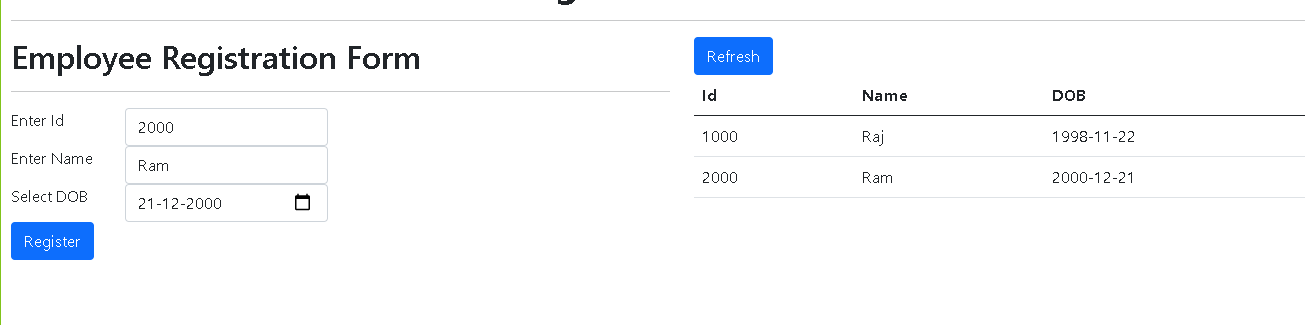
employee-register.component.ts



employee-register.component.html



Output:



HttpClient: It is an instance which is used to call the backend REST API’s, since REST API’s use different HTTP methods like:  
GET, POST, PUT, DELETE, the HttpClient also provides these methods where you can use the REST API’s uri and call them to get the data or send the data.

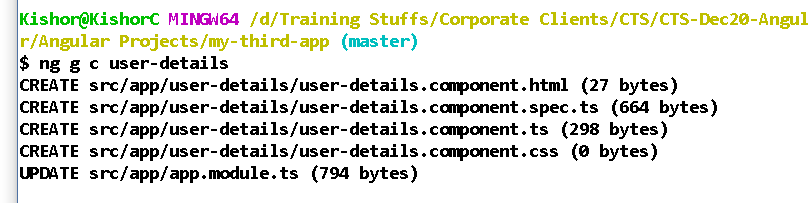
HttpClient is automatically instantiated by angular, with the help of HttpClientModule which has to be mentioned in the imports of app.module

app.module.ts

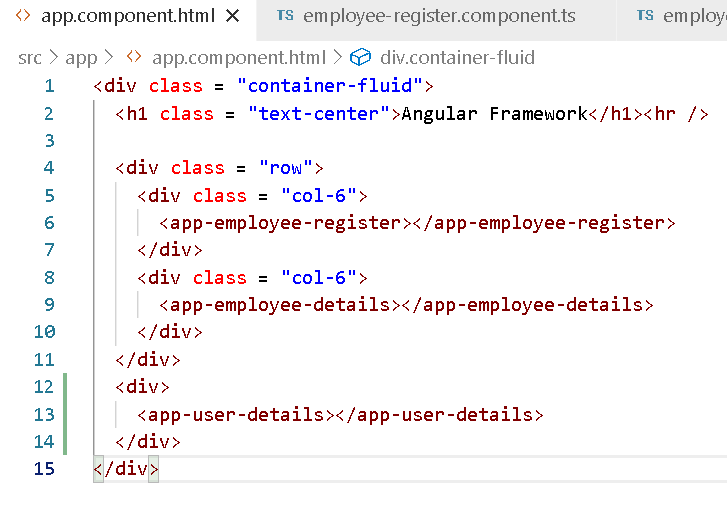


The HttpClientModule will help in supplying the HttpClient instance to your service via constructor parameter.

We are going to use Fake REST API’s to get the data.



Add the user-details to the root component



Use the HttpClient in the Angular service

employee.service.ts



HttpClient methods returns Observable which keeps the response from the Online API.

Now the components can call these methods returning the Observable and extract the response content using subscribe(..) method.

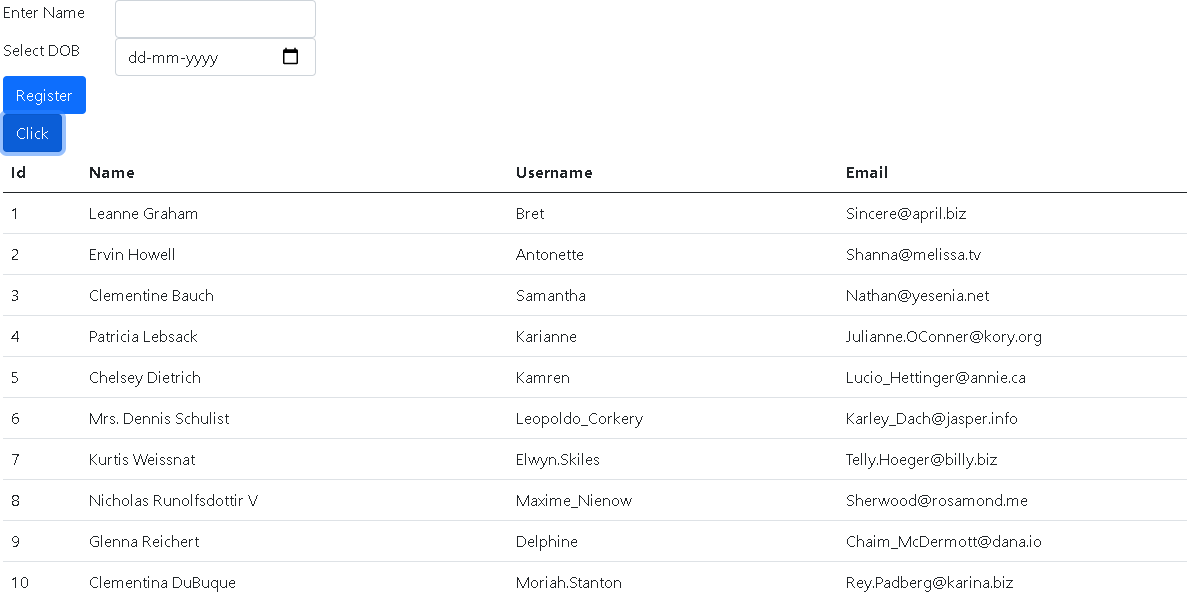
user-details.component.ts



user-details.component.html



Output:



HttpClient has methods which returns Observable<T>.

Observable<T>: It maintains the response from back-end service, we need to call subscribe(arrowFunction) to extract the response data.

Making get request

http.get(url)

Making post request

http.post(url, javascriptObject)

Making put request

http.put(url, javascriptObject)

Making delete request

http.delete(url, javascriptObject)

Angular Router:

Router is used to navigate from one component to another, you can use Router by adding RouterModule in the AppModule imports.

RouterModule is part of @angular/router, it knows the URL and Component for the URL that has to be loaded in the routing area.

In Angular Router you use the following elements:

<router-outlet>: It is a reserved area to load the components based on the uri.

Routes: It configures the path & the component

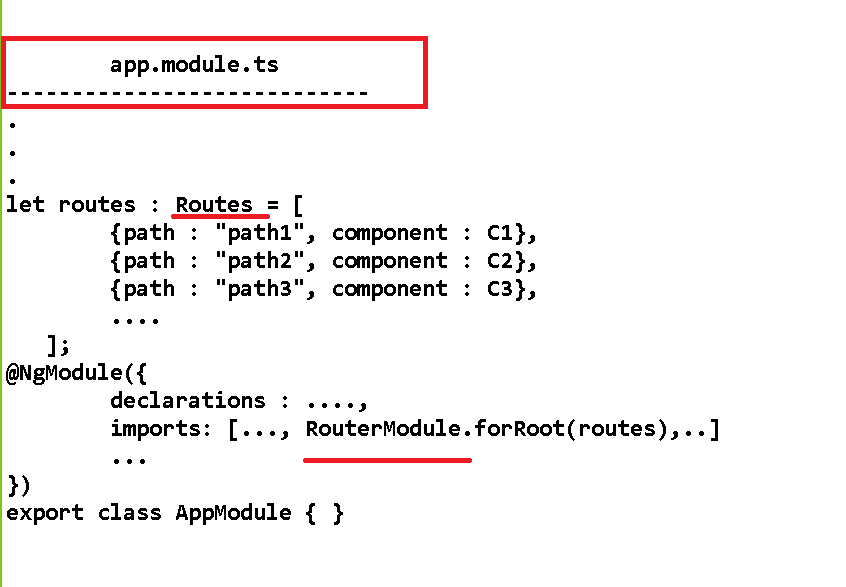
RouterModule: It enables your application to use the angular routers, this takes care of loading the component to the <router-outlet> based on the Routes configuration

routerLink: It provides the path in the HTML template

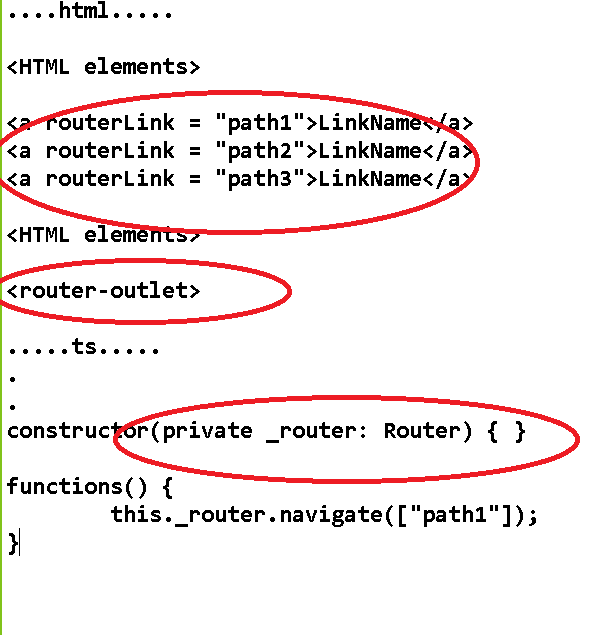
Router: It provides the path in the component class

ActivatedRoute: It helps in extracting the data from source to destination component, it has to be used in the destination component

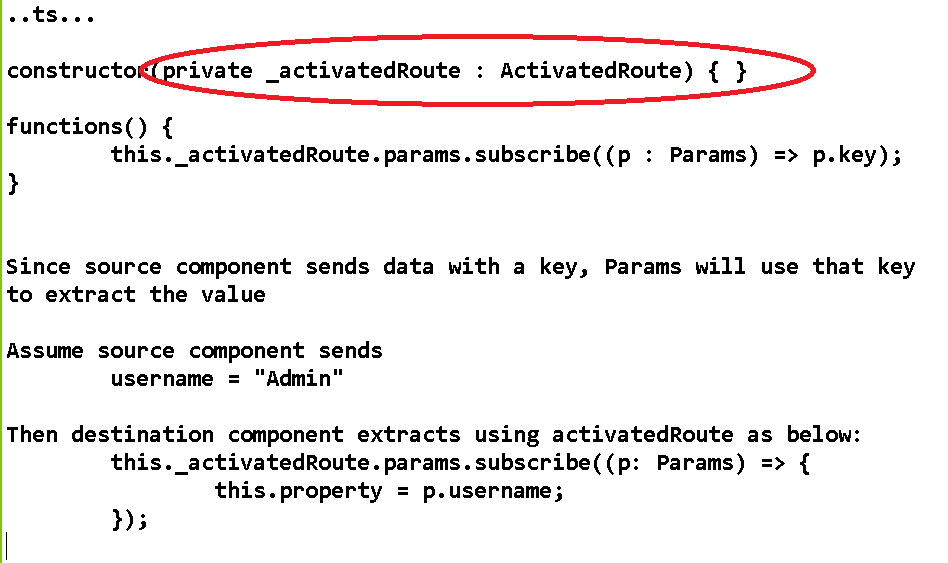
How you will add routes & router module in the app.module.ts



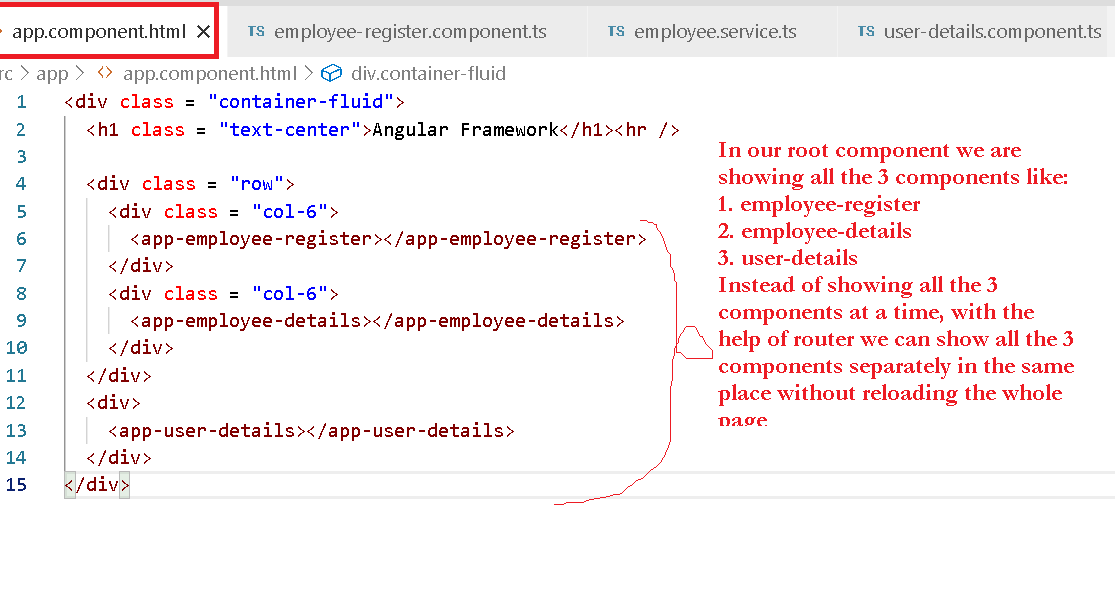
How to provide paths to the RouterModule



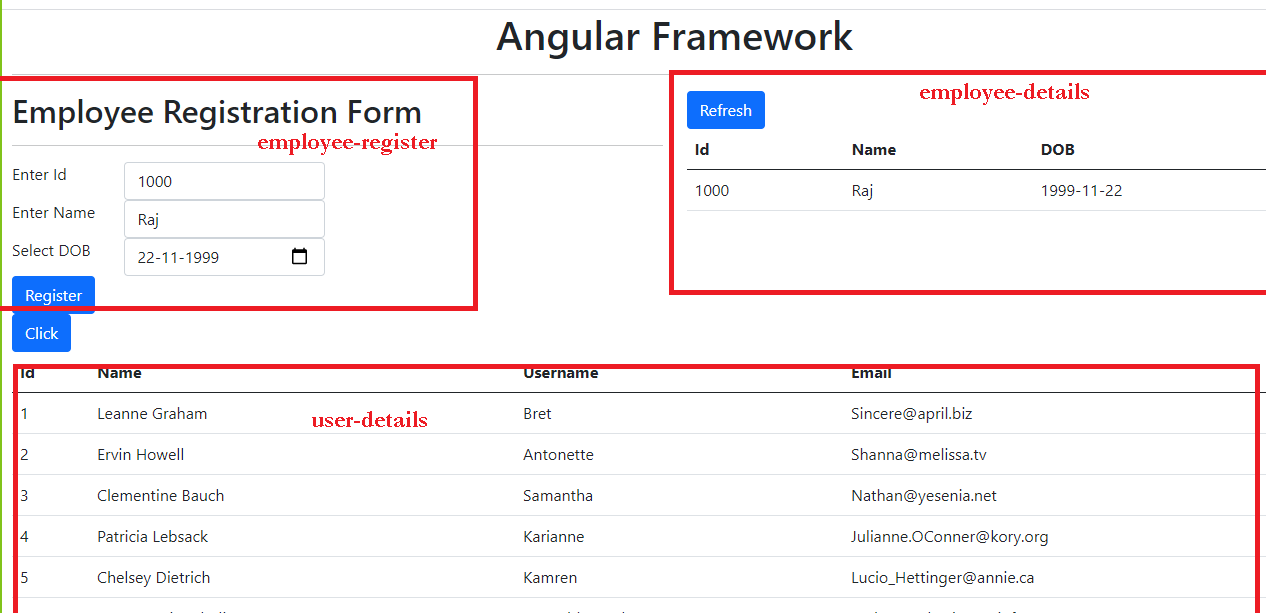
How to use ActivatedRoute to extract the data from source component



We have all the 3 components displayed in the app.component.html which is looking as below:



Output:



Now we must create links & show only one component at a time in the same root component with the help of router

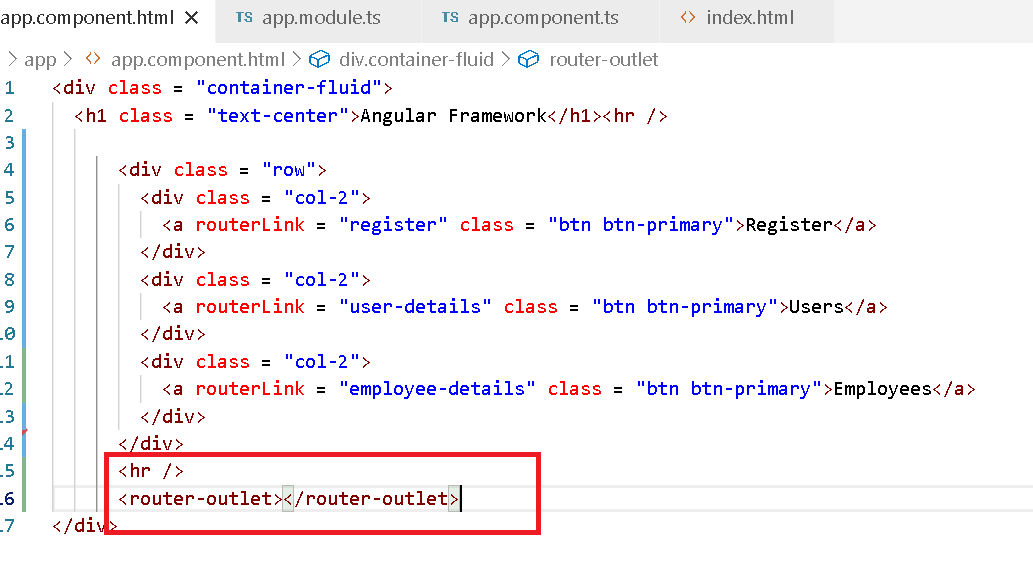
Steps:

1. Configure Routes in app.module.ts
2. Add RouterModule in the imports in app.module.ts
3. Configure routerLink: app.component.html
4. Configure router-outlet: app.component.html

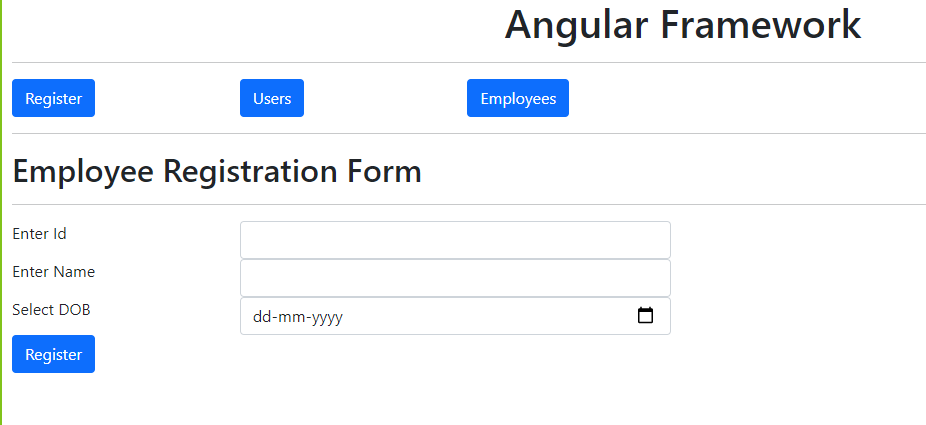
app.module.ts



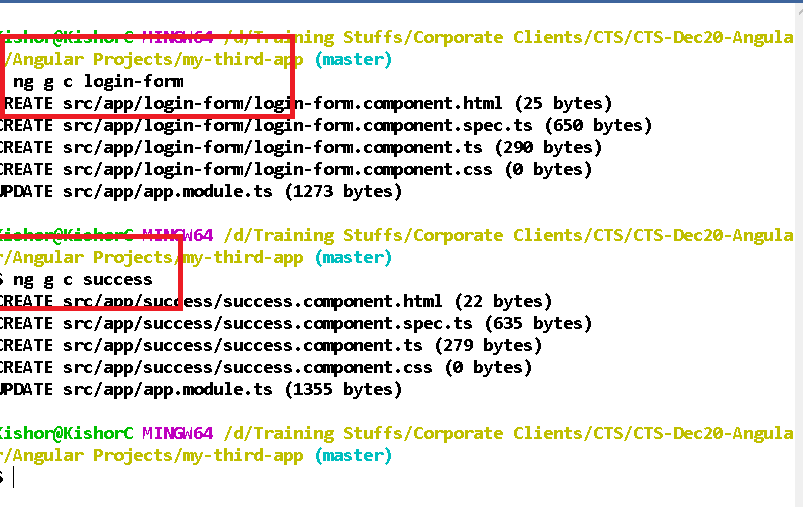
Remove all the components from app.componen.html & add routerLink & router-outlet



Output:

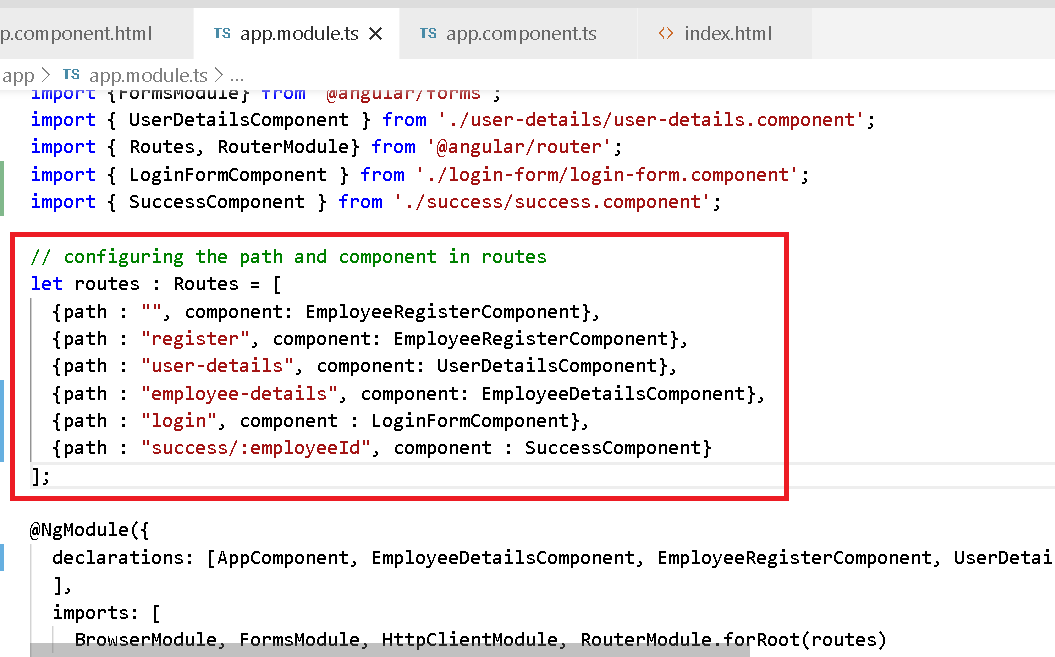


Using Router & Activated Route



We will have some dummy credentials in the angular service login method & later we will send the credentials to Spring REST (mvc/boot)

Modifying the app.module.ts to have paths to Login & Success



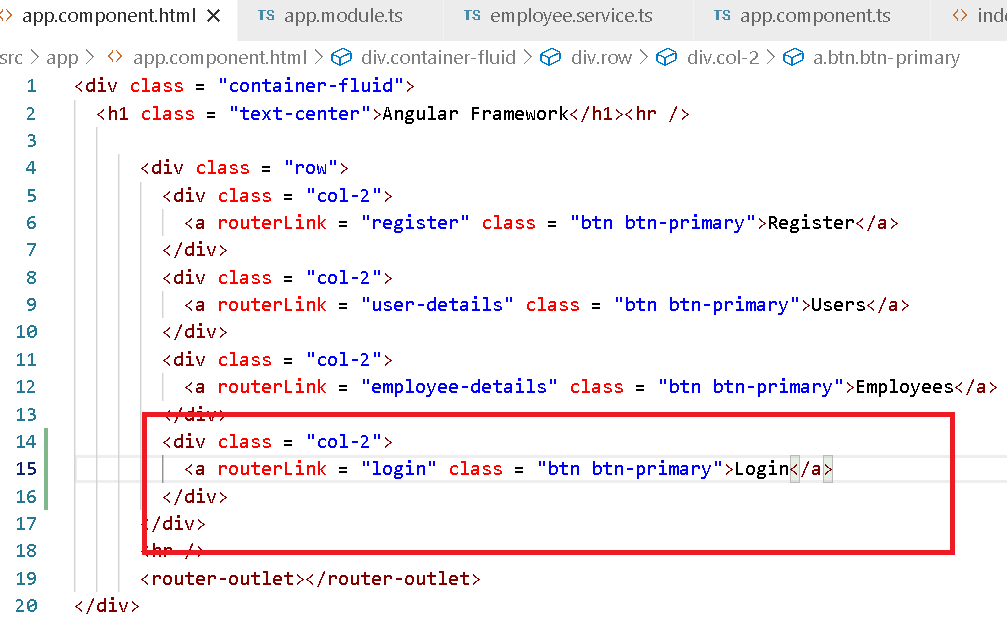
/login -> LoginFormComponent

/success/1000 -> SuccessComponent -> extract employeeId (1000)

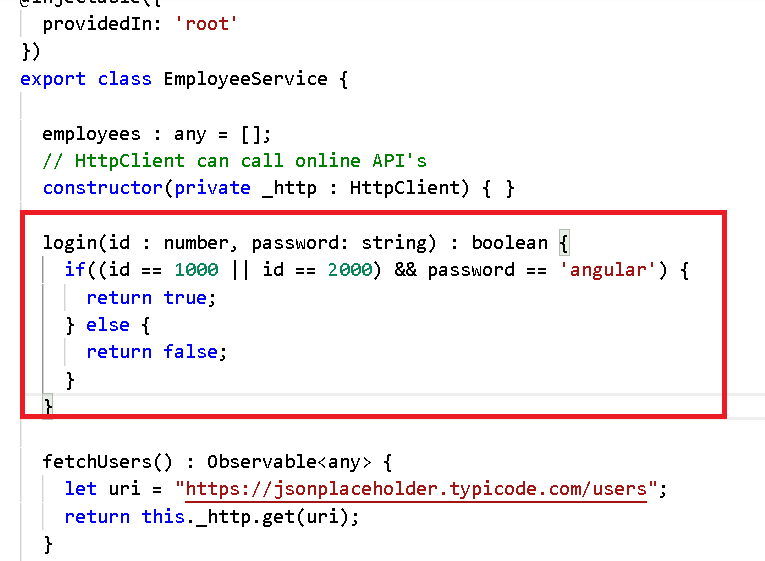
/success/2000 -> SuccessComponent -> extract employeeId (2000)

Now the SuccessComponent can use ActivatedRoute to extract the employeeId from the path and display that particular employee details.

Firstly modify root component to show loginform



Let us hard code service to take dummy login details

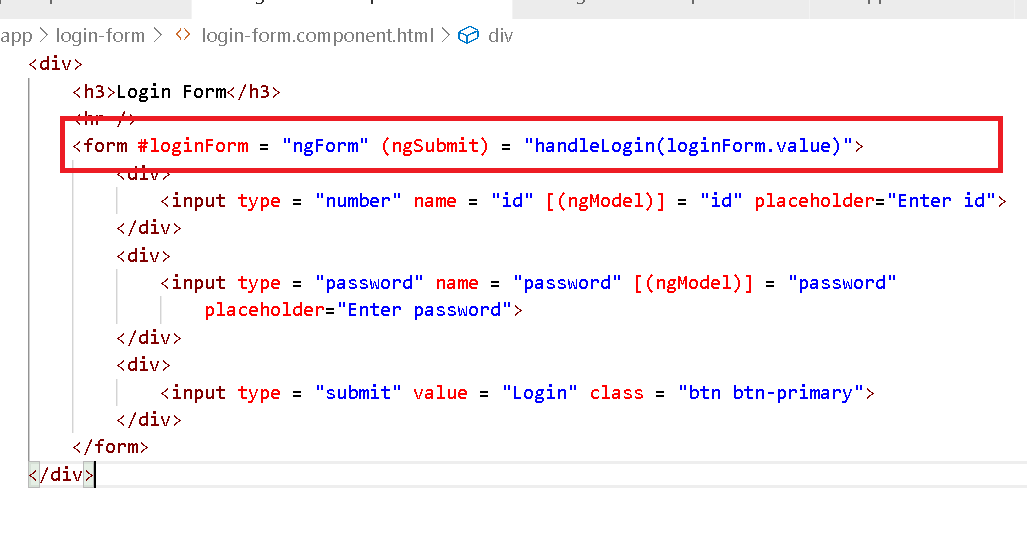


login-form.component.ts



It takes id & password & calls the login and routes to success/:employeeId or register

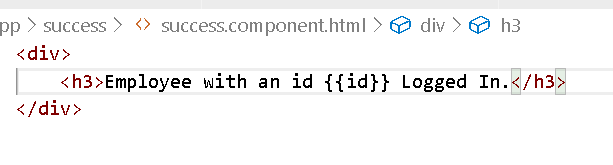
Now you can add login form in the login-form.component.html



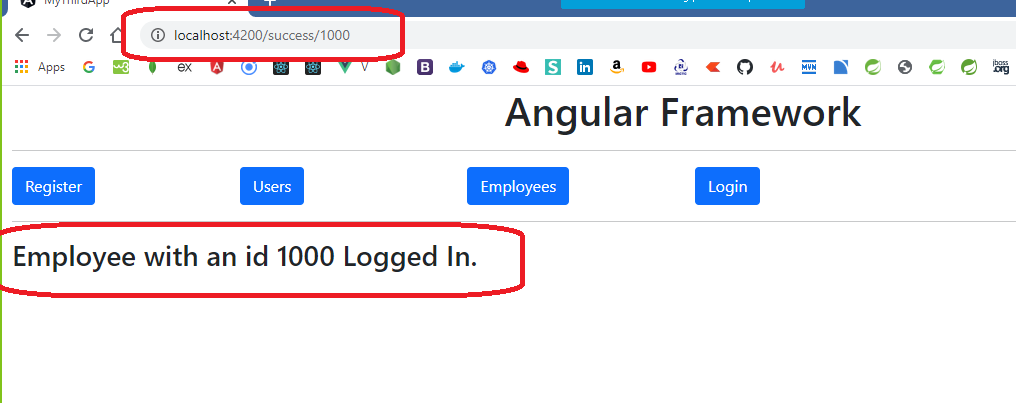
Now after successful login it takes you to success component, let us read the /success/:employeeId path parameter



You can display id in the html

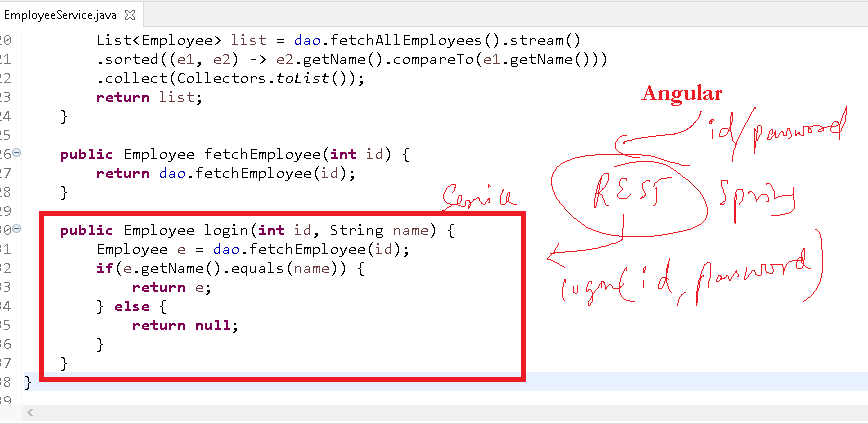


Output:

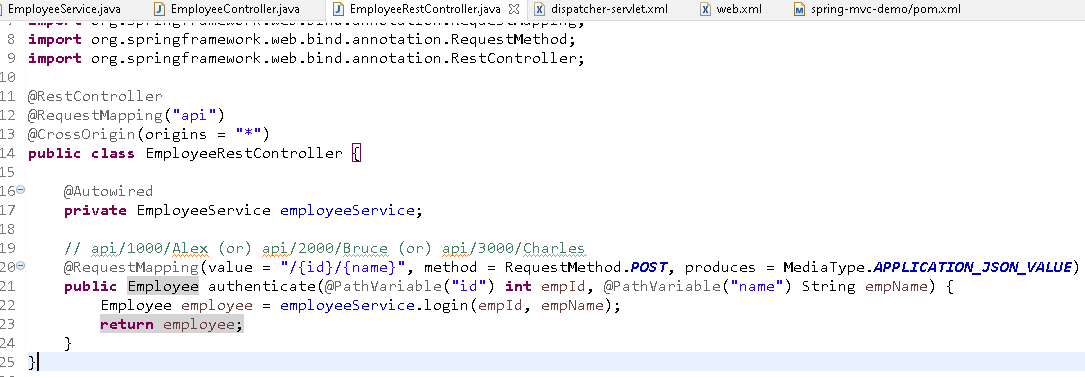


Accessing Spring REST API’s with Angular

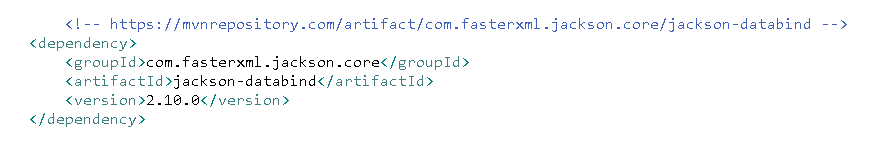
Existing Spring Service layer



Now you must create a REST end point in the spring



Modify the pom.xml to have Jackson library: this takes care of converting java to json and vice versa

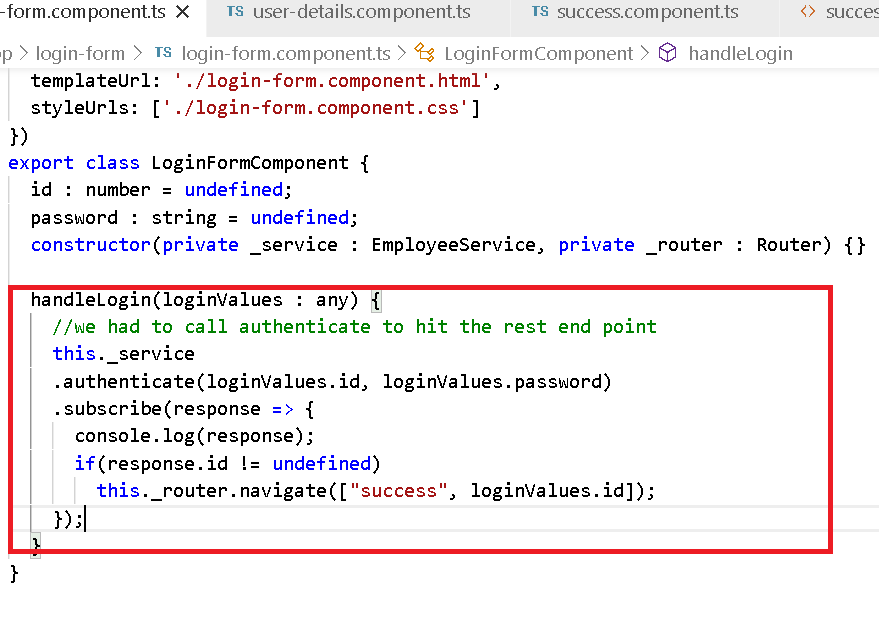


Now angular can send the id & name to the spring rest

employee.service.ts

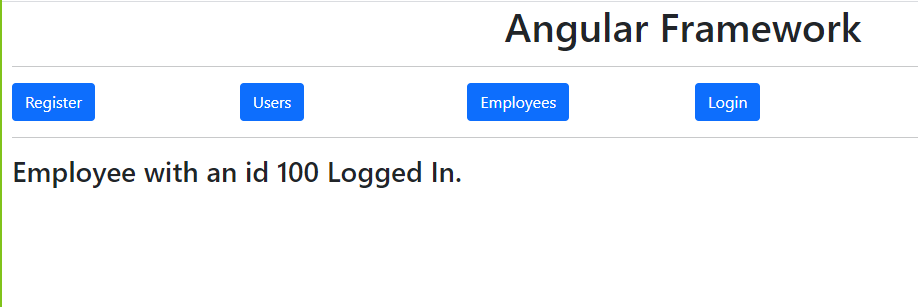


Modify the login.component.ts



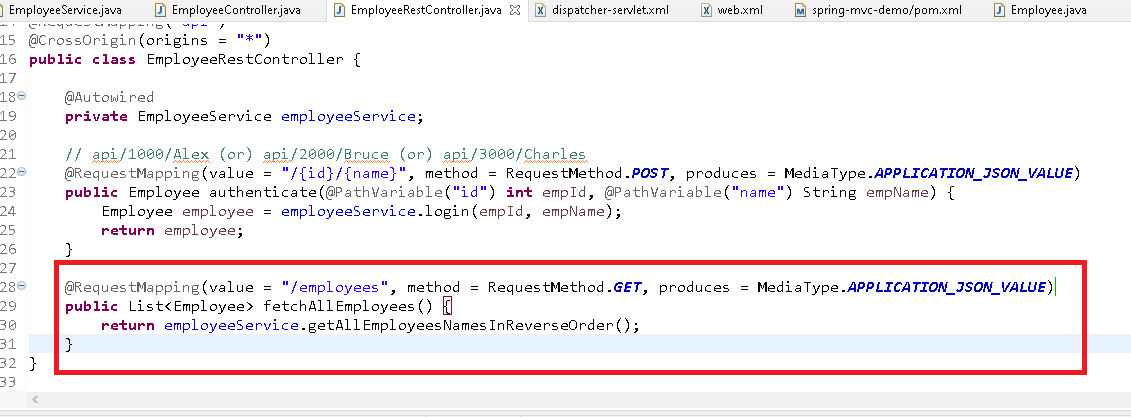
No changes in other files

Output:



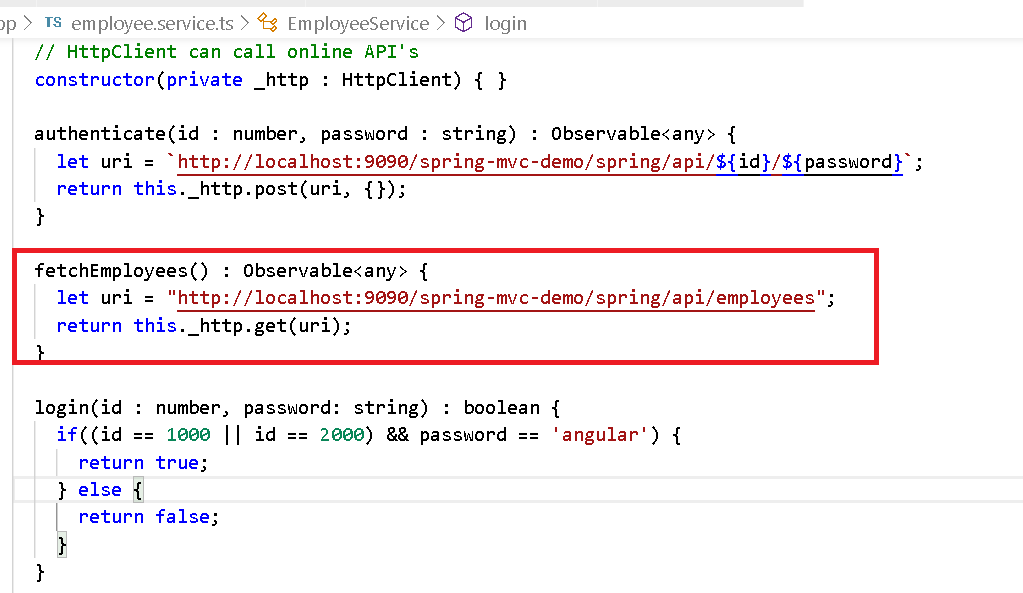
How to access all employees from the backend through angular

Call the method present in spring service to get all employees



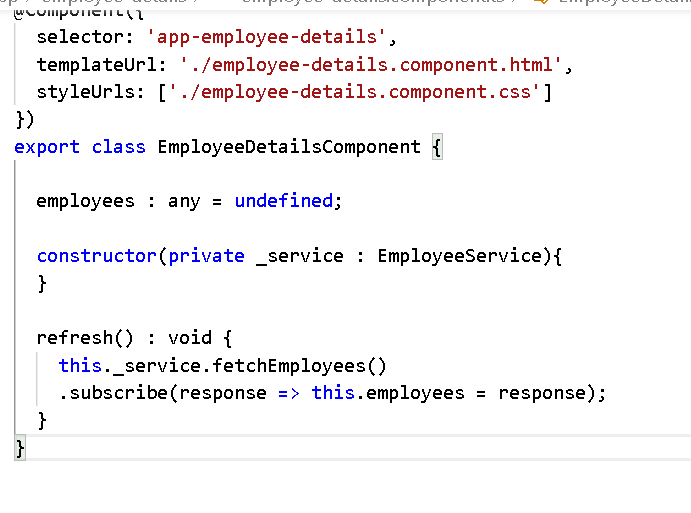
Now from angular you need to send request to /api/employees

employee.service.ts



Now employee-details.component can call fetchEmployees() and subscribe the Observable to get json response

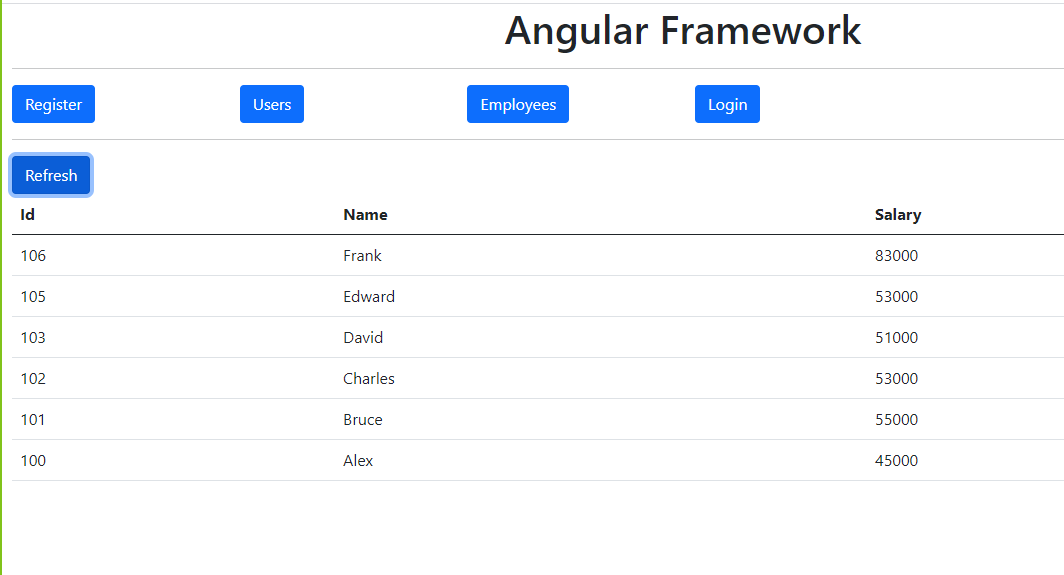
employee-details.component.ts



change dob to salary in employee-details.component.html



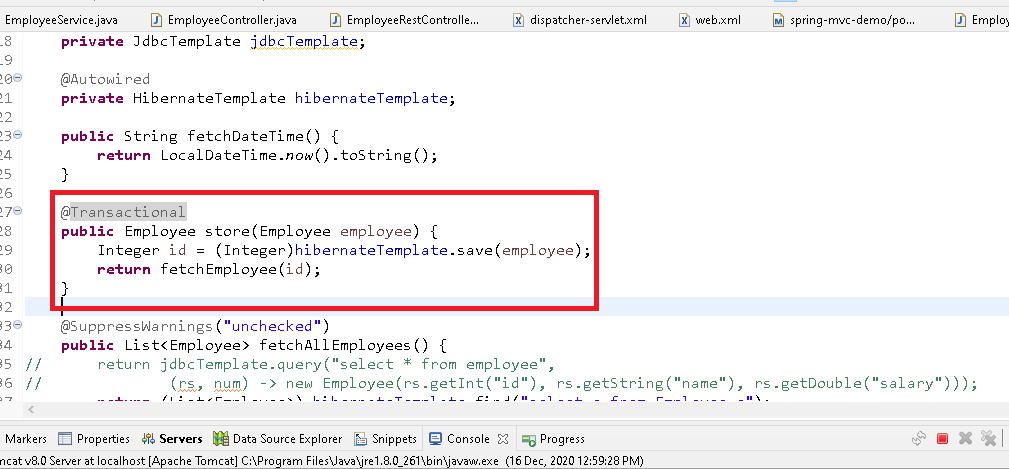
Output:



Let us send id, name & salary to the Webservice, firstly create a web service that can receive these in json structure

Firstly have a method in Dao to store employee

EmployeeDao.java

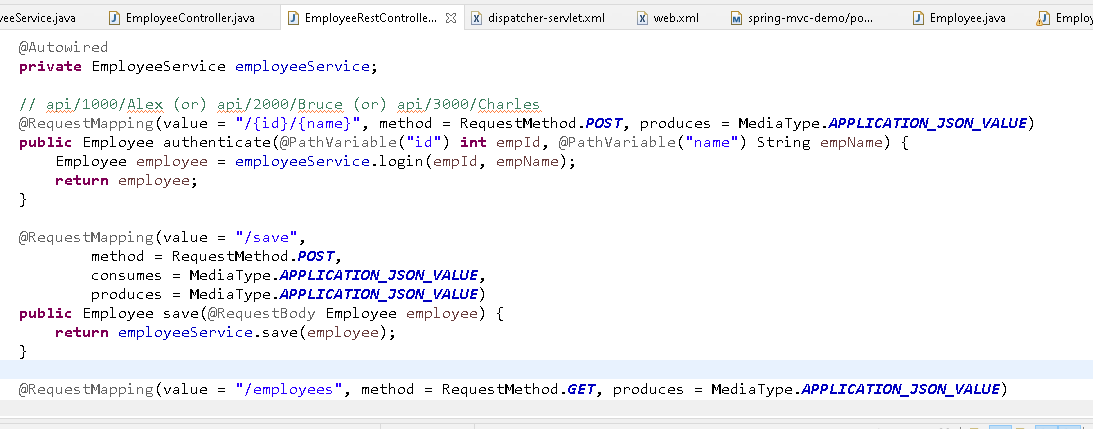


From service you access store & return the employee to the caller

EmployeeService.java



Now Restendpoint to call this save method



Now angular can send employee details to the /api/save

employee.service.ts



Now your employee-register.component.ts can send employee data



employee-register.component.html



Output:

