**Angular-Project-1:-**

**Step-1:-** Install Angular CLI from node package manager.



**Step-2:-** Create a new project of angular



**Step-3:-** Start the project on localhost and use the below command to refresh the page automatically whenever there is a change in change in the project.



**Step-4:-** Give the properties in that folder only by using double curly braces {{}}.



**Step-5:-** Create new component using below command



**Step-6:-** Import created component in the main component (app).

1. Using created component selector as tag in HTML file of app component.

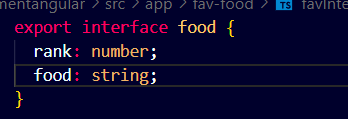


1. After that import that component in the main component ts file also.

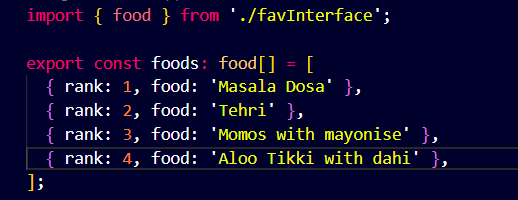




**Step-7:-** Create Interface for food and specify the type of properties we want.



**Step-8:-** Create an array of objects(favourite food) using the above interface.



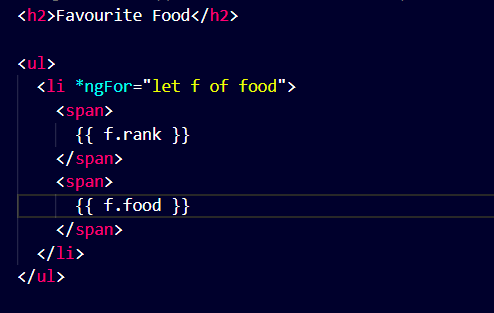
**Step-9:-** Traverse the array of object using for loop by using **NgFor**.

**Import in ts file of created component:-**





**Using in HTML file of created component:-**



**Types of Binding In Angular -**

1. **One Way Binding or Text Interpolation** - Its direction is from Component to DOM. It is done by using double curly braces.

Example- 

1. **Two Way Binding** - In this data can flow in both the direction from DOM to Component and from Component to DOM by using ngModel by importing it from FormModule.

**Example -** 

1. **Property Binding** - Its direction is from Component to DOM. It helps us to transfer some property from one component to another component.

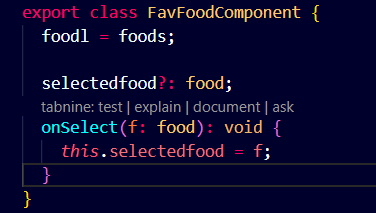
**Example -** 

1. **Reverse-Binding or Event-Binding** - Its direction is from DOM to Component. It help us to take some property from DOM and manipulate it according to the given conditions.

**Example** - 

**Directives** are ngFor, ngIf etc

**Step 9** - Define the onSelect function in order to access the selected value.

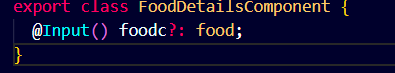


**Step 10** - Pass the selected value to another created component by using property binding.

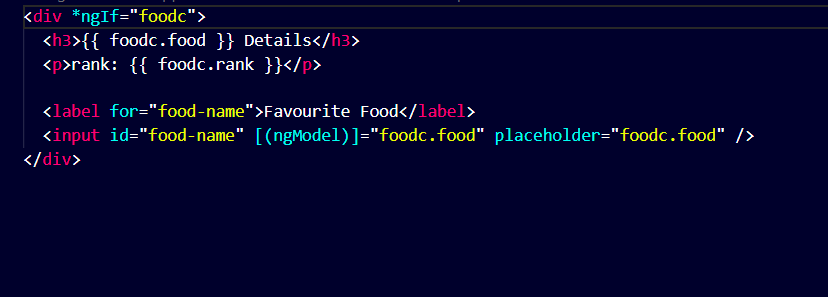


**Step 11**- Then apply the decorator (@Input) to take the input from other component





**Step 12** - After taking input apply the condition using ngIf and also applying two way binding by using ngModel by importing from FormModule.



**Dependency Injection:-** Dependency Injection, or DI, is a design pattern and mechanism for creating and delivering some parts of an application to other parts of an application that require them. Angular supports this design pattern and you can use it in your applications to increase flexibility and modularity. It help us to pass the data to other components and if we want to make changes in supply of data we can do it directly from services . We don’t need to make any changes in component.

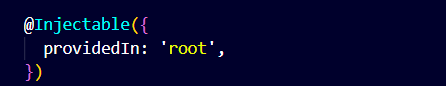
**Service in Angular -** Angular services provide a way for you to separate Angular app data and functions that can be used by multiple components in your app.

**Command to create new Service -**



**Provider in Service -** They are key to its Dependency Injection (DI) system, creating and configuring dependencies for applications

**ProviderIn: root** -Determines which injectors will provide the injectable.



**@Injectable -** Decorator that marks a class as available to be provided and injected as a dependency.

**Step 13 -** Make the service observable so the method can return observable list. It means that it will notifies the component whenever the data is present or come.

**Steps to make service observable:-**

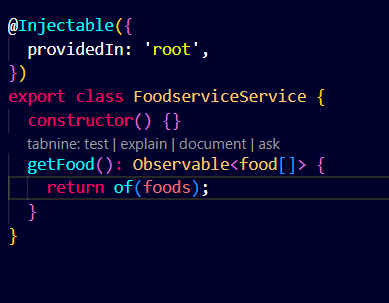
**Import Observable and of from rxjs**

****

**Observable-** Observables are a powerful feature used extensively in reactive programming to handle asynchronous operations and data streams. · Observables provide a way to subscribe to and receive notifications when new data or events are emitted, enabling you to react to changes in real-time

**Of -** Converts the arguments to an observable sequence. Each argument becomes a next notification.

**Usage of Observable and of -**



**Step 14 -** For using service in an component

1. Import the service in the respective component.



1. Initialise the empty array of given interface type

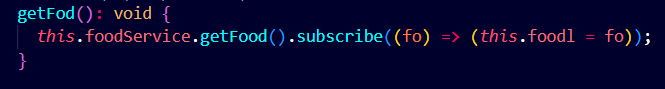


1. Then make the constructor in order to inject the service in the component

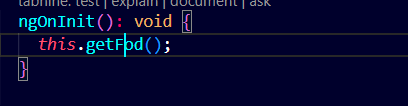


1. Make a method to access or notifies whenever the observable data has come or available

**subscribe()** - An observable begins publishing values only when someone subscribes to it. That "foods" observable won't emit any numbers until you subscribe by calling the observable's subscribe() method.



1. Then apply one of the Component lifecycle method (ngOnInit) in order to initialize the observable data which is received.



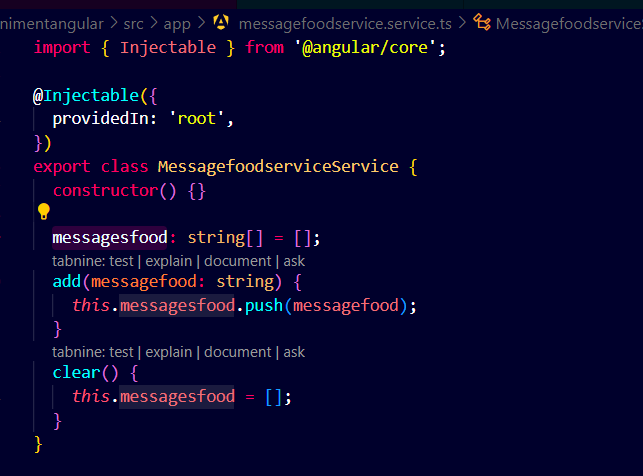
**Step 15** - Make another component for giving the details of food.



**Step 16** - Make another service for the newly created component.



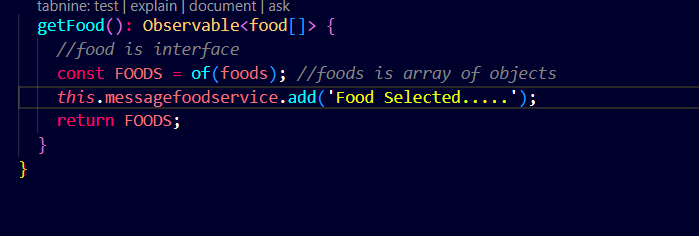
**Step 17** - In newly created service create a add function to add the message or detail in the array and create a clear function to clear the array.



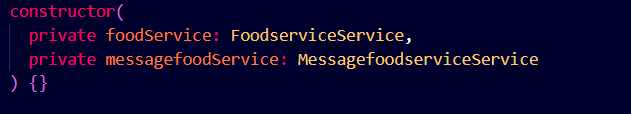
**Step 18** - Inject the created service(messagefoodservice) in food Service.

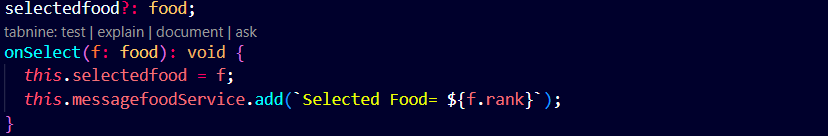


**Step 19** - Call the add function in order to print Food Selected already on screen while returning observable list.



**Step 20** - Inject the messagefoodservice in favfoodcomponent in order to call the add function in that.

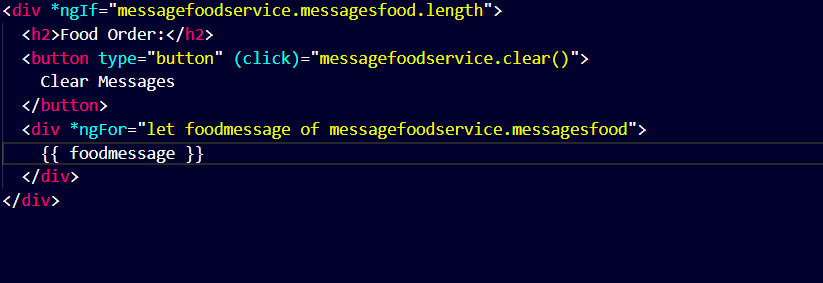




**Step 21** - Inject the messagefoodservice in message component. Public is used as we are using the property of messagefoodservice in the html file of message component.



**Step 22** - In html file of message component give the condition that it will run only when the message is present. Also add the clear functionality and iterate through the messages.



**Step 23** - Import messagefoodcomponent in app component in order to render on the screen.

In HTML-

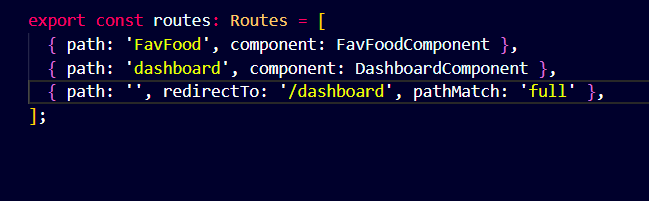


In ts-



**Step 24 -** Giving Routes

* Give the path and component name in app.routes.ts file

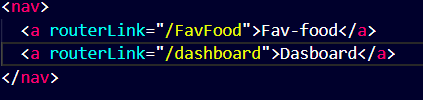


* Give router-outlet tags in html file and import it in ts file





* Give the path and anchor tag to the particular end point using routerLink

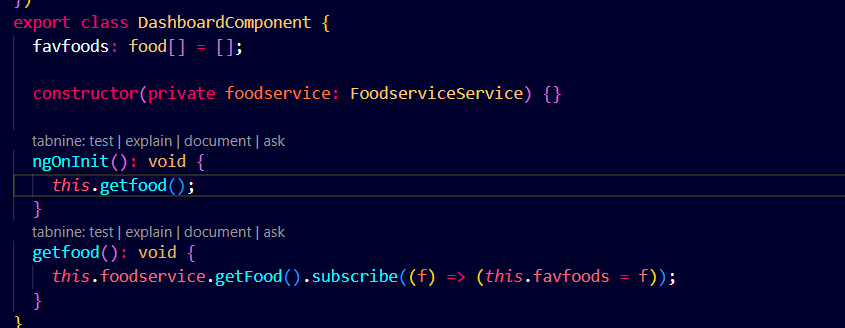


* In order to routerLink we have to import RouterModule in ts file.



**Step 25 -** Create a new Component Dashboard

**Step 26-** Give the data to this component using service .



**Step 27** - In html file of dashboard component iterate over each element and display it.



**Step 28** - Give the path of dashboard component in app.routes.ts file



**Step 29** - Give the anchor tag and the path of dashboard component.



**Step 30** - When there is no end point then it will redirect us to the dashboard



**Step 31** - Routing using particular id of a person

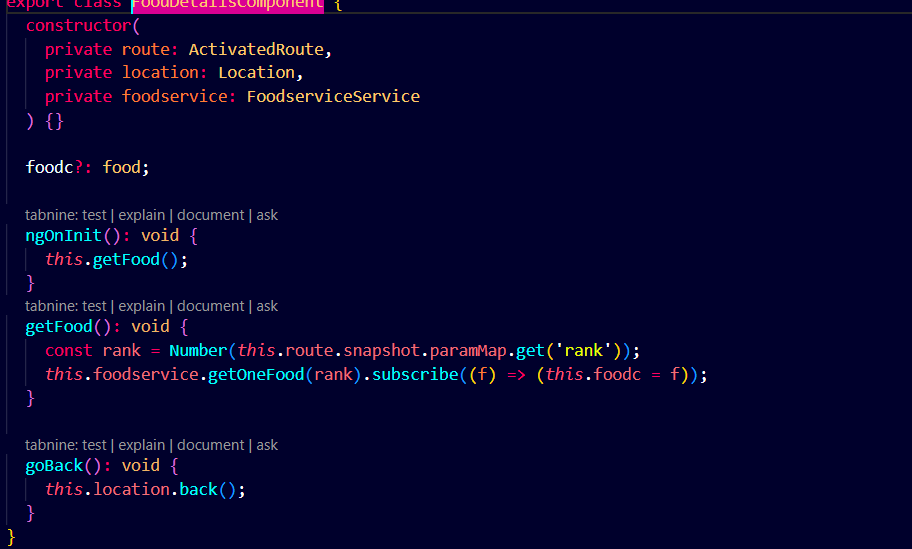
Giving Path-



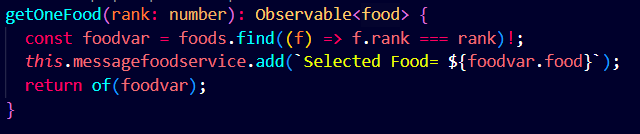
**Step 32 -** Then in FoodDetailsComponent

Import activatedroute for accessing the id , location for going back and foodservice for displaying the data according to that id.

Make to a method in order to send the id to foodservice. Make another method in order to go back

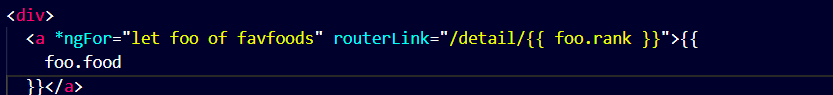


**Step 33** - Write the logic for accessing the detail according to selected id in foodservice to send the obtained data.

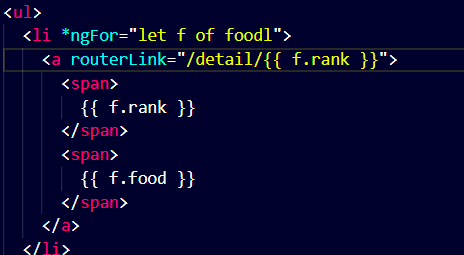


**Step 34** - Give route where we want the above functionality to display the details.

In dashboard component:

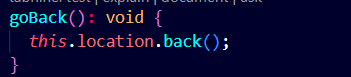


In fav-food component:



**Step 35** - For Back functionality first import Location in foodDetailComponent then make a method getBack(). Apply this functionality to a button in html file.







**Step 36 -**

Create a Mock DataBase in order to test the code-

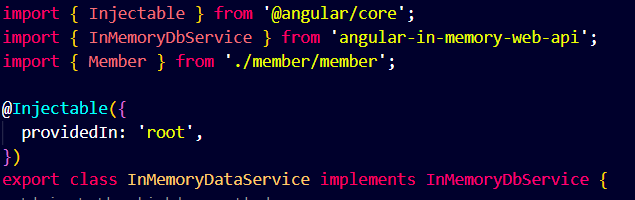
1. Create a service for Mock server



1. Install angular-in-memory-web-api



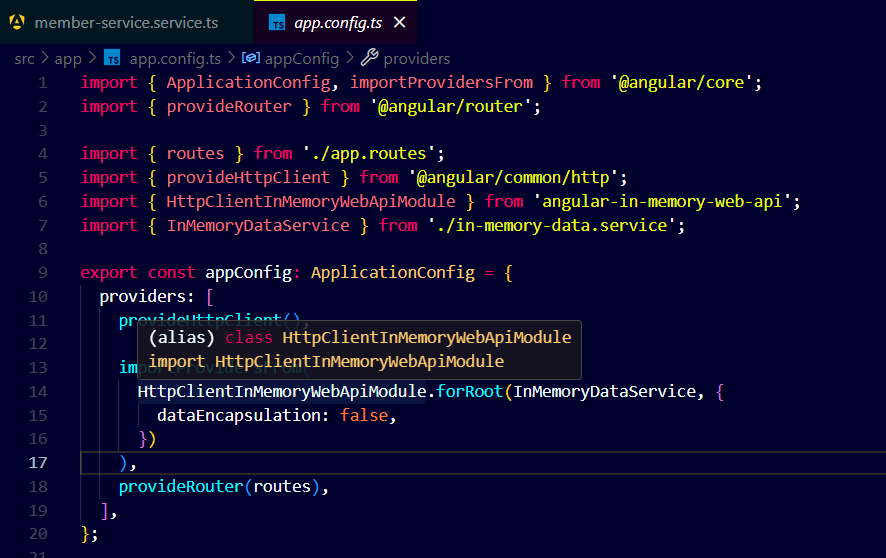
1. Import the InMemoryDbService and implement that interface in service in which mock server has to be made.



1. Create a mock database or server and a function to generate the id of newly added member



1. Make some configuration in app.config.ts file-



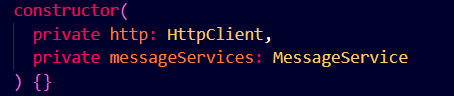
**Step 37 -**

Make some changes to member service file -

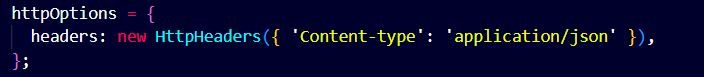
Import HttpClient, HttpHeaders-



HttpClient - for sending the request to the server.



HttpHeaders - It specifies the type of data to be passed.



There are 4 types of request -

GET Request - To retrieve the data from database.

POST Request - To add some data to database.

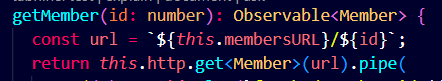
PUT Request - To update some data in database.

DELETE Request - To delete some data from database.

In getMembers method -



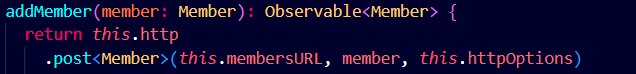
In getMember method -



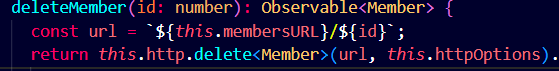
In Update Method -



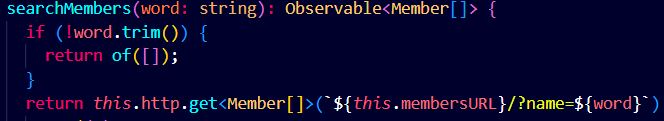
In AddMember Method -



In deleteMember Method -

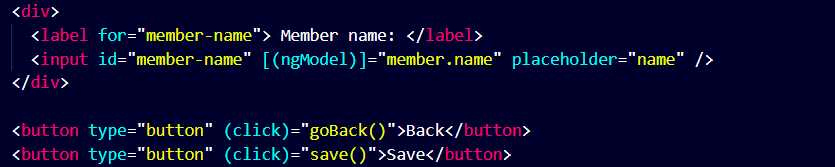


In searchMembers Method -

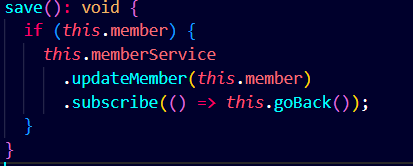


In member-detail component html file we make a save button and call the save function in order to update the member detail - tap will not modify the data

In HTML file -

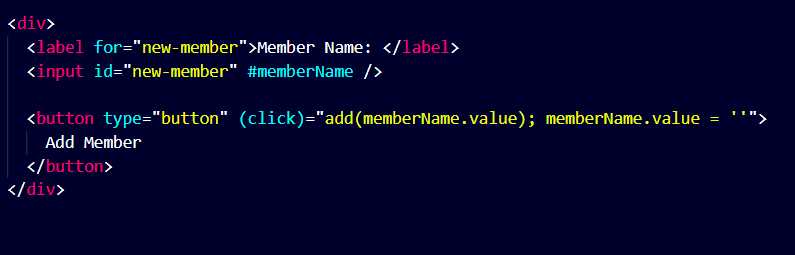


In ts file -

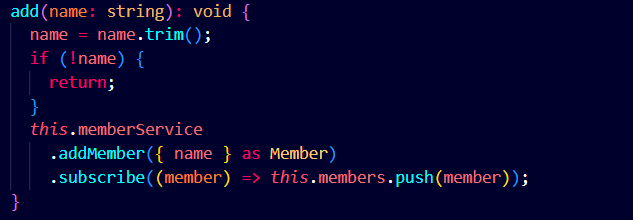


In Member Component we have to add to add member functionality -

In HTML file -



In ts file -

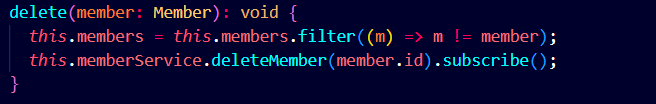


In Member Component we can also implement the delete functionality -

In HTML file -



In ts file -



For Searching Functionality -

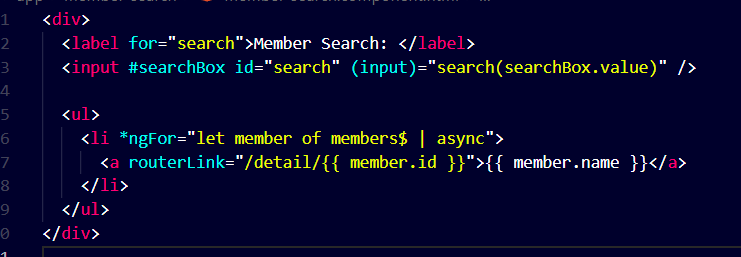
Make a new component named member-search.

Use that component in dashboard component -

In dashboard component HTML file -



In member-search component HTML file -



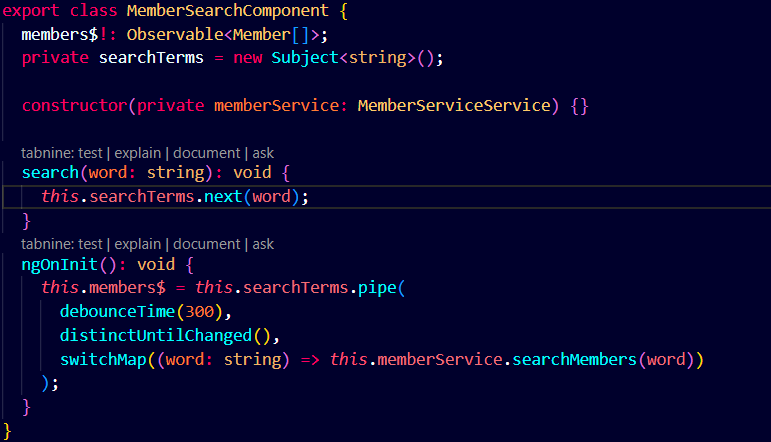
#searchBox - It is used for accessing the input value

(input) - It represents whenever there is change in input then the function will trigger.

$ - This symbol is show that it returns observable.

| async - It represents subscribe

In member-search ts file -



Subject - It is similar to Observable but in it we can add things using next keyword.

this.searchTerms.next(word) - It is used to add another character to the array.

.pipe - It is used to add multiple functionalities.

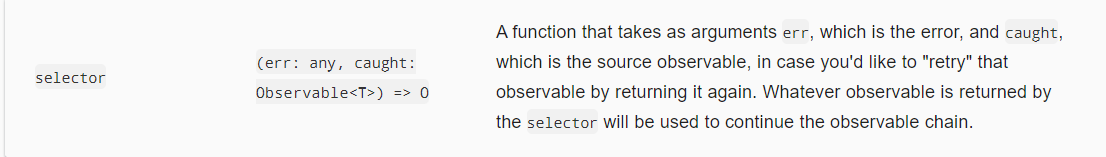
debounceTime - It helps us to add delay while triggering a function.

distinctUntilChanged() - It checks if there is a change then only trigger the function.

switchMap() - If both the conditions are fulfilled then the required function will triggered and the data is passed

**Error Handling** - One traditional way of handling errors in Angular is to provide an ErrorHandler class. This class can be extended to create your own global error handler. This is also a useful way to handle all errors that occur, but is mostly useful for tracking error logs.

catchError - Catches errors on the observable to be handled by returning a new observable or throwing an error.

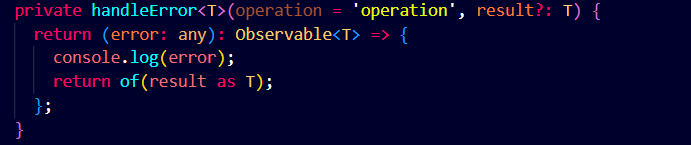


Tap - Used to perform side-effects for notifications from the source observable.

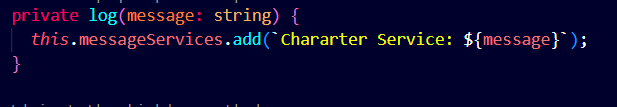
Importing catchError and tap -



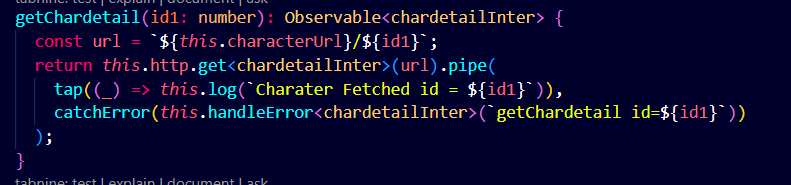
Declaring handleError function -



Declaring log function -



Applying the error handling -

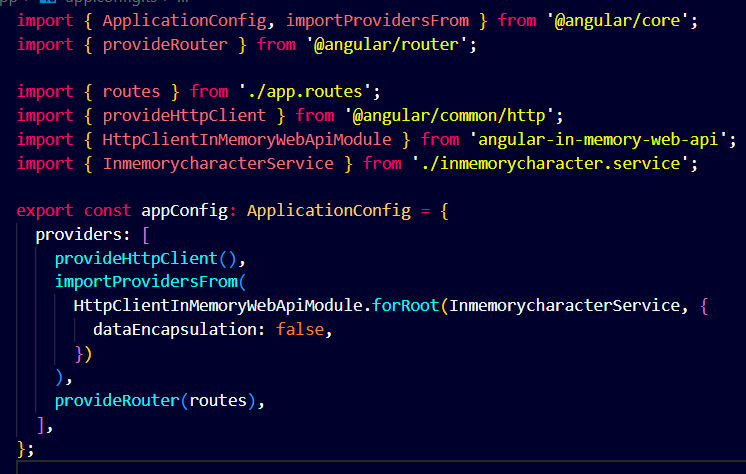


**Changes made in Doremon\_Explorer\_hub**

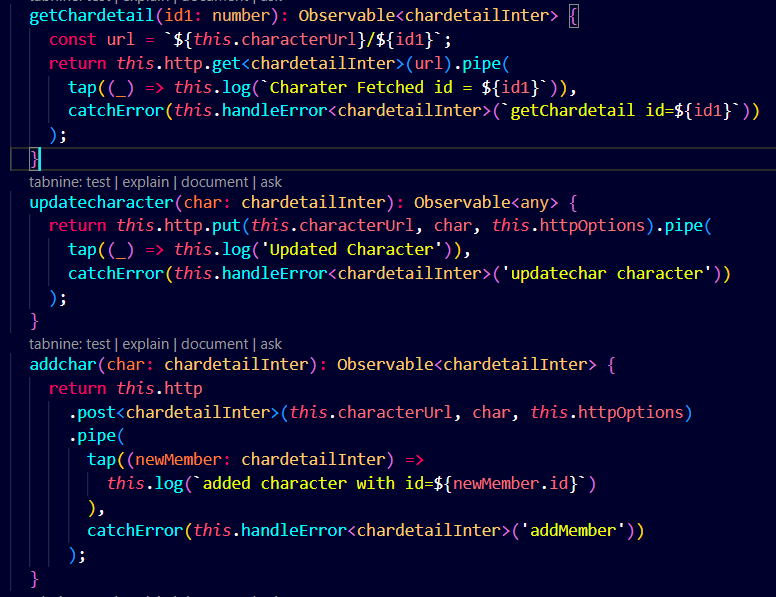
**Step 1 -** Made a in memory web api. Basically i made a mock server in order to test the angular code.



In app.config.ts file -



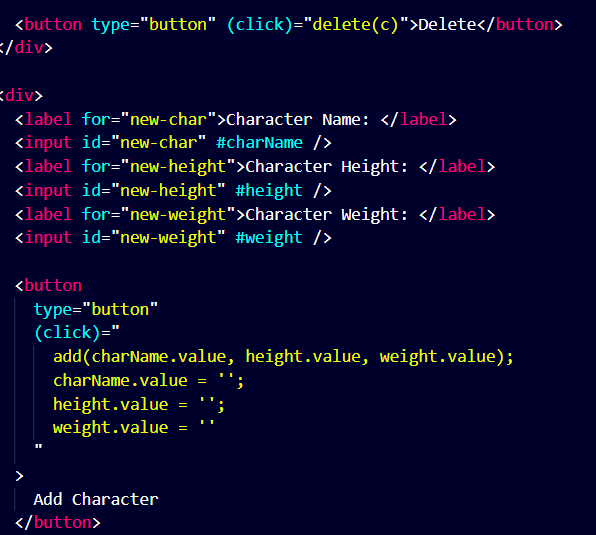
**Step 2** - Make request from services



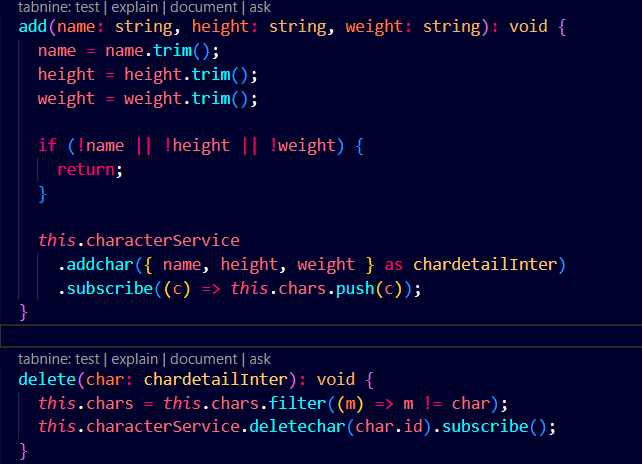
**Step 4 -**

Add and Delete Character -

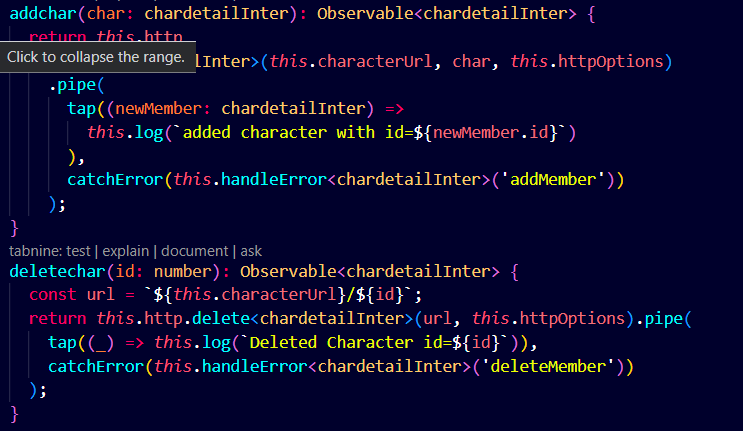
HTML file -



Ts file -

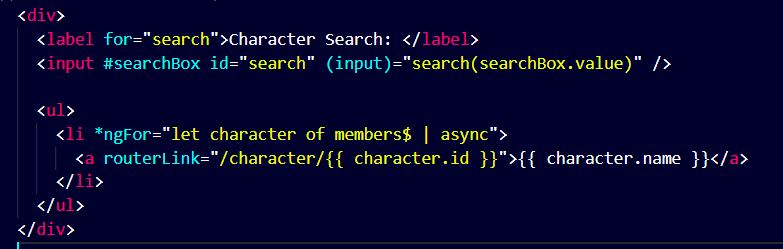


**In Service making request -**

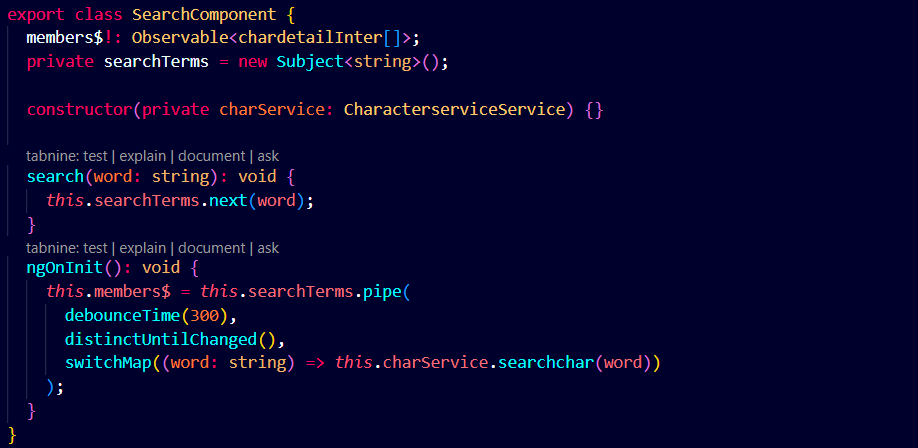


Search Component -

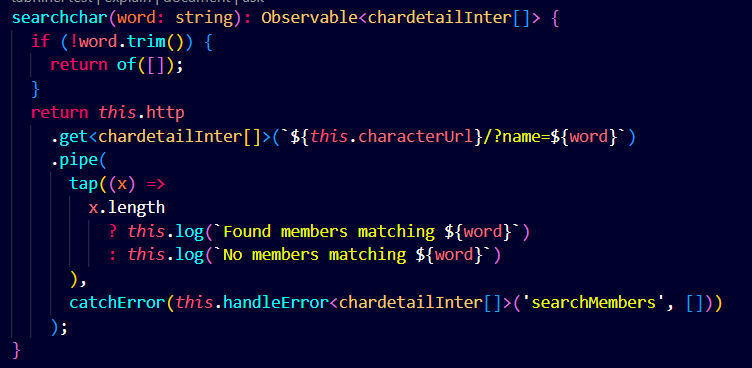
HTML file -



Ts file -

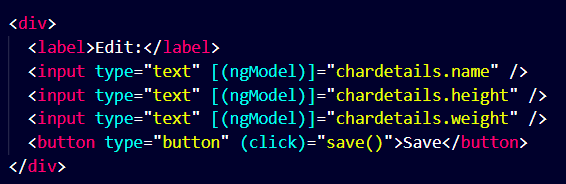


In Service making request -

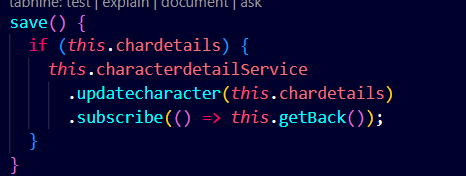


Update Character -

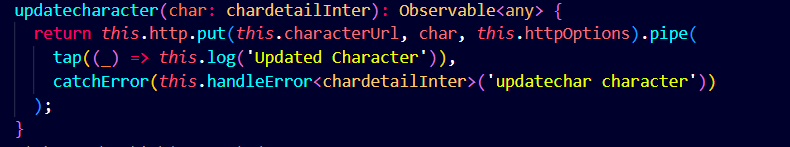
HTML File -



TS File -



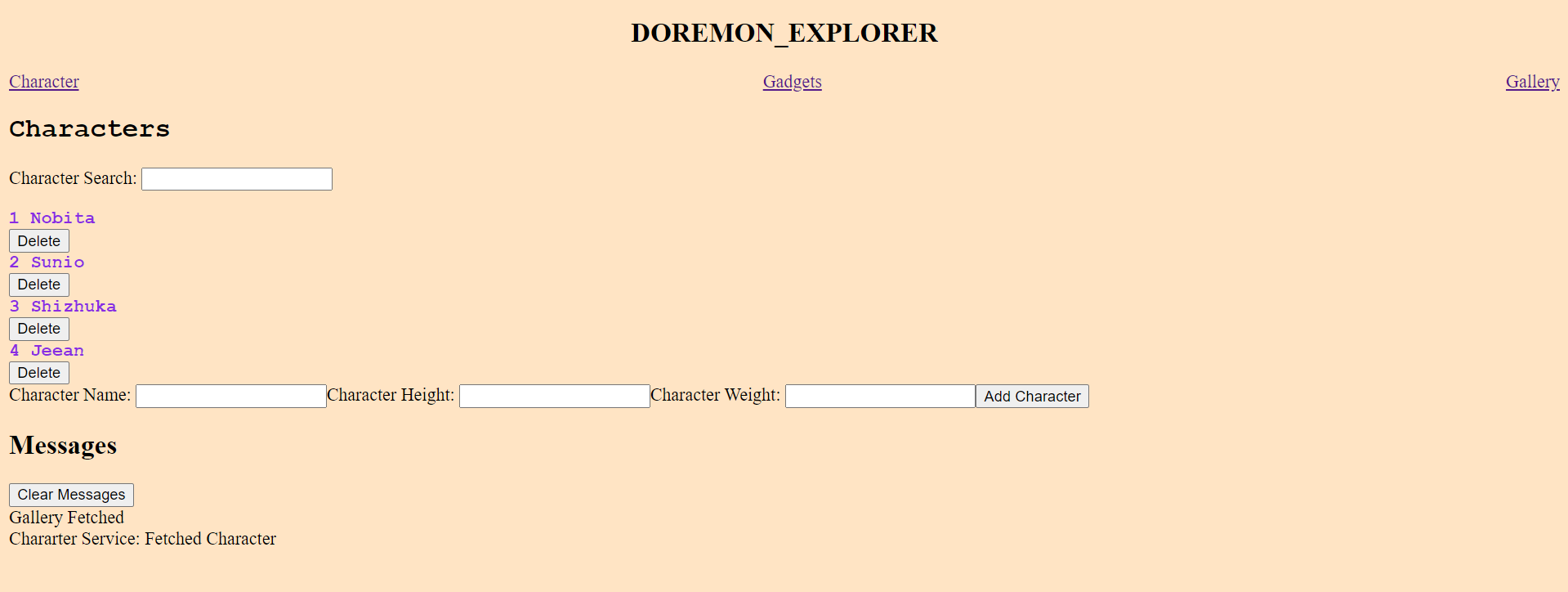
In Service making request -



Same functionalities are also in gadget component.

**Output -**

Character-Component



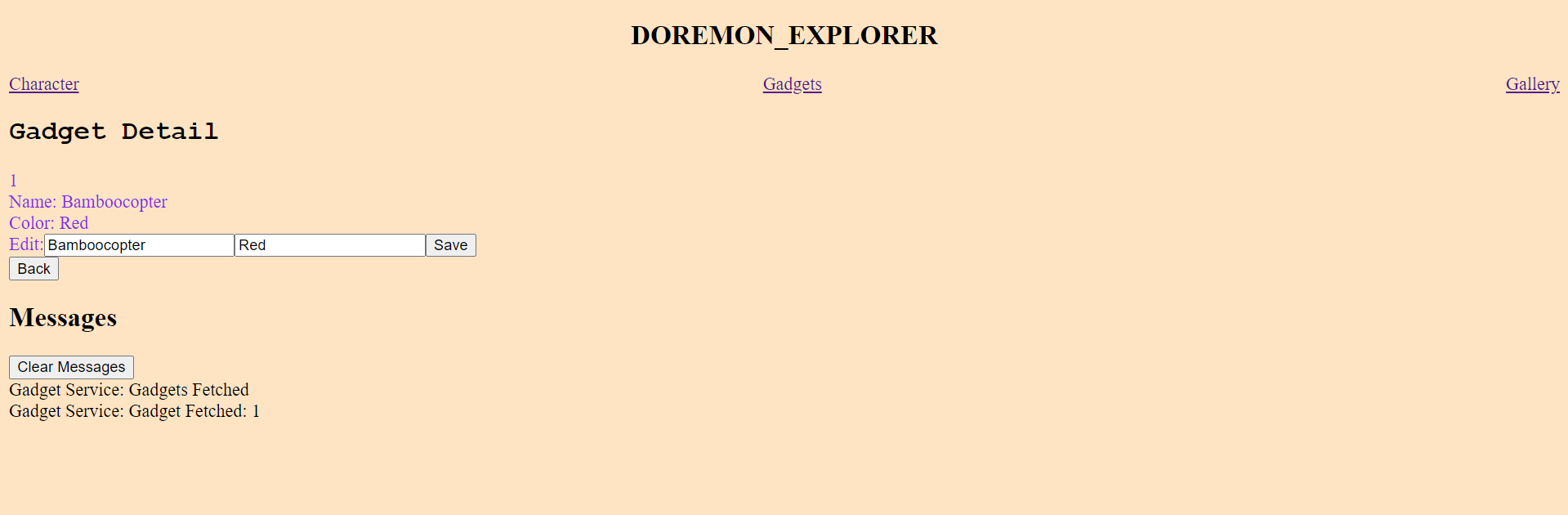
Character-Detail Component -



Gadget-Component -



Gadget-Detail Component -



Gallery Component -

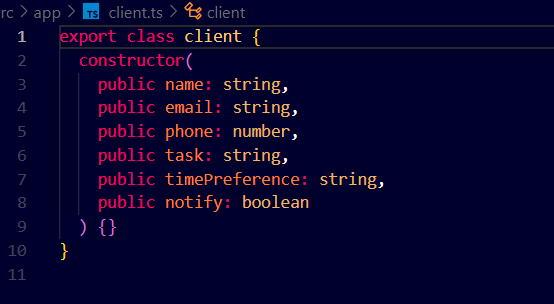


**Form Using Bootstrap and angular -**

**Step 1 -** Apply the CSS and JS link in index.html file in order to use Bootstrap.



**Step 2** - Make a interface for a member



**Step 3** - app.component.ts file



Common Module - It is imported in order to use the pipe.

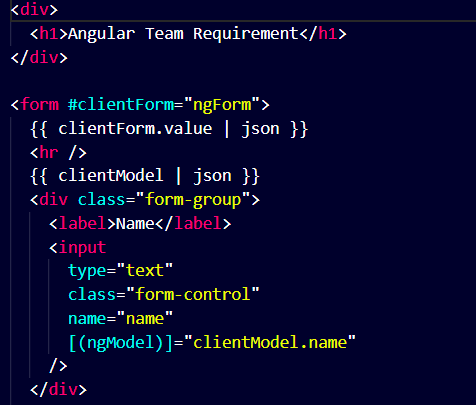


Pipe - It is used to convert the data in json form.

Tasks array - It is used in order to make drop down menu.

clientModel - We made a member of client(interface) type.

HTML file -



ngForm - It is used in order to get hold on all properties present inside the form tag.

#clientForm - It is used to get hold on the values of the form element. The #clientForm assigns a template reference variable to the form.

{{clientForm.value | json}} - It is showing the changes made in form element by converting it in json form and displaying it

name - It is used as json uses key-value pair.

ngModel - It is used for two-way binding by using this if we made changes in input field it also make the changes in other part wherever this code is present.

[ngModel] - If we don’t use round brackets then it don’t show the functionality of two-way binding. It will show one-way binding.

There are 6 types of angular classes for form validation -

1. Ng-touched - It means that we have clicked that field.
2. Ng-untouched - It means that we have not clicked that field yet.
3. Ng-dirty - It means that some changes have made in input field or in textarea field.
4. Ng-pristine - It means that no changes have made in input field or in textarea field.
5. Ng-valid - It means that input has data matching with the conditions.
6. Ng-invalid - It means that input has data that is not matching with the conditions.

 - It means that class is invalid if name is empty as required is given and also when we have clicked it.

 - It means that do not display anything if the name field is having the data or it is not clicked yet.

 – It means that if any error occurs and phone is invalid means it has to be of 10 digits and it is clicked.

 - It means error occurs of required type.

 - It means that phone number has to be of 10 digits.

 - It means error occurs of pattern type.

 - Blur is used if we don’t make any changes in the selected option of drop down menu.

 - Change is used if we make some changes in the selected option of drop down menu.

 - It is used so that we able to use classes like valid, invalid, pristine, dirty, touched and untouched.

 - taskHasError is declared true in ts file if the value is equal to default which is (required for the task) then it the warning or error (task is required).

 - timePreference is the property defined in the interface.

 - It means that if the form is invalid or taskHasError is true then the submit button will be disabled.

Conditional-styling - 

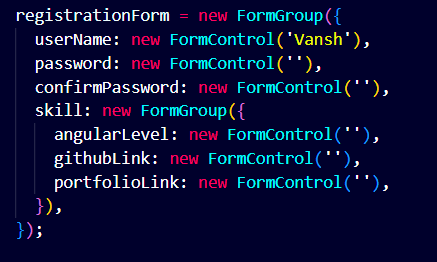
Reactive Form using form-group and form-control -

Form-group - In Angular, FormGroup is a class provided by the @angular/forms module that represents a collection of form controls and their associated validation states. When you're building forms in Angular, especially reactive forms, you often use FormGroup to manage and validate the form controls.

Form-control -

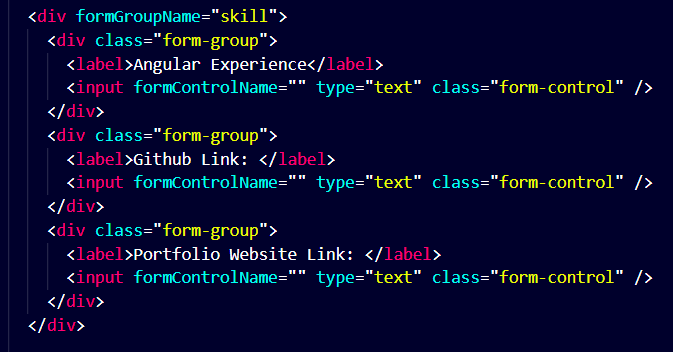
In Angular, FormControl is a class provided by the @angular/forms module that represents a single input control in a form, such as an input field, textarea, or select dropdown. FormControl is used to track the value and validation state of an individual form control.

In ts file -



In html file -

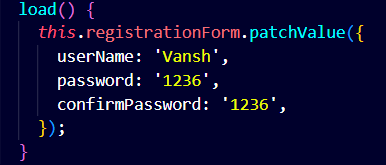


Nested Form-Group - 

To give sample data -



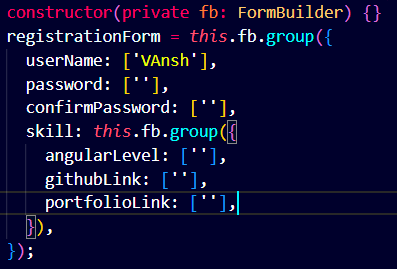
In ts file we can define the load function -



setValue - In this we have to set value for all the fields.

patchValue - In this we have to set value for only that fields which we want.

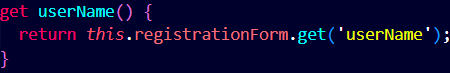
In ts file - using formBuilder



The main difference between formbuilder and form-control is that with formbuilder we can apply validators.

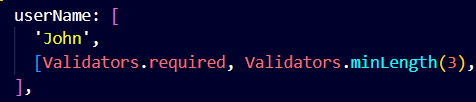
**Applying Validators -**

In order to get hold of userName which is present in the form we use -

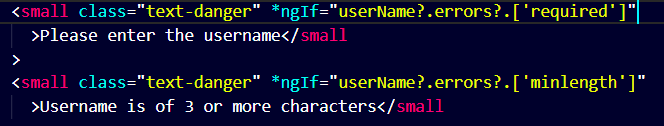
****

* Applying Inbuilt Validators -

In ts File - Import Validators



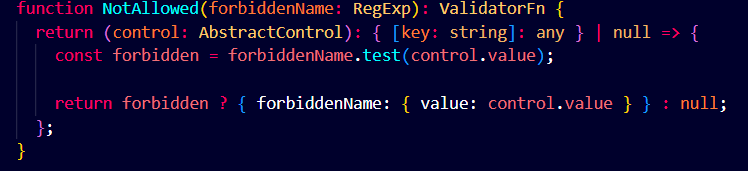
In HTML File -



errors means that some error has occurred and required and minlength are the type of errors.

* Creating a custom validator -

Make a function in which we define the validator



forbiddenName - It is used for taking the argument of Regular Expression type.

ValidateFn - A function that receives a control and synchronously returns a map of validation errors if present, otherwise null.

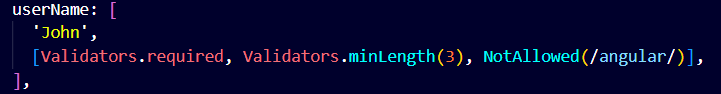
Control - It takes the hold of form-control or form-group

{ [key: string]: any } | null :- It returns a key-value pair or null.

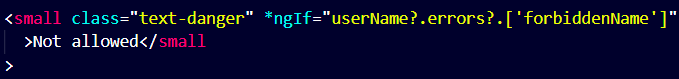
forbiddenName.test(control.value) - It is basically checking the equality between the value coming from argument with the value which we enter in the input field.

If it is matches then it will return the forbiddenName with its value otherwise it will return null.

Applying the custom validator in the userName which is a part of form-group -



In HTML file - Applying the custom validator in HTML file.



Applying the conditional validation - Example in this project if we click on notify me then only the email field is required otherwise it is not mandatory.

In ts file -

