**Angular 5**

* Angular 5 was announced to the world on 1st November, 2017
* The previous version was Angular 4.4.0
* This release focused on making Angular smaller and faster to use.
* Created and maintained by Google
* Front-end/ClientSide JavaScript Framework

Features:

* **Modularity** – all modules are maintained by different developers and easily maintained
* **Consistency –** importing all package in angular standard and then using it
* **Maintainability, Productivity** and ability to catch errors early.
* Angular 5 is designed with better mobile support.
* In Angular 5, “everything is component”so itfacilitated better code reuse.
* Cross Platform, Single Page Application (SPA)
* Angular is built using Typescript

**How to Install: install npm (3 or >), node (6.9 or >)**



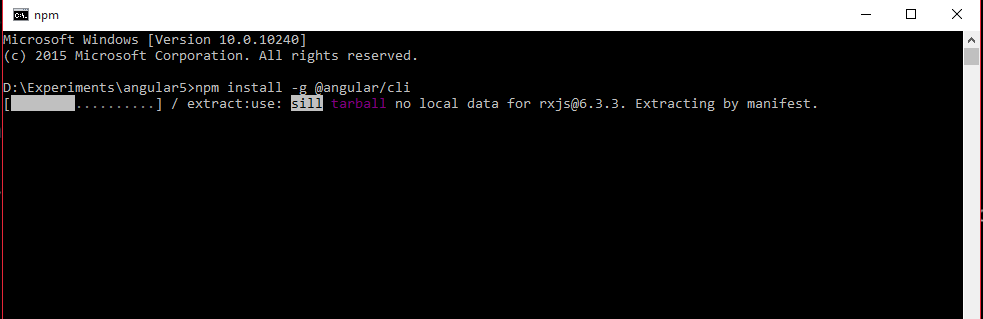
Now go to the google type **angular cli:**

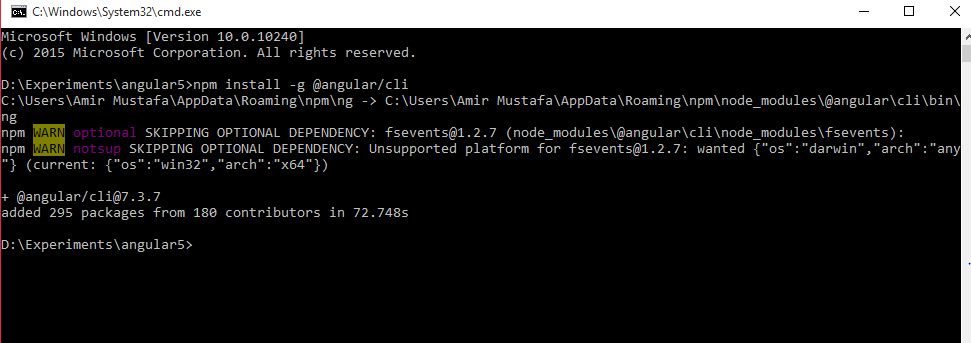


We will follow this command line and server angular using this command line

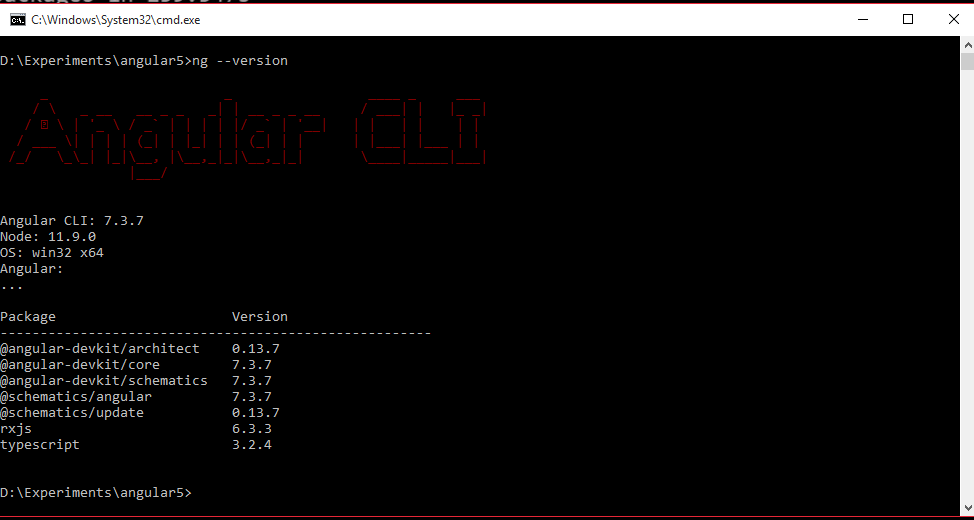
* Create a folder where you want to angular and there we will install globally

**npm install –g @angular/cli**

****

****

* So when you will run ng –version you will see Angular CLI information

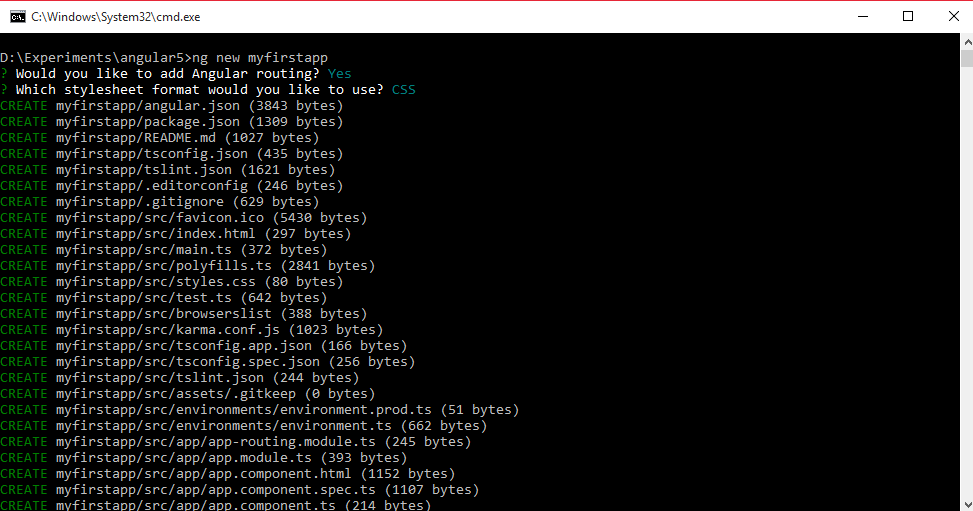


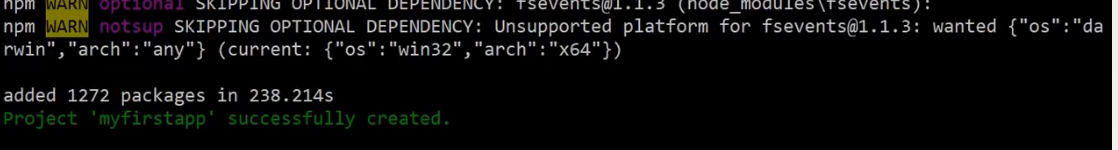
* So all your installed apps will inside this directory (where angular CLI is installed)
* So to create a new project:

**ng new <your\_project\_name>**

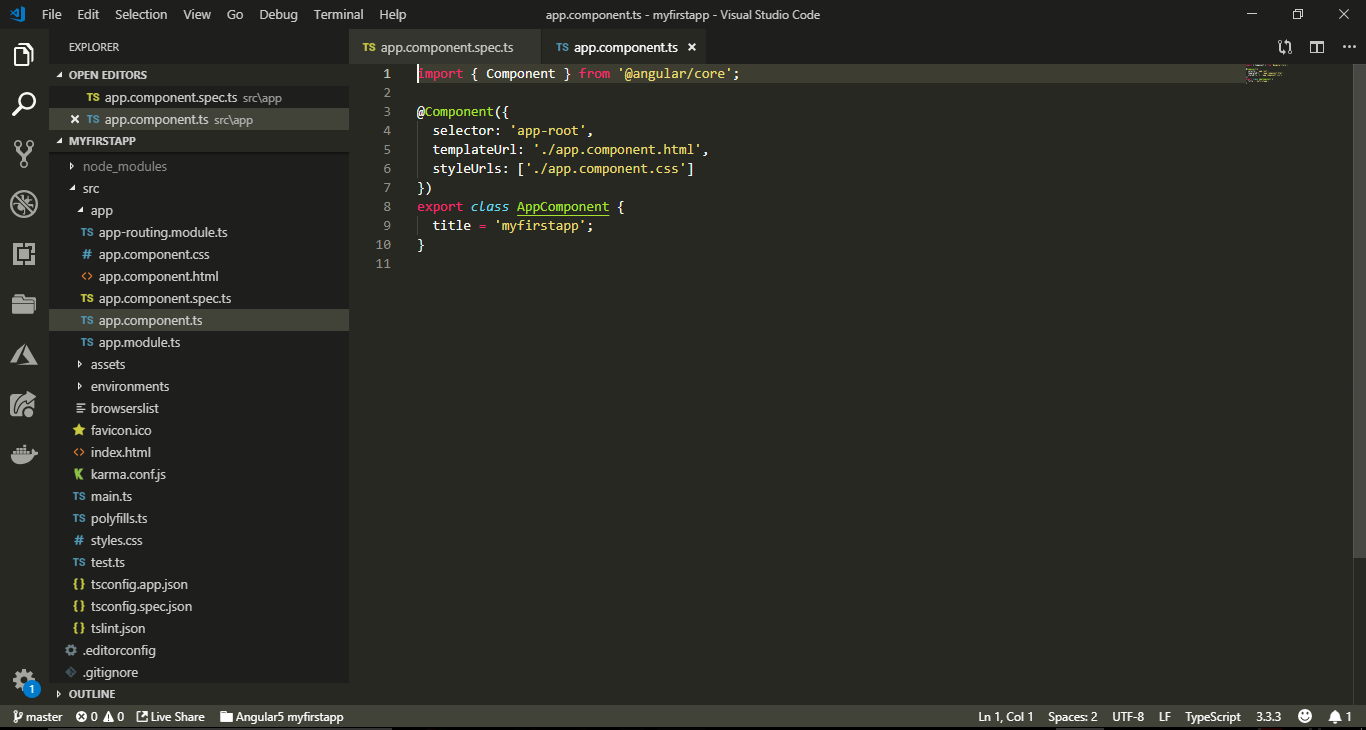
ng new myfirstapp

This takes some time to install

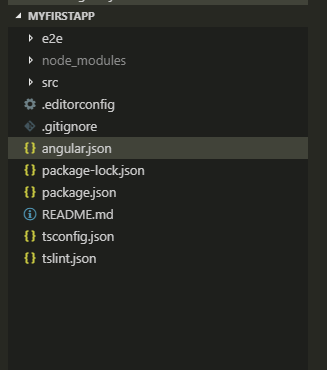




Project: myfirstapp in browser

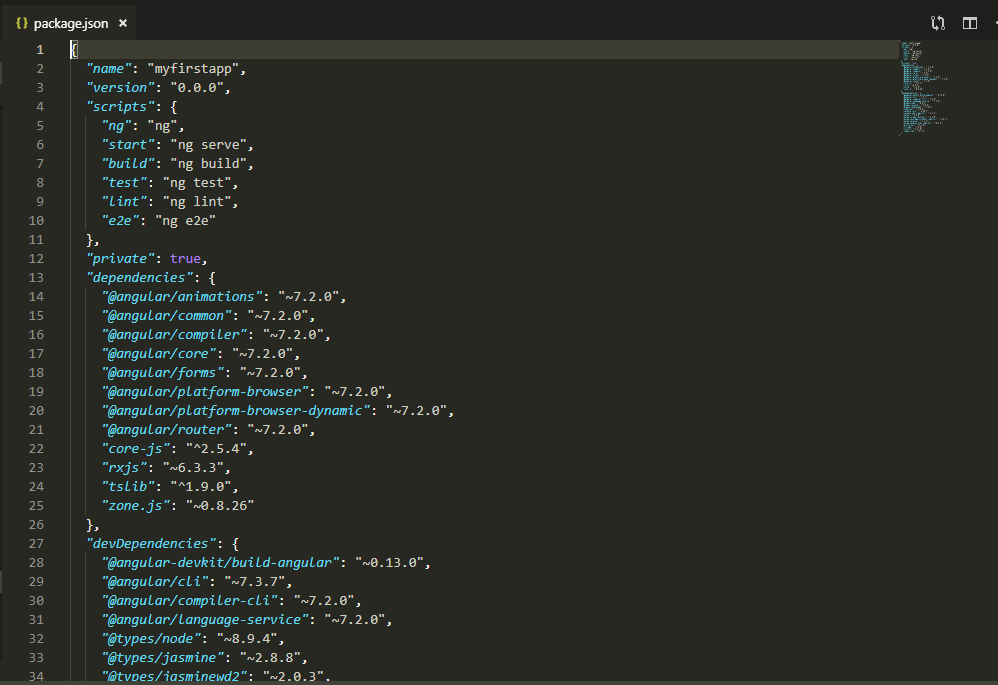


**Directory Structure in Angular 5**



1. package.json:

This is the file where all the dependent libraries related with your angular / project is written



Here:

**Name** – name of the project

**Scripts** – here we will the scripts are written. Generally here we will use typescript and not javascript. Typescript is the superset of js.

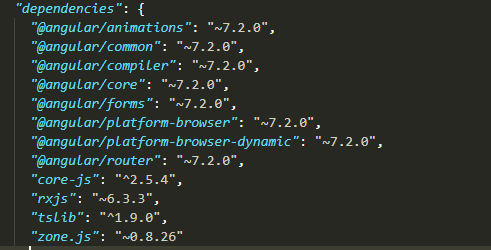
**Ng –** angular

**Start –** angular server (we will use them as a server)

**e2e – end to end**, basically used for the testing purpose

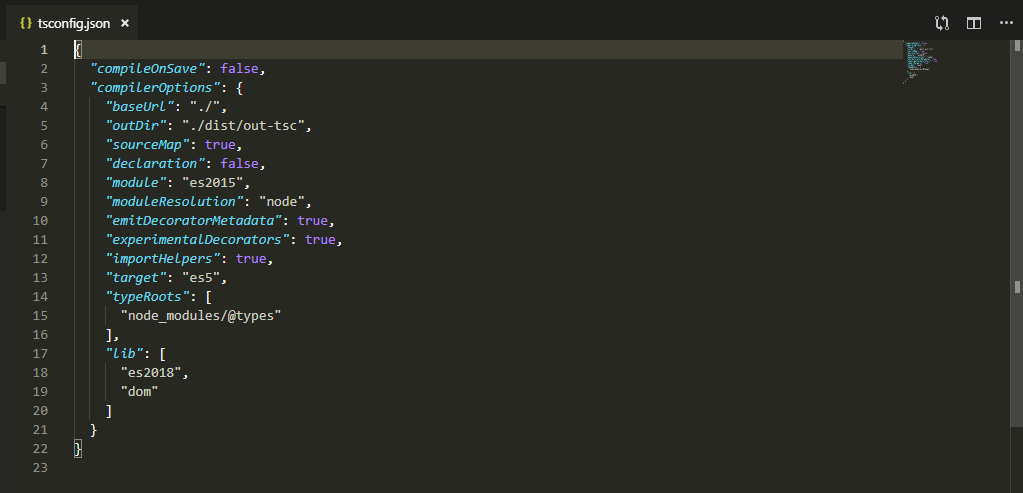
**dependencies –** all package written within are dependent files like jquery dependent for javascript

At present all dependent version is 7 (version 7 is latest at present)



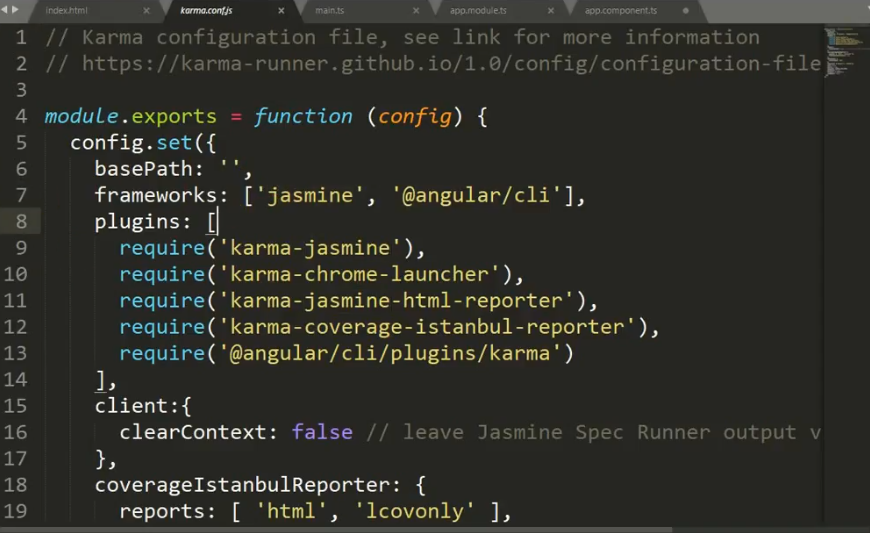
1. tsconfig.json

* This is **typescript configuration** json file
* This is the very important file. Browser only undertand javascript. So we convert typescript into javascript by using **typescript compiler**



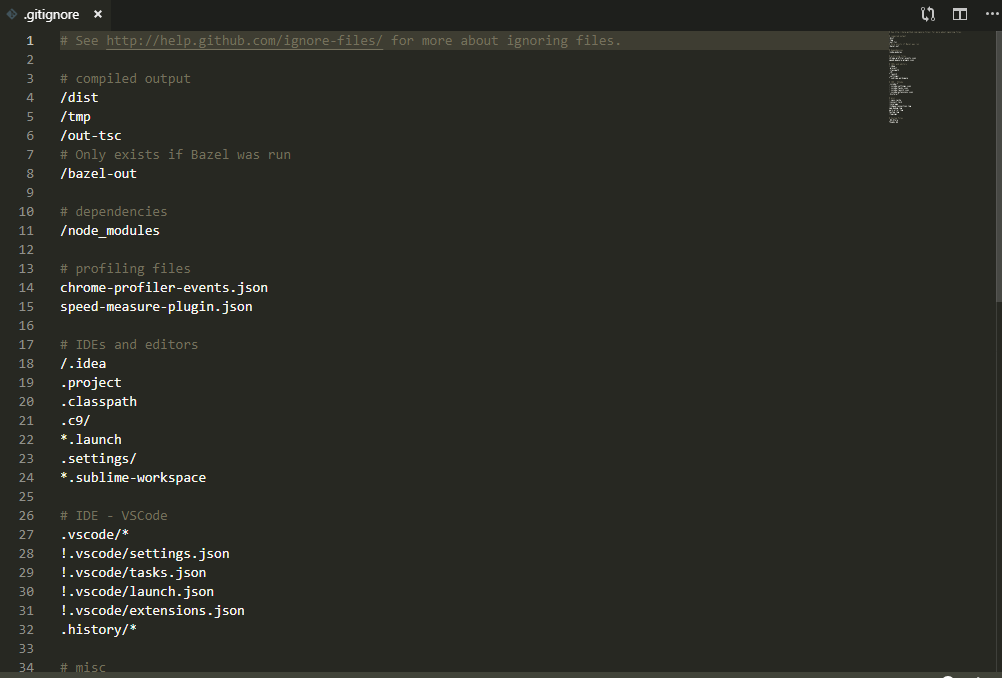
1. karma.conf.js

* Karma is the file used for the testing purpose



1. Gitignore

* Git ignore is a git file which tells us what to ignore at the time of git push

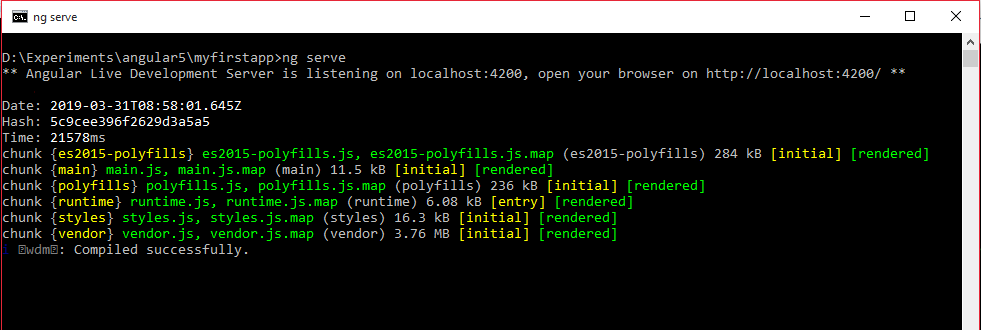


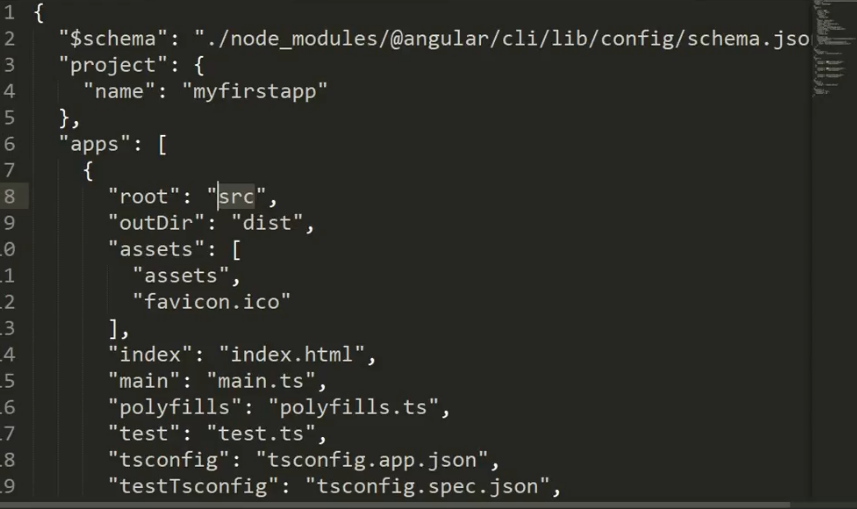
* So ignore all the files listed at the time of upload so to reduce git repository load.

Eg. node\_modules – we don’t need to upload all the package to github. So we can clone from repository and then run **npm install** command as listed in the package.json file

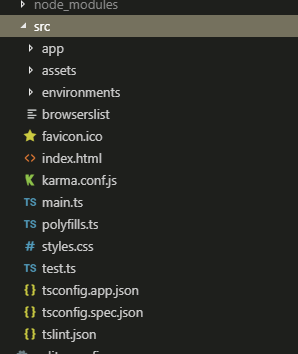
1. angular-cli.json / angular.json

* This is also very important file. So we have run ng serve in commad prompt/terminal which should have been served by now





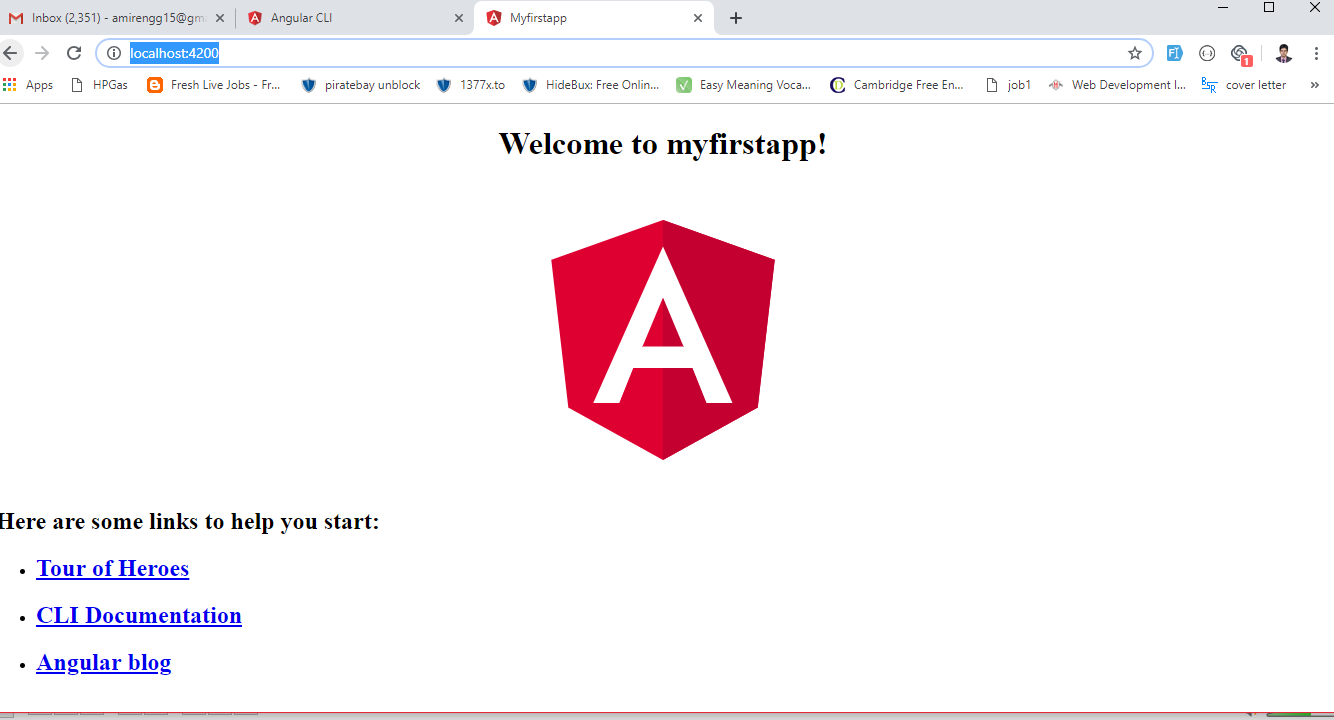
* So the root of our angular project is **src (as we see in the above screenshot)**



1. **assets –** this basically contains all the dependent files like css, js or ts, images etc
2. **index –** main file to run (index.js – inside src folder)

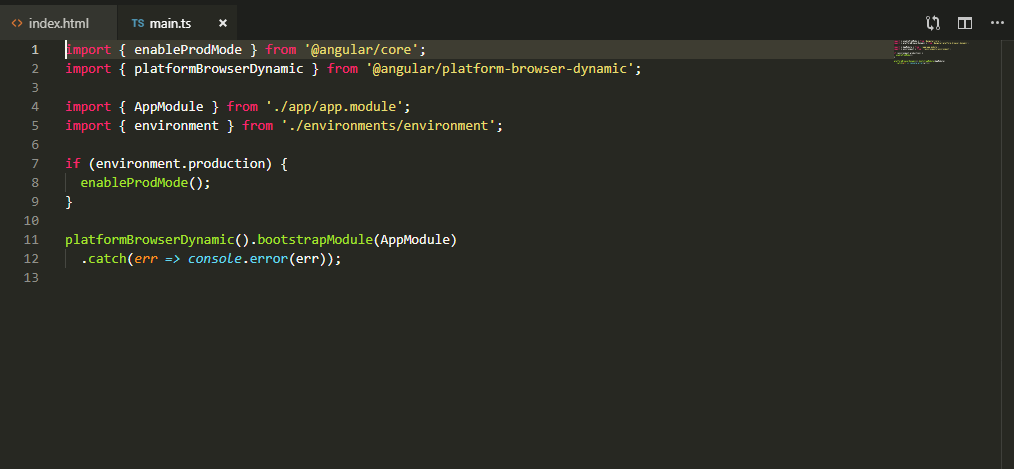


In browser – **localhost:4200**



1. **main.ts (inside src) –** This is the first typescript file which hits from the index.html page written as:

**<app-root></app-root>**

****

So first platformBrowserDynamic runs which runs AppModule which is from **app.module (path – src/app/app.module.ts)**

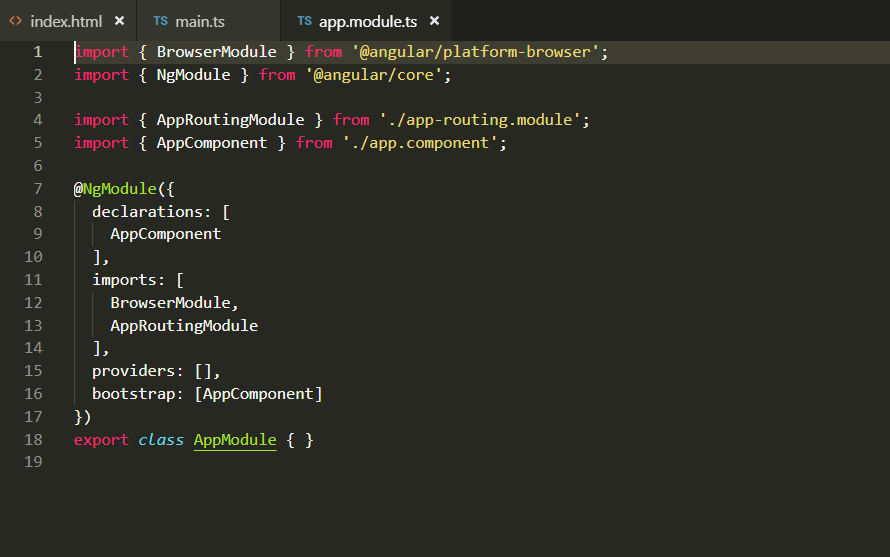
1. **app.module.ts –** It’s path is **project/src/app/app.module.ts**

This another important file where we organize all the components, directive

We have imported some file

**@NgModule** – is the decorator

And finally we export the **AppModule** class so that we will import this file in whatever page I want to use



* Lets return to our **angular-cli.json** file

1. **style.css –** all css files are written here. This is inside src folder

so style.css is written in array in angular-cli.json file. So we can add more css files like bootstrap by adding them in array.



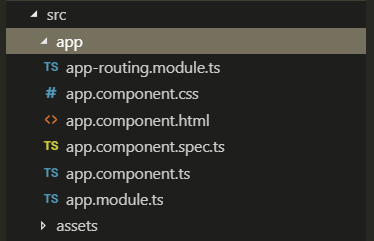
1. node\_modules:

All packages are installed here. The package name is written in package.json file and

**npm install** command will fetch the module here.

1. **App:**

* **95%** of the file we will usually work in this folder where all the components are in **app.module.ts (already discussed above).** Everything is **component** in angular 5
* Here we can create custom tags like **<app-root></app-root>** and thesefiles are also called component

****

1. app.component.css – any css related files
2. app.component.ts- one of the main file



**Line1 – we import Component from @angular/core module**

**Line3 –**

**@Component –** is the decorator

**Selector – app-root // this is what is written in index.html page <app-root></app-root>**

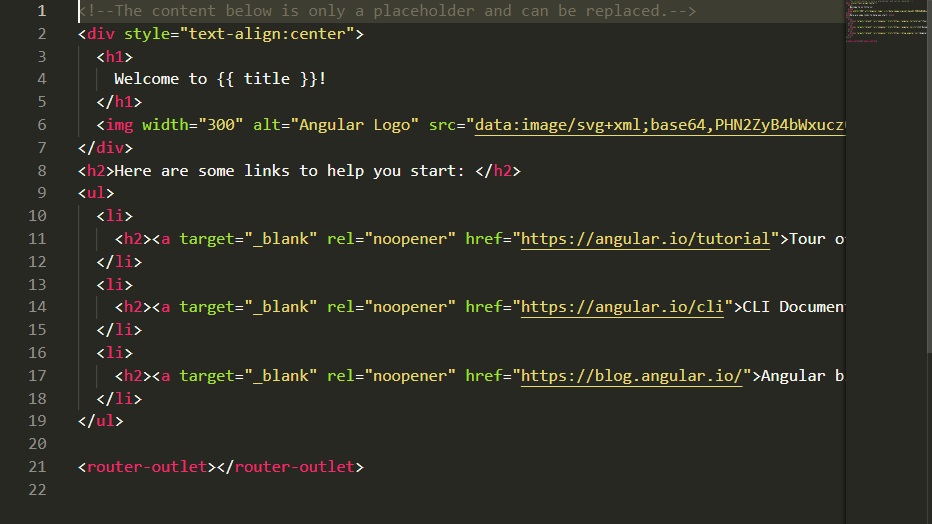
**templateUrl – replace <app-root></app-root> with code written in this page**

eg – templateUrl: ‘./app.component.html’ // the html page in app

**styleUrls – css file connected with this templateUrl**

eg styleUrl: [‘./app.component.css’] // the css page in app

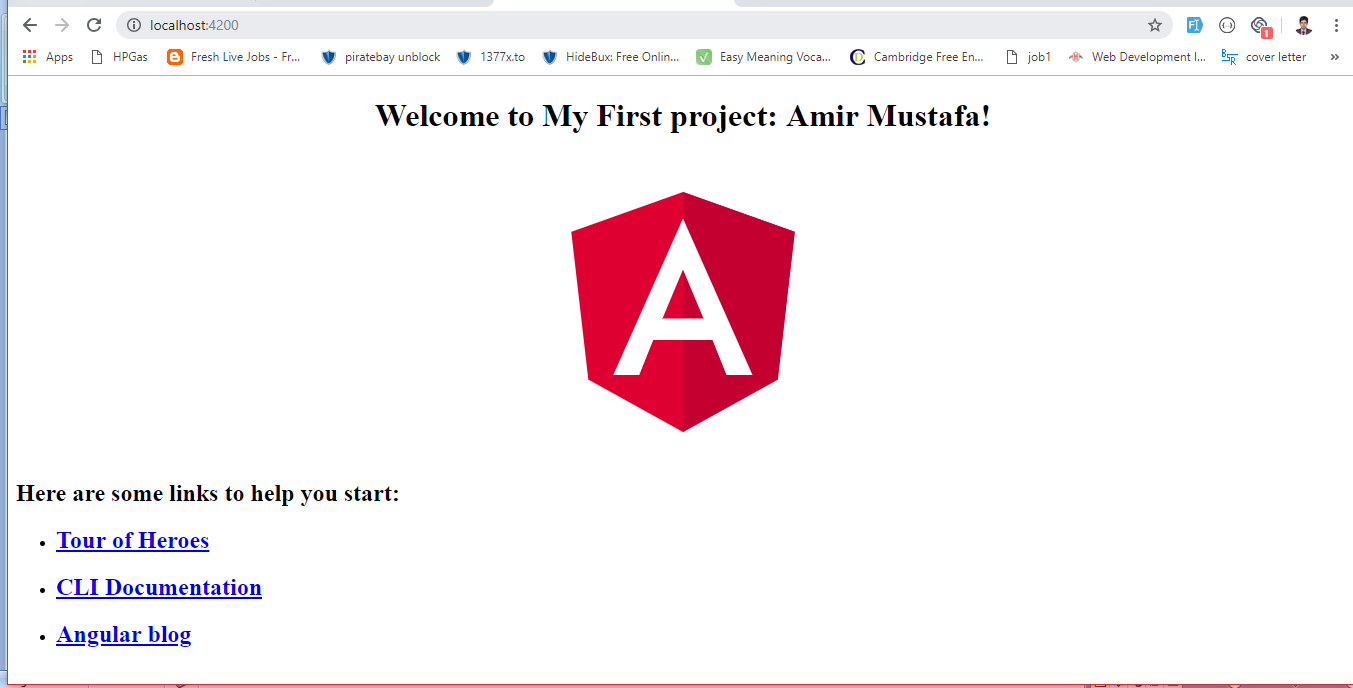
1. app.component.html – The html page connected in point 2



{{ title }} – this is called interpolation module This data is coming from app.component.ts

1. app.component.ts –





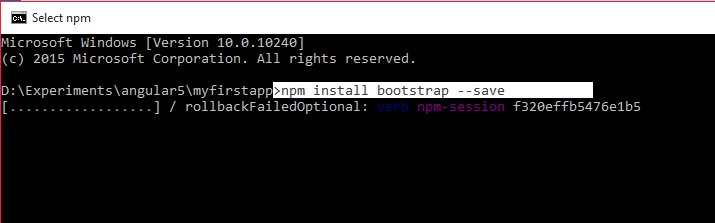
So whatever changes in code and hit save in editor you will do will reflect automatically (without actual browser refresh)

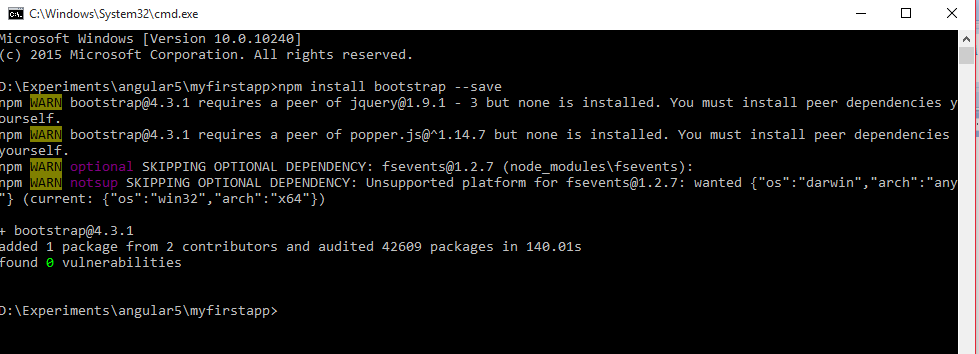
1. app.component.css – The css page connected in point 2

**How to use bootstrap in Angular 5**

1. Go to node\_modules directory and check if bootstrap is installed or not.
2. Open the the project in terminal and type:

**npm install bootstrap –save**





Go to angular.json file in your app root:

Add the path of bootstrap.css in syles array:

*"styles"*: [

"src/styles.css",

"node\_modules/bootstrap/dist/css/bootstrap.min.css"

],

*"scripts"*: [

"node\_modules/bootstrap/dist/js/bootstrap.min.js"

],

Restart angular server i.e. **ng serve**

You can create a button in project/src/app/app.component.html

<div style="text-align:center">

<h1 class="text-danger">

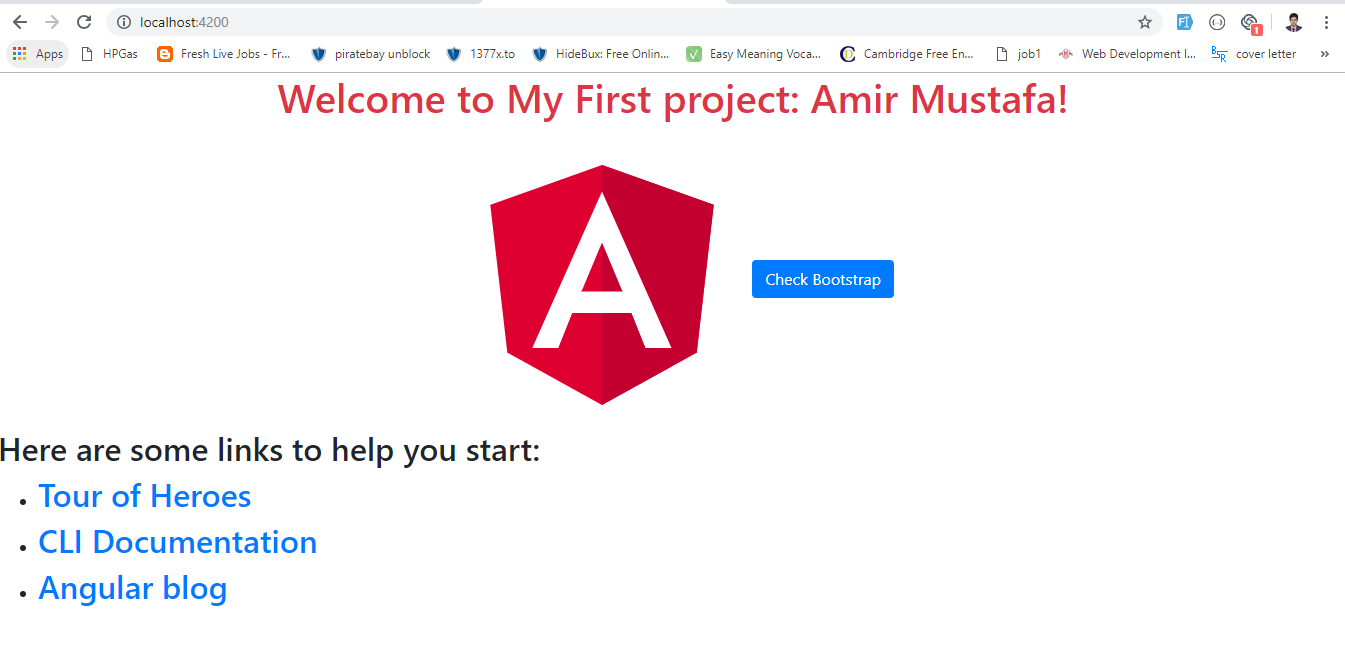
Welcome to {{ title }}!

</h1>

<img width="300" alt="Angular Logo" src="data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAyNTAgMjUwIj4KICAgIDxwYXRoIGZpbGw9IiNERDAwMzEiIGQ9Ik0xMjUgMzBMMzEuOSA2My4ybDE0LjIgMTIzLjFMMTI1IDIzMGw3OC45LTQzLjcgMTQuMi0xMjMuMXoiIC8+CiAgICA8cGF0aCBmaWxsPSIjQzMwMDJGIiBkPSJNMTI1IDMwdjIyLjItLjFWMjMwbDc4LjktNDMuNyAxNC4yLTEyMy4xTDEyNSAzMHoiIC8+CiAgICA8cGF0aCAgZmlsbD0iI0ZGRkZGRiIgZD0iTTEyNSA1Mi4xTDY2LjggMTgyLjZoMjEuN2wxMS43LTI5LjJoNDkuNGwxMS43IDI5LjJIMTgzTDEyNSA1Mi4xem0xNyA4My4zaC0zNGwxNy00MC45IDE3IDQwLjl6IiAvPgogIDwvc3ZnPg==">

<button class='btn btn-default'>Check Bootstrap</button>

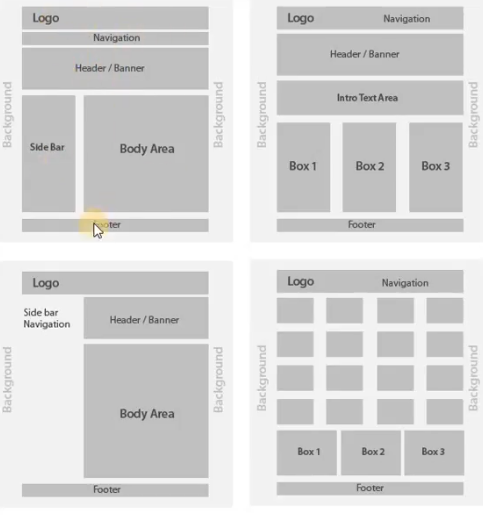
</div>



**Component In Angular 5**

* Whether in Angular 2, 4, 5 or 7 everything is component

Project/src/app – so app is one component created by default. Likewise in a website we create component of header, footer, etc as shown in the screenshot below

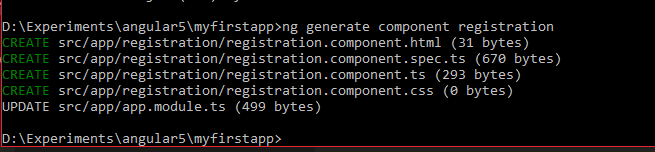


* Angular uses ES7 concept for decorators
* Angular with the help of component decorator tells us the newly created component that is created by us 🡪 its purpose and why is it created
* Another important use of component is reusability. Just call the component and reuse. We know angular with the concept of **Single Page Application (SPA) just written <app-root></app-root> in index.js.** Screenshot above in the left and right little different but Logo, header and Navigation is common so we will call these component.
* There are 2 ways in which we can create the component:

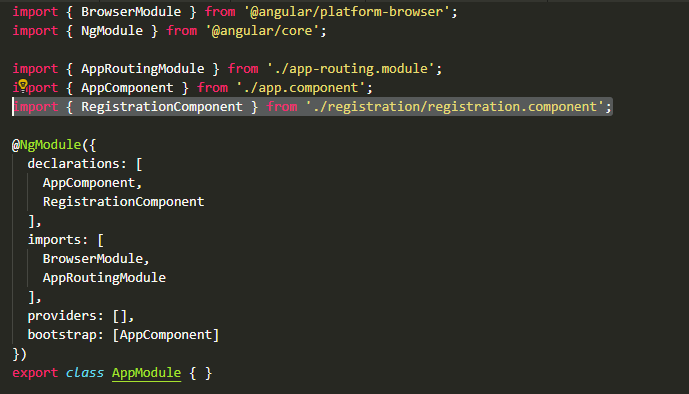
1. Command line
2. Manually
3. Command line:

ng generate component <component\_name> or ng g c <component\_name>

eg ng generate component registration



Component is created inside **src/app** and **module.ts is updated with the new component**

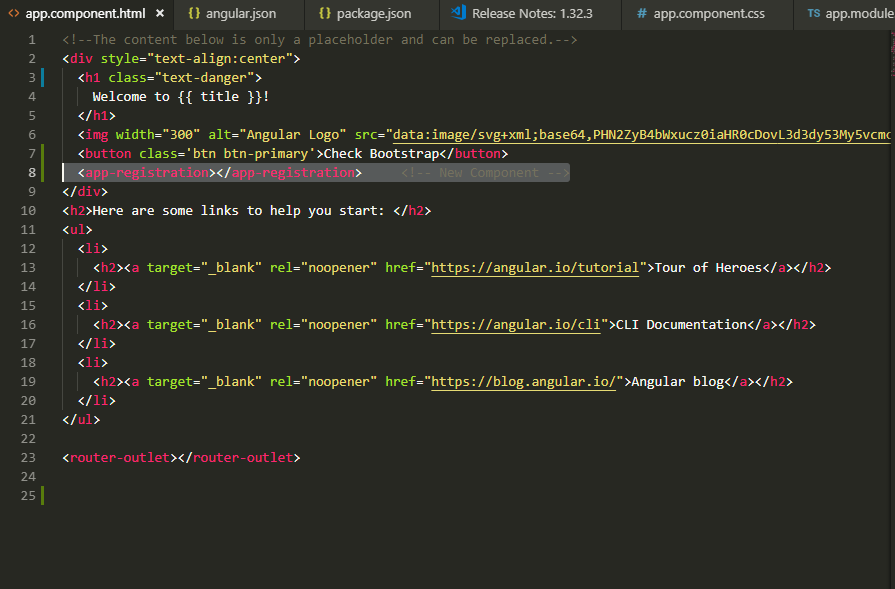


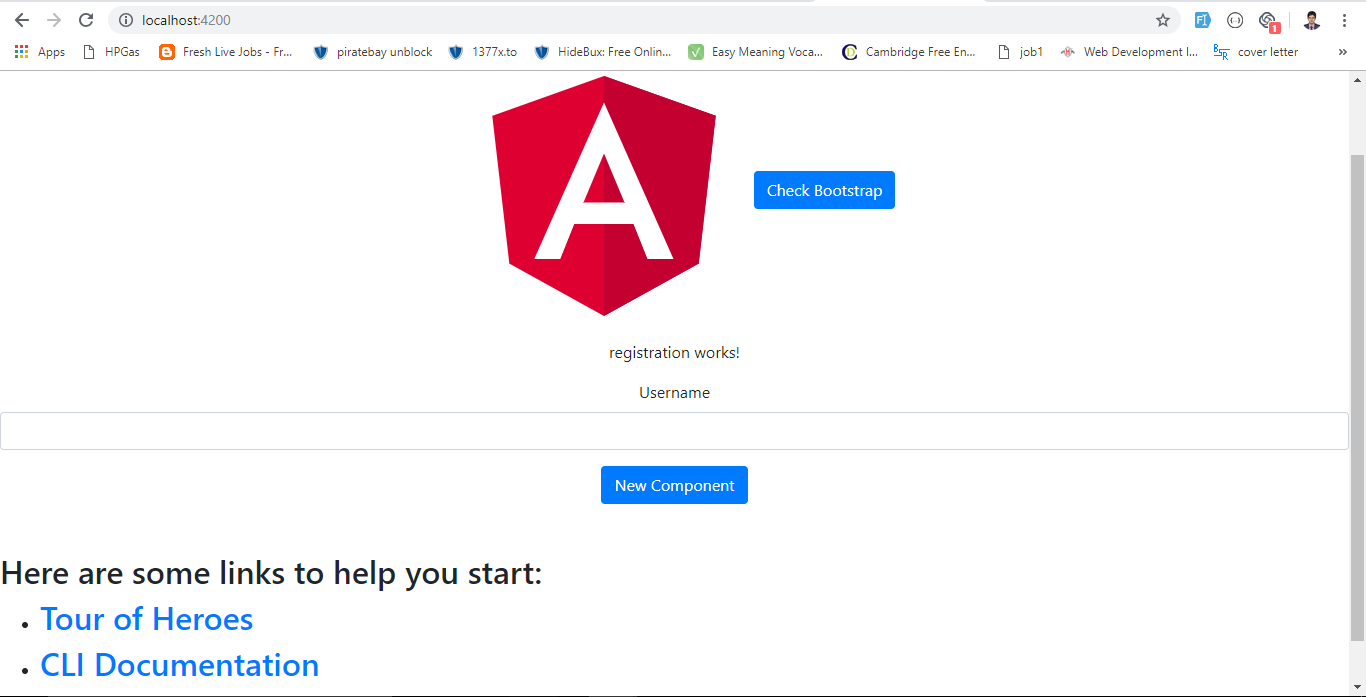
Module.ts file is updated with the component

Now copy and paste the component name i.e. selector. We will use them in as a custom tag in app.component.html

registration.component.ts:

app.component.html:



In browser:

**Interpolation in Angular 5:**

* Whatever we write within {{ }} in html page is used by interpolation i.e. data binding

Eg.

In app.component.ts

export *class* AppComponent {

title = 'My First project: Amir Mustafa';

}

app.component.html

<h1 class="text-danger">

Welcome to {{ title }}!

</h1>

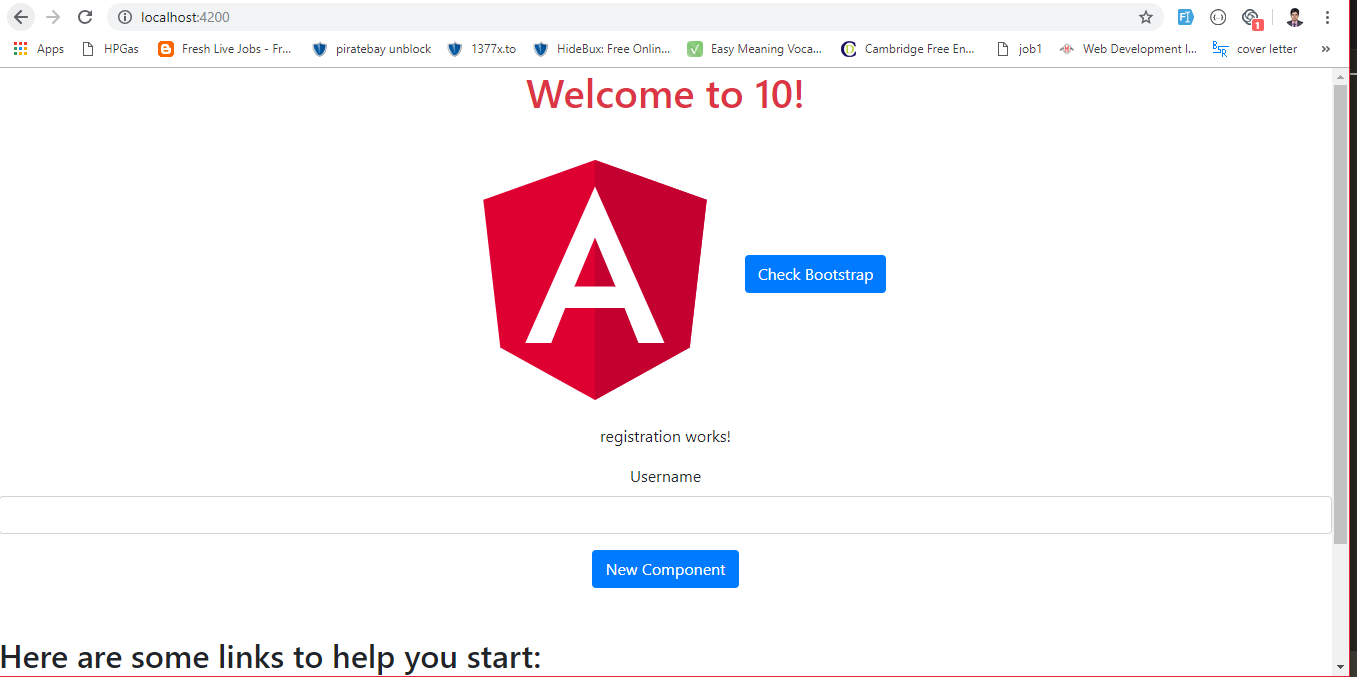
* It is a kind of binding. So data flows from component(app.component.ts) to view page(app.component.html)
* The other use of interpolation is it can solve maths expressions inside {{}}

App.component.html

<h1 class="text-danger">

Welcome to {{ 5 + 5 }}!

</h1>



* {{}} can also solve ternary operation:

Eg1.

App.component.ts

export *class* AppComponent {

title = 'My First project: Amir Mustafa';

video : *string* = 'Akon - Criminal'; // we can define data type by using :

music : *string* = 'Dua Cover | Maryam Waquar'

}

app.component.html

<h1 class="text-danger">

Welcome to {{ title ? video : music }}!

</h1>

O/P:



Eg2:

app.component.ts

export *class* AppComponent {

// title = 'My First project: Amir Mustafa';

video : *string* = 'Akon - Criminal'; // we can define datatype by using :

music : *string* = 'Dua Cover | Maryam Waquar'

}

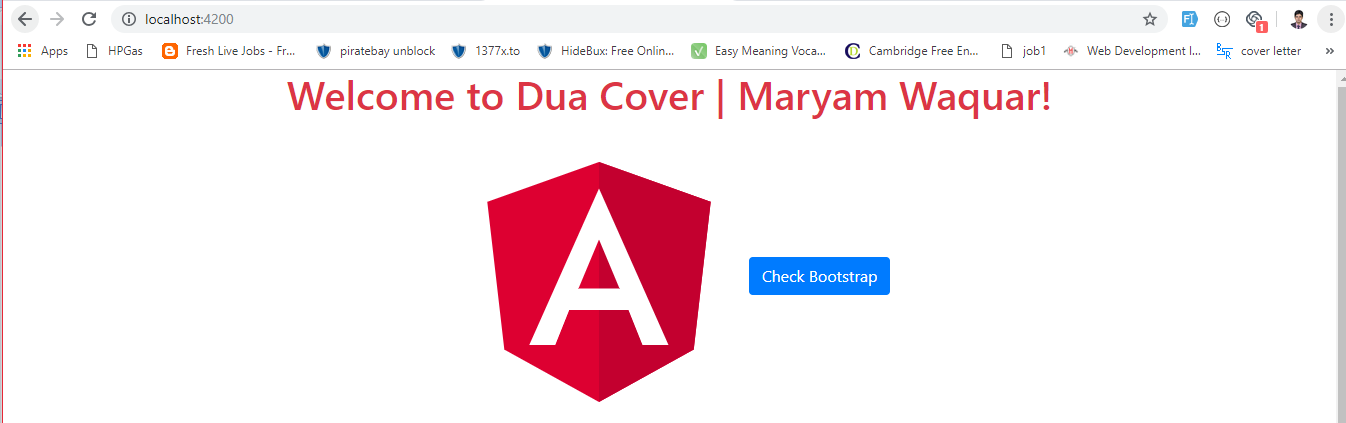
App.component.html

<h1 class="text-danger">

Welcome to {{ title ? video : music }}!

</h1>

O/P:



* We can concatenate two properties inside interpolation:

<h1 class="text-danger">

Welcome to {{ 'video = ' + video + ' & Music = ' + music}}!

</h1>

O/P



* We can use methods / functions for interpolation:

**app.component.ts:**

export *class* AppComponent {

firstName : *string* = ' Amir';

lastName : *string* = ' Mustafa';

youknowmyname() : *string* {

return this.firstName + this.lastName;

}

}

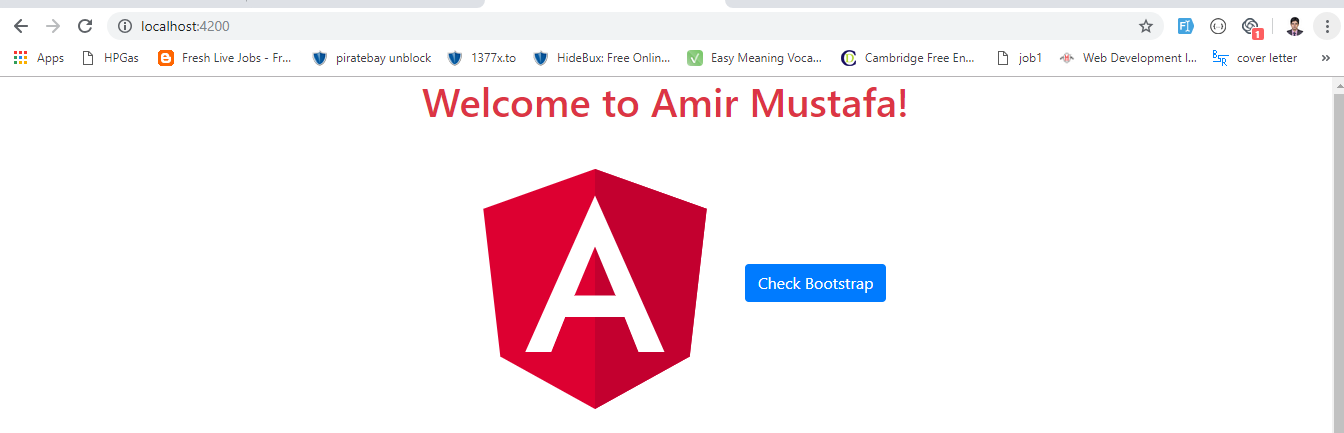
**app.component.html:**

<h1 class="text-danger">

Welcome to {{ youknowmyname()}}!

</h1>

**O/P:**



**Property Binding**

* Interpolation and property binding are almost same. We will see here the differences.
* Property binding deals with binding component class properties to a HTML class properties

i.e. what we write in variable of class in .ts file is access in .html file inside {{ }}

* One way data flows from ts to html

Eg.

app.component.ts

export *class* AppComponent {

// title = 'My First project: Amir Mustafa';

checkProperty : *string* = 'My text';

}

App.component.html

<div style="text-align:center">

<h1 class="text-danger">

Welcome to {{ youknowmyname()}}!

</h1>

<button class='btn btn-primary'>Check Bootstrap</button><br>

<!-- <input type="text" value="{{ checkProperty }}"> --> <!-- With Interpolation -->

<input type="text" [value]="checkProperty"> <!-- With Property Binding -->

</div>

* Difference between interpolation and property binding

|  |  |
| --- | --- |
| Property Binding | Interpolation |
| Deals with string , non strings and all | Deals only with string |

Eg.

app.component.ts

export *class* AppComponent {

// title = 'My First project: Amir Mustafa';

// isHidden : boolean = true;

isHidden : *boolean* = false;

app.component.html

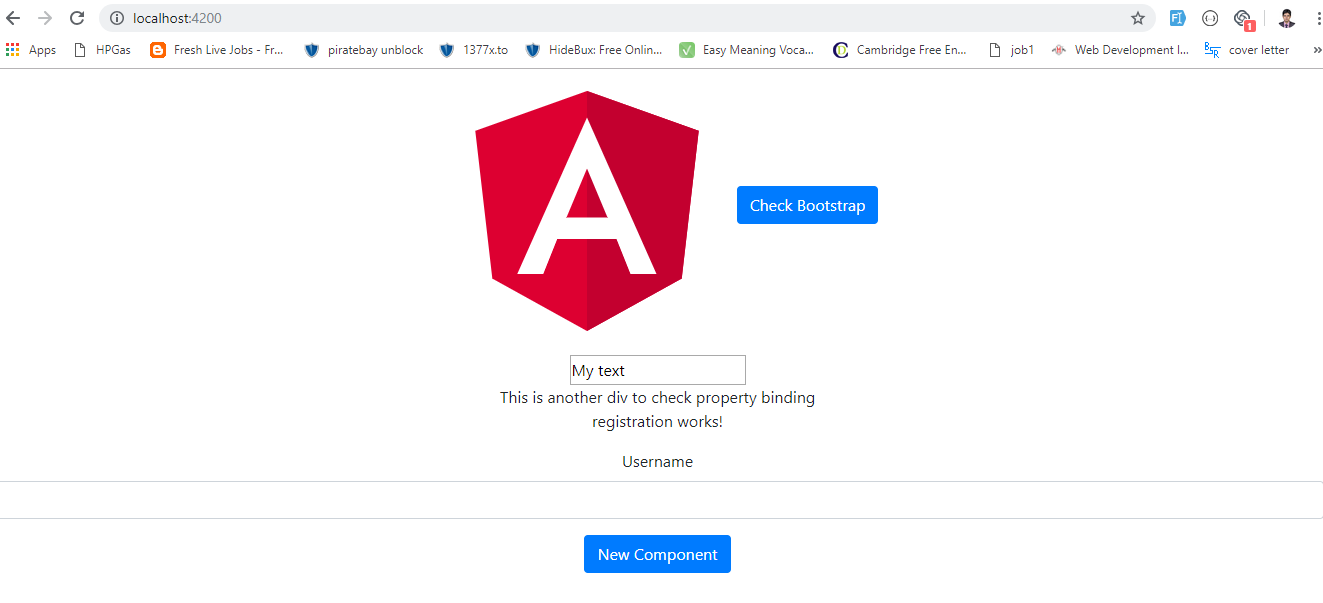
<!-- <div hidden = '{{ isHidden }}'> --> <!-- Interpolation does not work with non string -->

<div [hidden] = 'isHidden'> <!-- Property bindng works with boolean -->

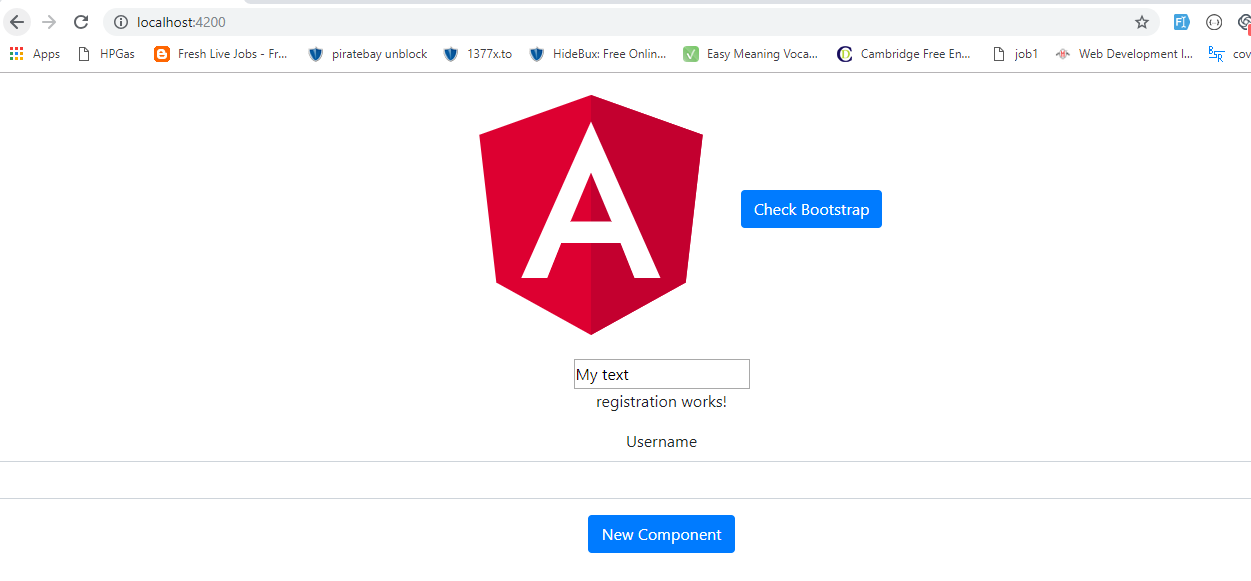
This is another div to check property binding

</div>

With isHidden: true



With is Hidden: false



Also works for button:

<button [disabled] = "isHidden">Test Button</button>

* There is another way to write property binding:

Cannonical format: bind-data

<button bind-disabled = "isHidden">Test Button</button>

When to use - Sometimes there is problem in server side try using this

* There are 3 types of binding: **property binding, event binding and class binding** and in all we can use canonical format.

**Event Binding:**

* **Event binding** means **one way data flow** from the **view (i.e. HTML) to component class (.ts)**

The opp. Of **property binding** (i.e. data flows from **component class to view**)

**Eg.** So we will create an i/p field and a button. On clicking the button i/p data should flow to the component class. This event is just like javascript onclick event but in angular style

app.component.html

<button class="btn btn-primary" (click) = "myFunc()">Event Binding</button><br><br> <!-- We use () for event binding -->

<input type="text" name="myinput" [value]="text"><br><br> <!-- Using Property binding to get new data -->

app.component.ts

export *class* AppComponent {

// title = 'My First project: Amir Mustafa';

// initializing data for Event binding

text : *string* = '';

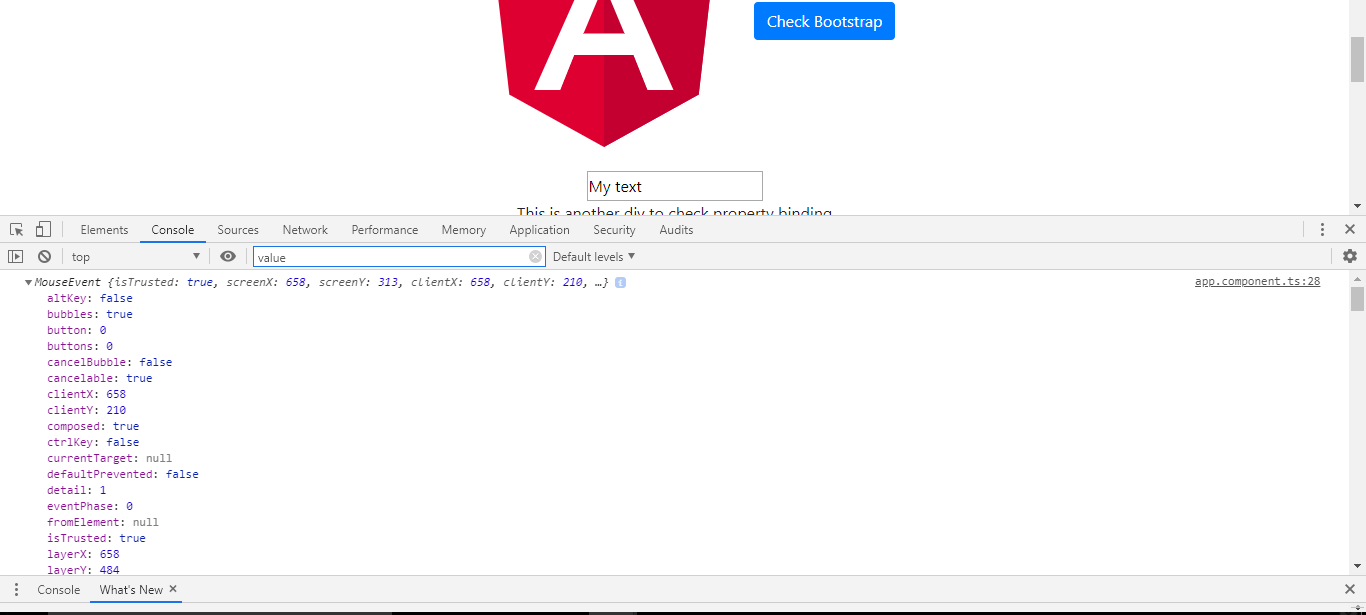
myFunc() {

*console*.log('Amir Rocks !!'); //you can check your console on button click

this.text = "Amir Rocks !!";

}

* So when we click the button it goes to the myFunc() using event binding and data is set to the i/p field using Property binding.
* Now with parameters We will pass **special parameter** in button **$event** andreceivein the function. It has many data like mouse over and the list goes on. It has special parameter value inside target i.e. event.target.valuewherevalue of the button is received



app.component.html

<button class="btn btn-primary" (click) = "myFunc($event)" value="value of button">Event Binding</button><br><br> <!-- We use () for event binding -->

app.component.ts

export *class* AppComponent {

// initializing data for Event binding

text : *string* = '';

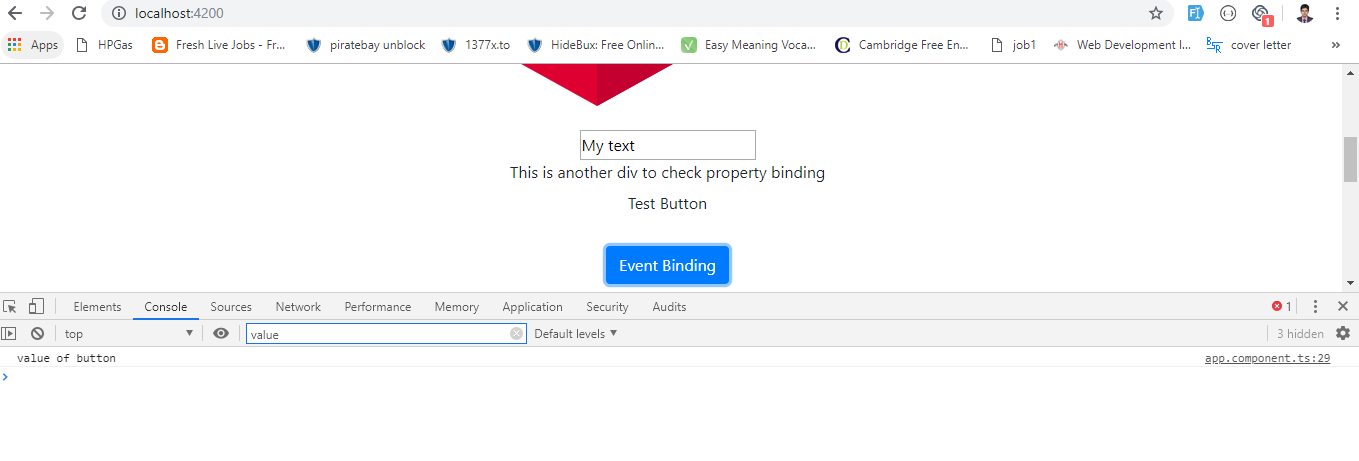
myFunc(*event*) {

// console.log(event);

*console*.log(event.target.value);

}

}



**Class Binding**

* Class Binding is a variation to the property binding.
* Suppose you want to apply group of property like active / inactive to the group of class we can use in this way

**Eg.**

So the property will be applied to all the group having [class.active] = "isActive"

app.component.html

<!-- Class Binding -->

<button class="btn btn-danger" [class.active] = "isActive">active class</button>

<button class="btn btn-danger" [class.active] = "isActive">active class</button>

<button class="btn btn-danger" [class.active] = "isActive">active class</button>

<button class="btn btn-danger" [class.active] = "isActive">active class</button>

app.component.ts

export *class* AppComponent {

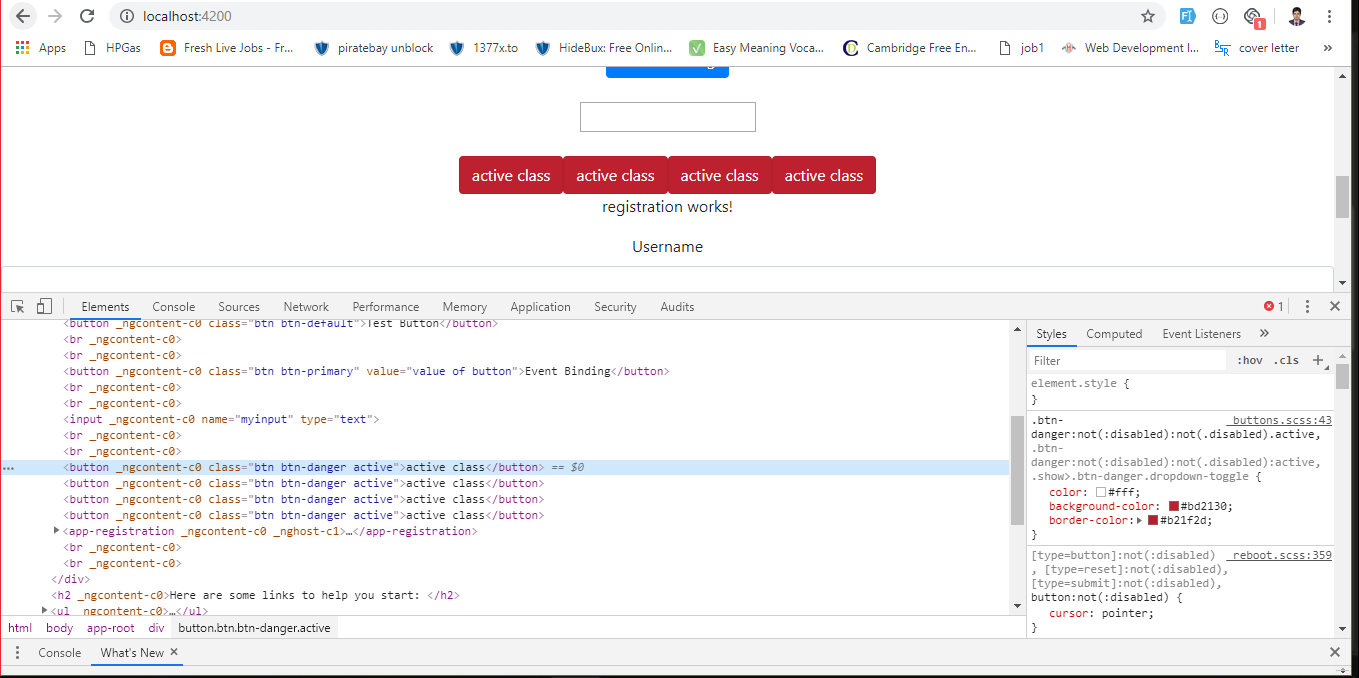
// Class Binding

isActive: *boolean* = true; // all the class will be active will be removed

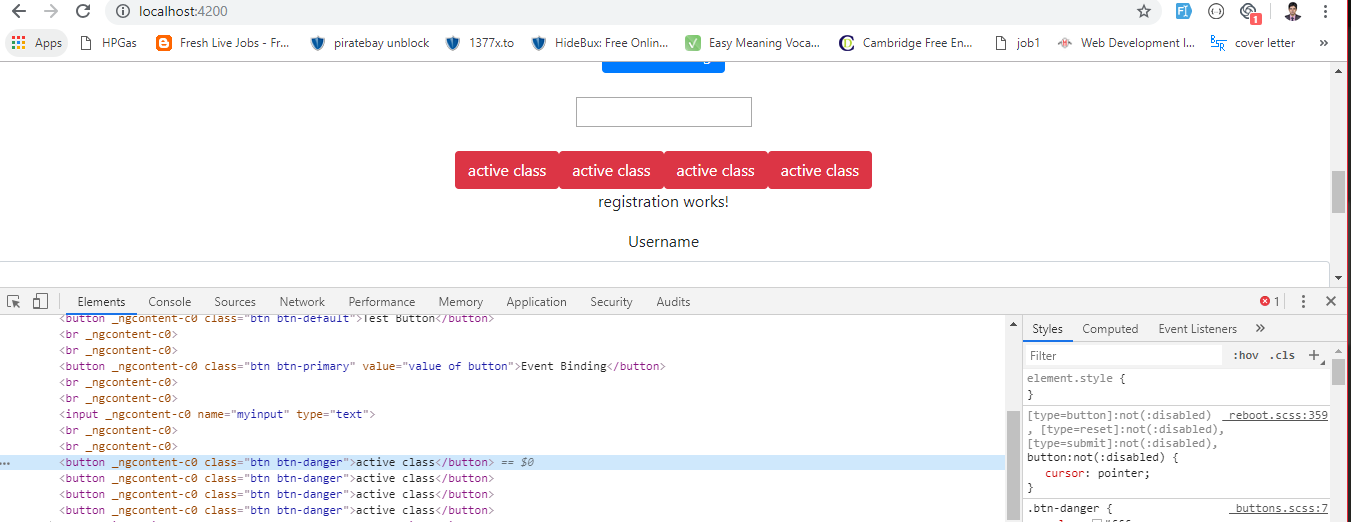
// isActive: *boolean* = false; // all the class will be active having class.active = isActive

}

When isActive is true



When isActive is false



* You can also apply disabled in similar way

App.component.html

<button class="btn btn-danger" [class.disabled] = "isDis">active class</button>

App.component.ts

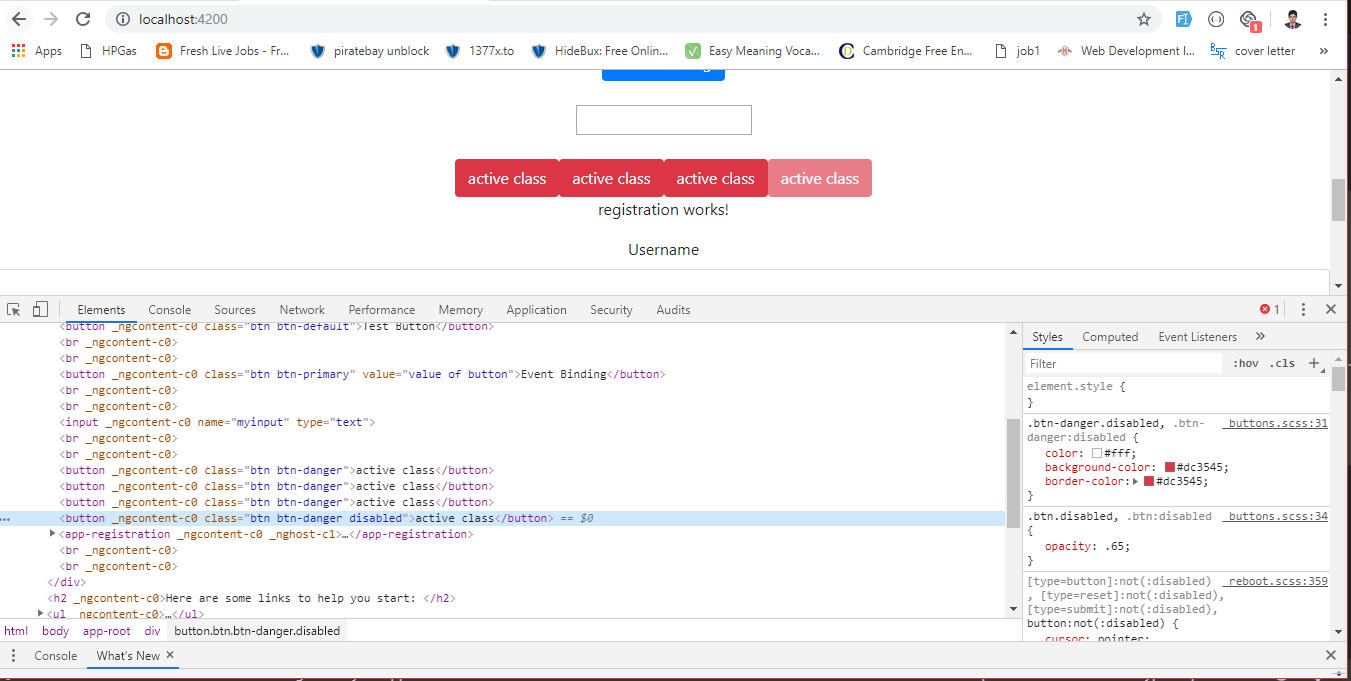
export *class* AppComponent {

// Class Binding

isActive: *boolean* = false;

isDis: *boolean* = true;

}



CSS Class Binding:

* This is equivalent to inline css we used to write in any element like button or div

<button class=’btn btn-primary’ sytle=”background: black; color: white”>

* We will use Angular to write this

app.component.ts

<!-- INLINE CSS IN CLASS BINDING -->

<!-- <button class='btn btn-primary' style='color:white; background: black;'>Multi CSS Button</button> -->

<button class='btn btn-primary' [class] = 'multiCssClasses'>Multi CSS Button</button>

app.component.ts

export *class* AppComponent {

// multiCssClasses

multiCssClasses: *string* = 'colorChange bgColor' // these are classes name which we will apply in app.component.css file

}

app.component.css

.colorChange {

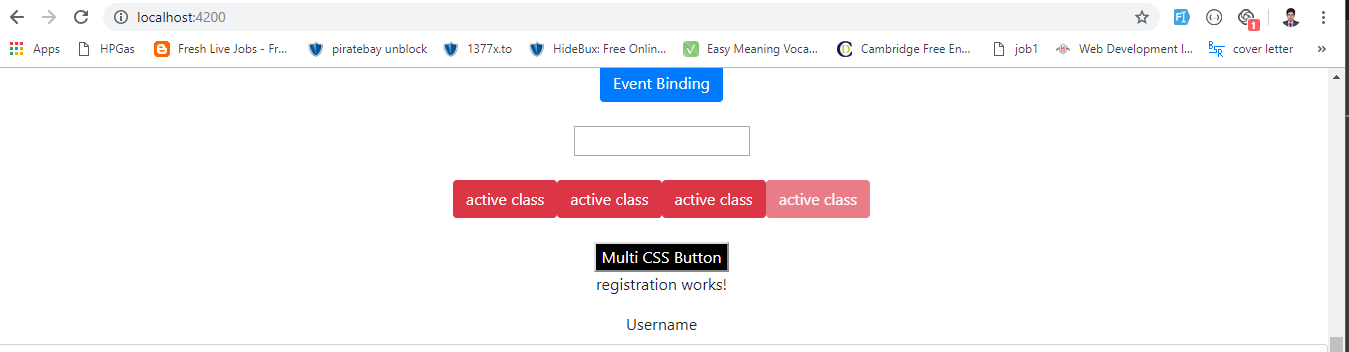
*color*: white;

}

.bgColor {

*background*: black

}



* So this is the way we can add css. There another better way to add classe using ngClass. We will see that soon.

Style Binding:

* This is another way of writing css inline css:

Eg:

app.component.html

<!-- STYLE BINDING -->

<button [style.color] = 'inlineColor'> Inline Style</button>

app.component.ts

export *class* AppComponent {

// Style Binding

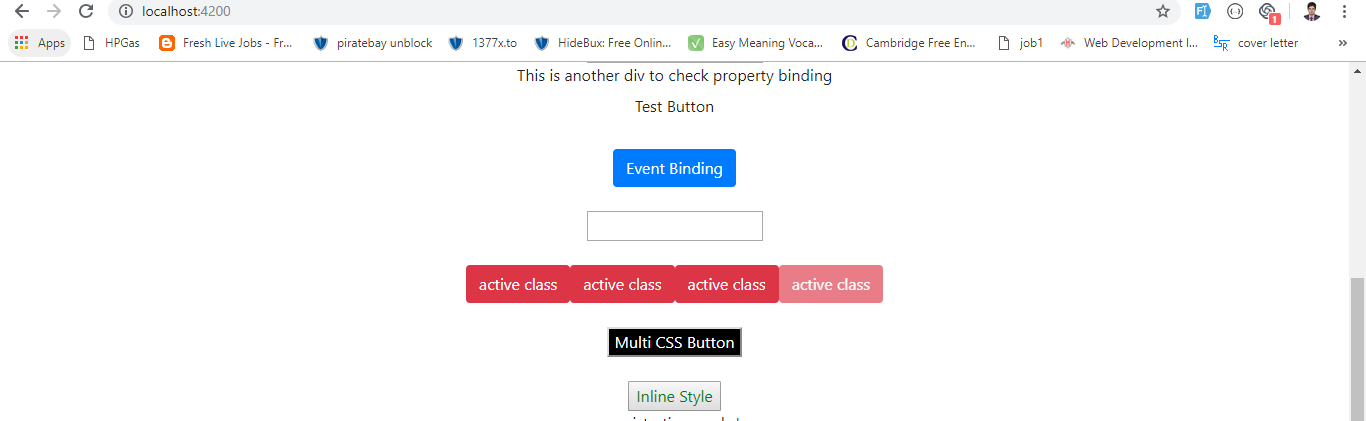
inlineColor: *string* = 'green'

}

Note:

[style.color] – this color is DOM object property and not css

<https://www.w3schools.com/jsref/dom_obj_all.asp>



* We can also apply ternary operator here:

app.component.html

<button [style.color] = "inlineColor2 ? 'red' : 'green'" > Inline Style</button>

app.component.ts

export *class* AppComponent {

inlineColor2: *boolean* = false; // show green

// inlineColor2: *boolean* = true; // show red

}

* For css we write: font-weight: bold.

So we can write either style.font-weight or style.fontWeight both are acceptable

<button [style.font-weight] = "inlineColor2 ? 'bold' : 'normal'" > Font style</button>

Or

<button [style.fontWeight] = "inlineColor2 ? 'bold' : 'normal'" > Font style</button>

11. Template Reference Variable

🡪 Template variable is another form of event binding i.e. we send data from app.component.html to app.component.ts

Lets see what we have done earlier first:

App.component.html

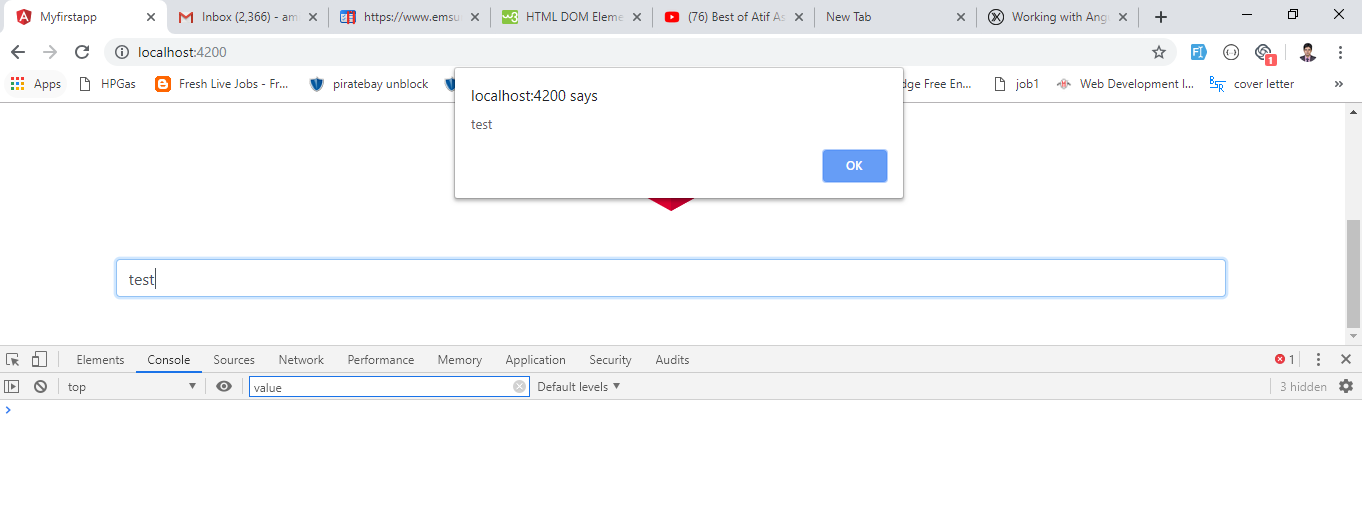
<input type="text" name="" class="form-control" (click) = "getEmail($event)">

App.component.ts

getEmail(*event*) {

alert(event.target.value);

}



* For the template variable we just write #<key> 🡪 here you have all access to $events.target

So pass like email.value 🡪 you have value here.

Eg.

app.component.html

<input type="text" name="" class="form-control" #email (click) = "getEmail(email.value)">

app.component.ts

export *class* AppComponent {

//// Template Reference

/\* getEmail(event) {

alert(event.target.value);

} \*/

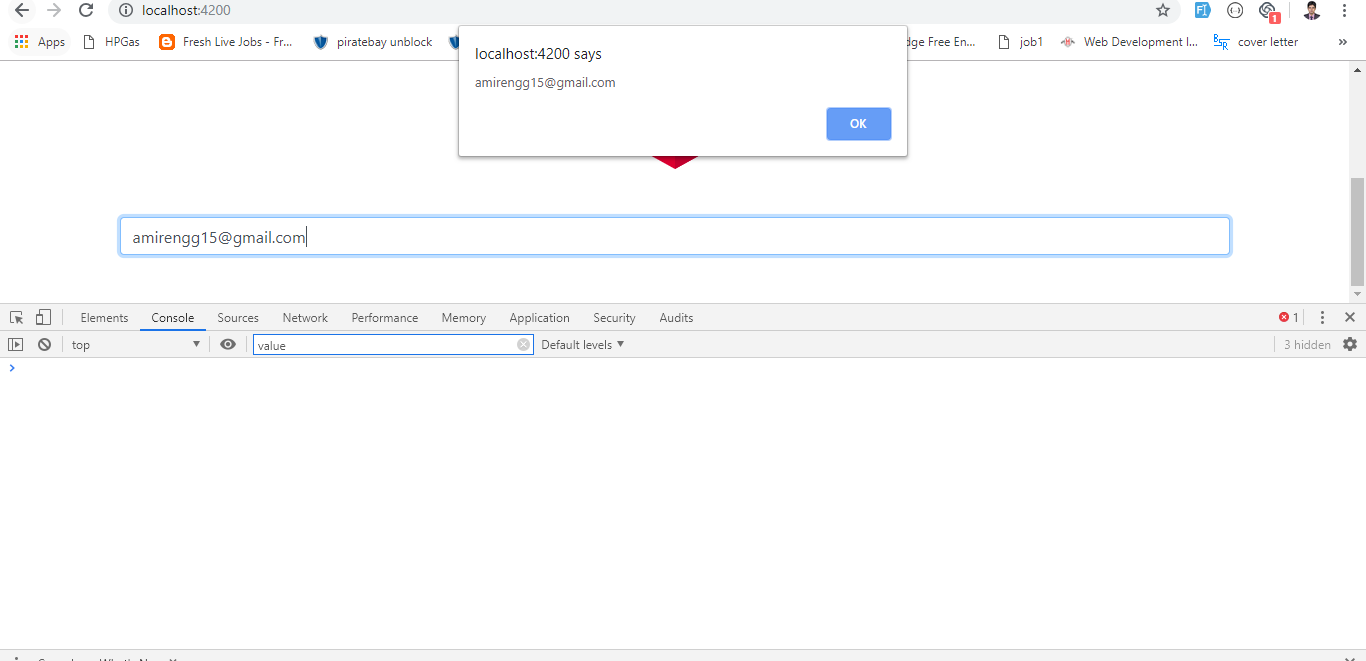
getEmail(*email*) {

// console.log(email);

alert(email);

}

}



Eg2: passing reference of input to the button

.html

<input type="text" name="" class="form-control" #email>

<button (click) = "getEmail(email.value)">Click Me</button>

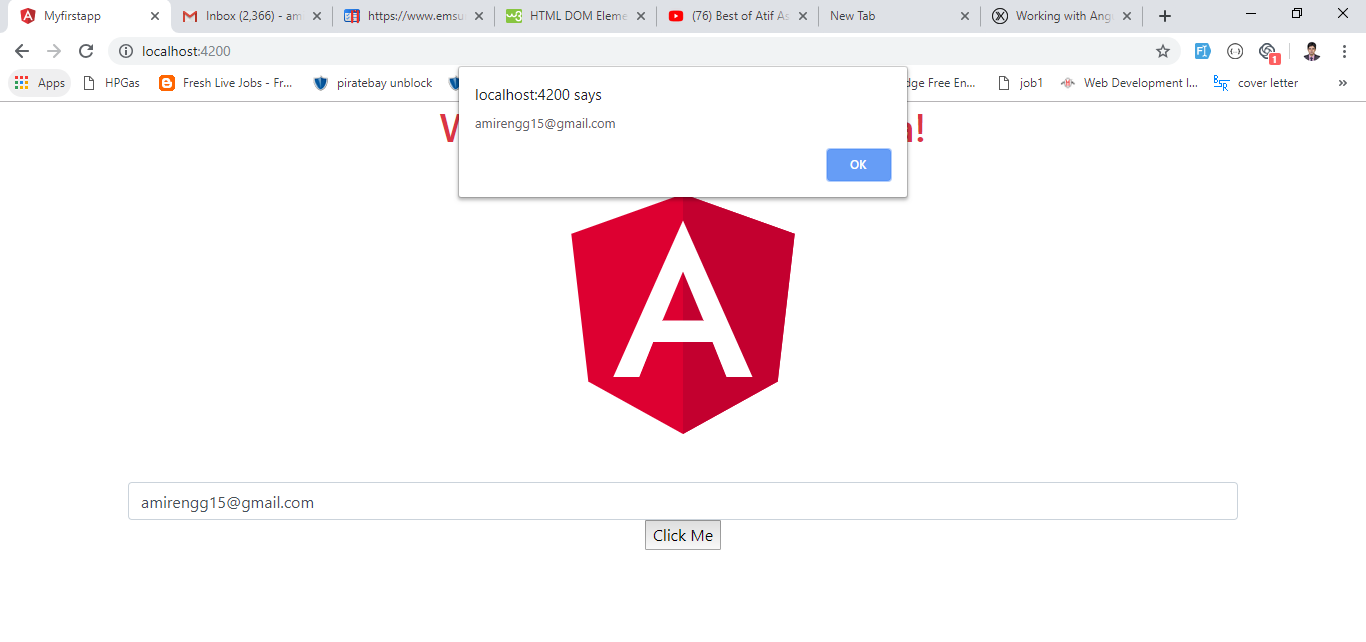
.ts

getEmail(*email*) {

// console.log(email);

alert(email);

}



12. Two Way Data Binding in Angular 5

🡪 Here we will see how data flows from component to view and back from view to component

* Angular have given us built in directive to deal with two way data binding = ngModel
* Anything with **ng prefix** is a directive
* TRICK – way to write a directive [()] – banana in a box

Eg

app.component.html

<input [(ngModel)]='email' (click) = getEmail()>

app.component.ts

export *class* AppComponent {

email = 'amirengg15@gmail.com'; // default from component to view

getEmail() {

*console*.log(this.email); // receive data from view to component

}

}

app.module.ts: you must include forms module here for ngModel to work

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

// For using ngModel - two way binding we need this module

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

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { RegistrationComponent } from './registration/registration.component';

@NgModule({

declarations: [

AppComponent,

RegistrationComponent

],

imports: [

BrowserModule,

AppRoutingModule,

FormsModule // Using FormsModule included above

],

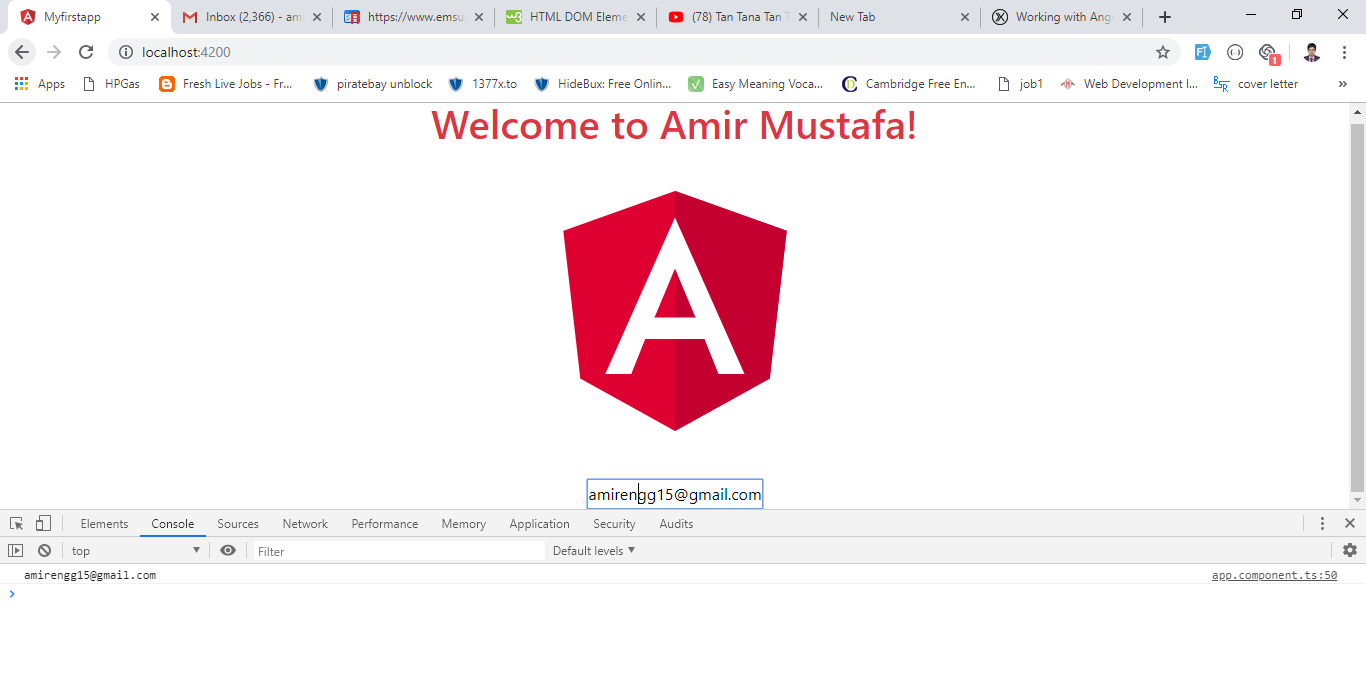
providers: [],

bootstrap: [AppComponent]

})

export *class* AppModule { }

Now first when default value of email is coming from component to view. When we click it we will see its value in console.



When we change in i/p and click it you will see updated value in console. Hence two way binding



13. Pipe in Angular 5

🡪 Pipes are basically used to format the data at the run time

Eg.

1. LowerCasePipe
2. UpperCasePipe
3. DecimalPipe
4. CurrencyPipe
5. DatePipe

* Pipes takes data as an input and transform it into desired output

Eg1. Pipe is written after | in interpolation like uppercase, lowercase, date etc

{{ }}

app.component.html

<h2 class="text-danger">

My name is {{ biodata.name | uppercase }}. I was born in {{ biodata.born | date }}. I am {{ biodata.age }} years old. I got {{ biodata.board }}

percent in 12th class. My salary expectation is {{ biodata.salary }}.

</h2>

App.component.ts

export *class* AppComponent {

// Pipes in Angular 5

biodata = { // object

name: "Amir Mustafa",

age: 26,

boards: 70,

salary: 0.001,

born: new Date(1992, 10, 1)

}

}

Without pipe:



With Pipe:



Some other references with date and its different format:

<https://angular.io/api/common/DatePipe>

<https://angular.io/api/core/Pipe>

Eg {{ biodata.born | date: ‘shortdate’}} // 11/1/92

{{ biodata.boards | number: ‘3.1-1’ }} before decimal 3, after decimal min 1, max 1 //069.9

{{ biodata.boards | currency}} // $0.00

{{ biodata.boards | currency : ‘AUD’}} // A$ 0.00

{{ biodata.boards | currency : ‘AUD’ : ‘code’}} // AUD 0.00

{{ biodata.boards | currency : ‘INR’}} // Rs0.00

14. NgClass Directive

🡪 This is Attribute Directive

* An Attribute directive changes the appearance or behavior of an element, component or another directive for eg. built in NgStyle or NgClass changed several element styles at the same time.

🡪 We should use NgClass when we are expecting multiple classes to be potentially added.

* Just add [ngClass] = “’ <your\_class\_name> ’” and define that class in app.component.css file

Eg

app.component.html

<!-- ngClass -->

<div [ngClass] = "' colorChange bgColor singleCssColorChange '">

ngClass applied. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Architecto accusamus

autem nisi officiis veniam eaque voluptates perferendis labore exercitationem maxime maiores fugit

unde eos corrupti amet esse nihil, fuga ullam.

</div>

app.component.css

.colorChange {

color: white;

}

.bgColor {

background: black

}

.singleCssColorChange {

font-style: italic;

}



* If there are more than many classes to add inside ngClass, adding them will become messy. We will create object in ts file and add here

App.component.html

<!-- ngClass -->

<!-- <div [ngClass] = "' colorChange bgColor singleCssColorChange '"> -->

<div [ngClass] = " multiCssClass ">

ngClass applied. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Architecto accusamus

autem nisi officiis veniam eaque voluptates perferendis labore exercitationem maxime maiores fugit

unde eos corrupti amet esse nihil, fuga ullam.

</div>

App.component.ts

export class AppComponent {

// Adding class for multiple css for ngClass

multiCssClass = {

'colorChange': true,

'bgColor': true,

'singleCssColorChange': true

}

}

And css remain the same

o/p will be the same and code will be much cleaner.

Various others way to add css are <https://angular.io/api/common/NgClass>

15. ngStyle

🡪 This is also Attribute Directive.

* This is similar to ngClass and shorter way of style binding
* ngClass is equivalent to giving a class and writing its css properties in class
* ngStyle is equivalent to inline css i.e. writing css properties there without giving class

eg <input type=”text” style=”color:red; background:yellow”>

eg.

app.component.html

<!-- ngStyle -->

<div [ngStyle] = " {

'color' : 'red',

'background' : 'yellow'

} ">

ngStyle applied. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Architecto accusamus

autem nisi officiis veniam eaque voluptates perferendis labore exercitationem maxime maiores fugit

unde eos corrupti amet esse nihil, fuga ullam.

</div>



* For cleaner code we will create object in ts file write that property here

app.component.html

<!-- ngStyle -->

<div [ngStyle] = " inlineCSSStyle ">

ngStyle applied. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Architecto accusamus

autem nisi officiis veniam eaque voluptates perferendis labore exercitationem maxime maiores fugit

unde eos corrupti amet esse nihil, fuga ullam.

</div>

app.component.ts

export class AppComponent {

// Adding class for ngStyle

inlineCSSStyle = {

'color': 'red',

'background': 'pink'

}

}



16. NgIf

🡪 This is Structural Directive

* Structural Directives are responsible for HTML layout. They shape or reshape the DOM’s structure, typically by adding, removing or manipulating elements.
* We use \*ngIf = “a === ‘xyz’”. For structural we use \*

Eg.

app.component.ts

export class AppComponent {

myName : string = 'Amir Mustafa';

}

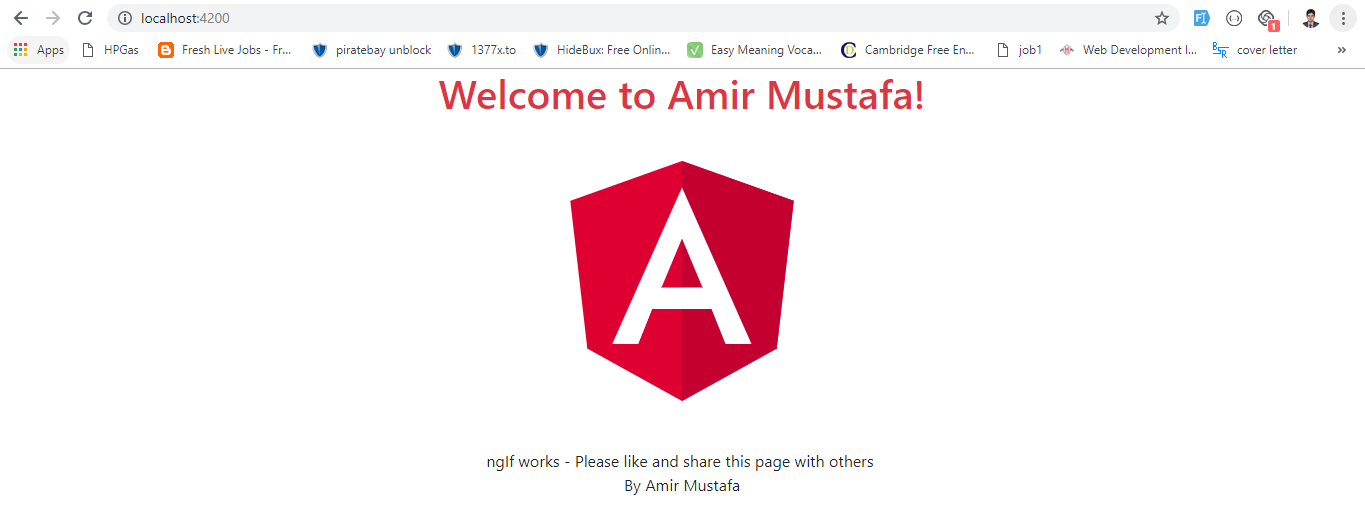
App.component.html

<!-- ngIF -->

<div \*ngIf = " myName === 'Amir Mustafa' ">

ngIf works - Please like and share this page with others

</div>



* If you will write wrong text, this div will be gone.
* Way to write else: we will create a another <ng-template> with its reference variable and will initialize in the reference of the previous div

app.component.html

<!-- ngIF -->

<div \*ngIf = " myName === 'Amir Mustafa'; then mainpart else elsepart " >

</div>

<ng-template #mainpart> <!-- Initializing this with the reference variable and will use this in if of above div -->

ngIf works - Please like and share this page with others

<br>

By {{myName}}

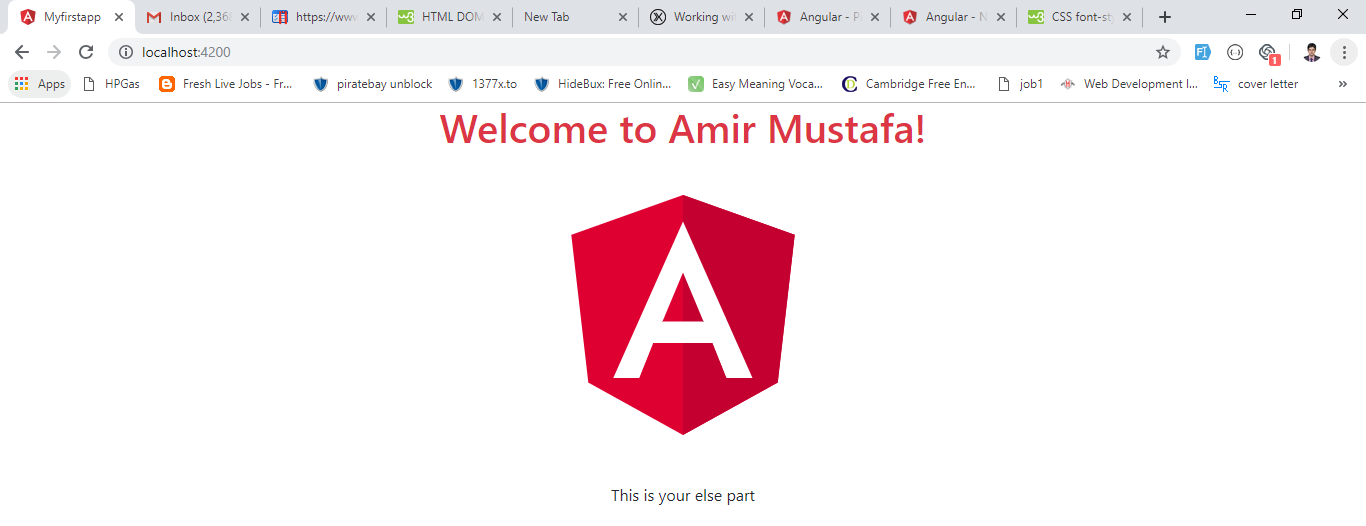
</ng-template>

<ng-template #elsepart> <!-- Initializing this with the reference variable and will use this in else of above div -->

This is your else part

</ng-template>

App.component.ts will be same as above



Note: writing multiple ng-template dosen’t harm the size as it will be in code only when called otherwise will not be there in webpage code.

17. NgSwitch

🡪 Outer Switch is **Attribute Directive** i.e. [ngSwitch = ‘favColor’]

* Inner case comparison is **Structural Directive** i.e. \*ngSwitchCase = “ ‘red’ ”
* For default case is **Structural Directive** i.e. \*ngSwitchDirective

Eg.

App.component.html

<!-- ngSwitch -->

<div [ngSwitch] = "favColor">

<div \*ngSwitchCase = " 'red' " > This is switch case for {{favColor}}</div>

<div \*ngSwitchCase = " 'green' " > This is switch case for {{favColor}}</div>

<div \*ngSwitchCase = " 'blue' " > This is switch case for {{favColor}}</div>

// Default case

<div \*ngSwitchDefault> This is the default case</div>

</div>

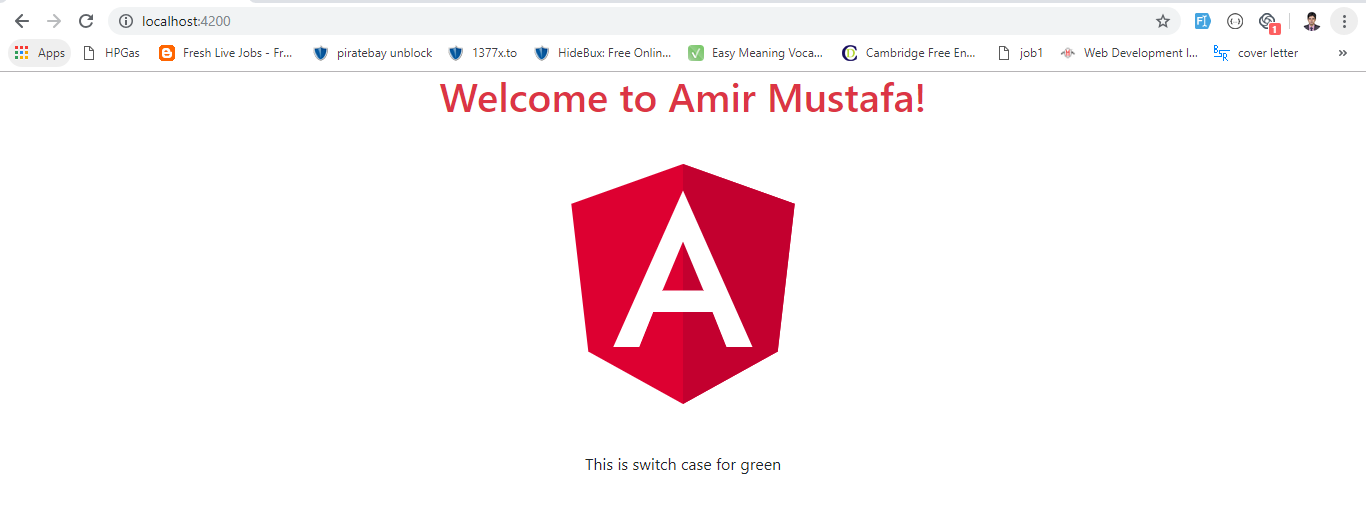
App.component.ts

export class AppComponent {

favColor = 'green'; // note default datatype is any

}

O/P



18. NgFor

🡪 Consider this as the foreach loop of PHP. If there are list in array or object this will show them one by one.

🡪 NgFor is a structural directive i.e. use \*ngFor

Eg.

app.component.ts

export class AppComponent {

// using for ngFor

colors = ['Red', 'Green', 'Blue']; // array of colors

}

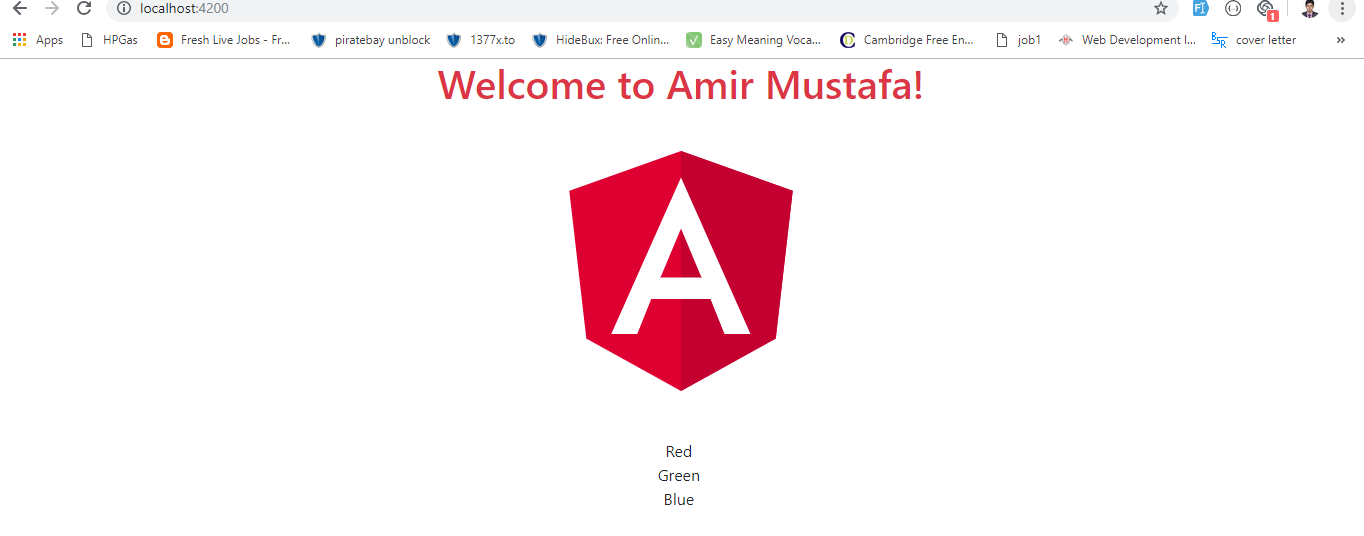
App.component.html

<div \*ngFor = "let col of colors">

{{ col }} <br>

</div>

O/P



Eg2:

app.component.ts

export class AppComponent {

biodatas = [

{'id': 1, 'name': 'Amir'},

{'id': 2, 'name': 'Loves'},

{'id': 3, 'name': 'Workout'},

];

}

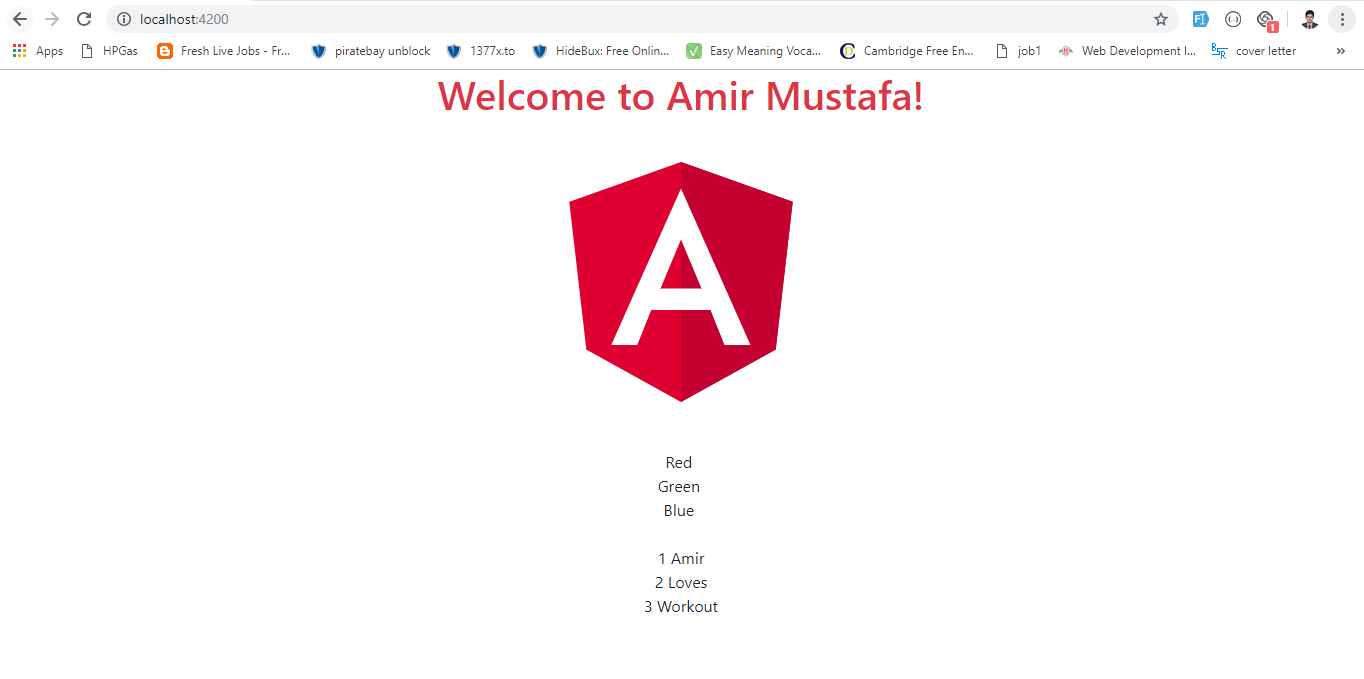
App.component.html

<div \*ngFor = "let bio of biodatas">

{{bio.id}} {{bio.name}}

</div>

O/p



🡪 By default angular also provides with the index property in ngFor which starts from 0, 1, ..

Eg 3

App.component.html

<!-- with index of ngFor -->

<div \*ngFor = "let bio of biodatas; index as i">

{{bio.id}} {{bio.name}} index = {{i}}

</div>

app.component.ts remains same

O/p



🡪 There is another property like index –

first = shows true for first element other will be false,

last shows true for last element,

odd shows odd as true

even shows even as true

app.component.html

<!-- with index of ngFor -->

<div \*ngFor = "let bio of biodatas; index as i; first as f; last as l">

{{bio.id}} {{bio.name}} index = {{i}} , first = {{f}}, last = {{l}}

</div>

o/p:



19. Input Property in Angular 5

🡪 Here app.component is parent component and all the other component created by us is child component

Eg ng gc registration // created registration component

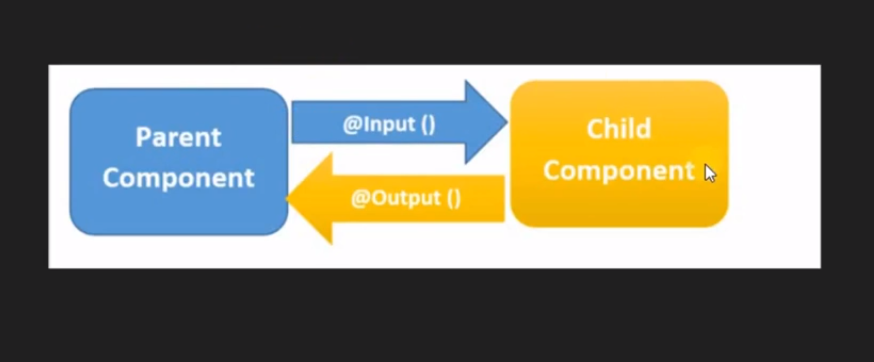
🡪 We have to add child component in parent component i.e. added

<app-registration></app-registration> <!-- New Component -->

In app.component.html

🡪 Input decorator binds a property within child component to receive a value from parent component

🡪 Let’s discuss input and output property with a diagram:



**Parent Component** = class of app.component

**Child Component** = class of registration component

🡪 If we want to transfer data from parent to child we will use input decorator

🡪 If we want to transfer data from child to parent we will use output decorator

Let’s see input decorator now: (4 steps)

Step1: create a property/variable in parent (i.e. app.component.ts) you need to send to child (registration.component)

App.component.ts

export class AppComponent {

// sending data to child (@Input Decorator)

public newData = 'This data is sent by parent component';

}

Step2: Add this property in app.component.html in the defined custom directive.

App.component.html

<app-registration [parentData] = "newData"></app-registration> <!-- New Component -->

Left name i.e. parentData is custom to be used in registration page. Right name is the name defined in above ts

Step3: First import the Input property in line 1 then use @Input decorator to access parentData defined in app.component.html

registration.component.ts

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

export class RegistrationComponent implements OnInit {

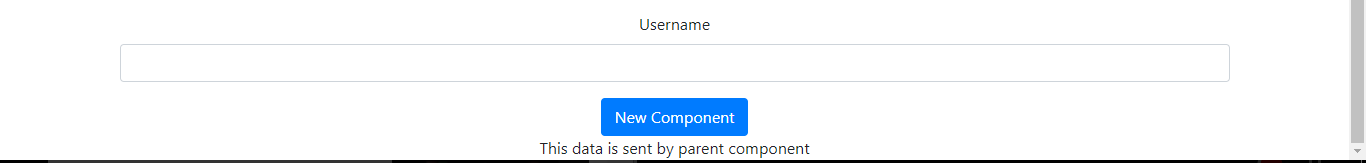
@Input() public parentData;

}

Step4: Now you can use this in html page

registration.component.html

{{parentData}}



20. Output Decorator:

🡪 we will send child component’s (i.e. registration.component) data to parent component (i.e. app.component) using output property / output decorator

🡪 we have to use Event Emitter when we want to send data from child component to parent component

Step1: We will create instance of EventEmitter in childclass (i.e. registration.component.ts) plus include the EventEmitter in import i.e. line 1

registration.component.ts

import { Component, OnInit, Input, EventEmitter, Output } from '@angular/core';

@Output() public childClass = new EventEmitter();

Step2: We have to create an event (say a button) and pass value from that

registration.component.ts

<button (click) = 'sendDataToParent()'></button>

Step3: We have to write back this function in registration.component.ts

We have used childClass instance with emit. Emit method sends data to parent

@Output() public childClass = new EventEmitter();

sendDataToParent() {

this.childClass.emit("This is the data from the child class");

}

Step4: Now in Parent component we will write as event in <app-registration>. When you write =$event, you have access to the emit data of child component

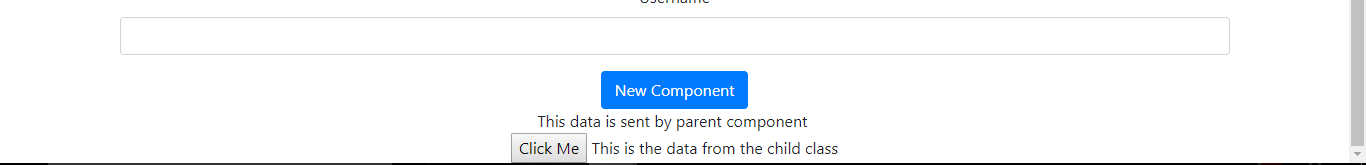
<app-registration (childClass) = "childData=$event" [parentData] = "newData"></app-registration> <!-- New Component -->

Step5: Create a childData property in app.component.ts

// receiving data from child component

public childData;

Step6: All done click the button in the browser, you will get the data here



22. NgContent

🡪 NgConent is a custom element provided by Angular

* ngContent is a way for a parent component to insert markup into a child component.
* @Input and @Output property is for bigger changes. Another easy alternate for it is

<ng-content></ng-content>

* So just in parent component (i.e. app.component.html). Add your data between the caller

i.e. <app-registration>Hiii this is text from Parent Component</app-registration>

app.component.html

<app-registration>

<div>

Hiiii for Parent Component

</div>

</app-registration> <!-- New Component -->

* In the child component i.e. registration.component add the <ng-content></ng-content> where you want that data to show

registration.component.html

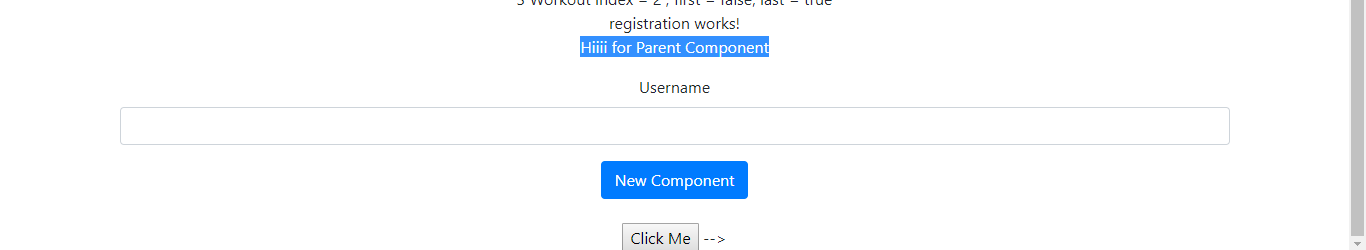
<p>

registration works!

<ng-content></ng-content>

</p>

O/p



* Now suppose there are multiple <ng-content> how will angular know which data to write in which part.

Give ng-content with select attribute and define there as class or id

Eg

registration.component.html

<br><br><h5>Data coming from Child Class</h5>

<div class="card">

<div class="card-header">

<ng-content select=".headingPart"></ng-content>

</div>

<div class="card-body">

<ng-content select=".bodyPart"></ng-content>

</div>

</div>

app.component.html

<app-registration>

<div class="headingPart">

Hiiii for Heading Part

</div>

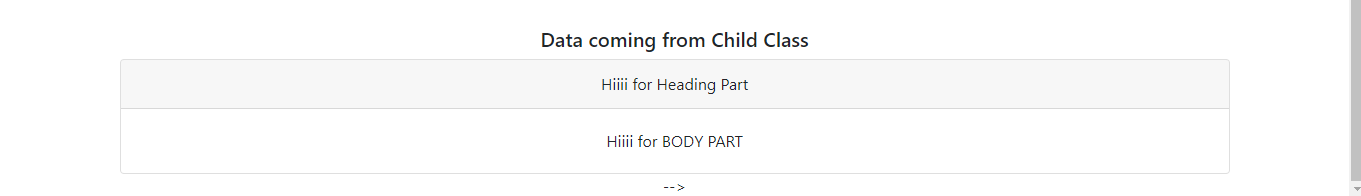
<div class="bodyPart">

Hiiii for BODY PART

</div>

</app-registration> <!-- New Component -->

o/p



22. NgContainer in Angular 5

🡪 When we were sending data from parent to child component and we wrote

<div class="card">

<div class="card-header">

<ng-content select=".headingPart"></ng-content>

</div>

<div class="card-body">

<ng-content select=".bodyPart"></ng-content>

</div>

</div>

* An extra div with headingPart and bodyPart comes which is not at all needed. So for this ngContainer is there.
* The Angular <ng-container></ng-container> is a grouping element that dosen’t interfere with styles or layout because Angular dosen’t put in DOM.

Eg.

app.component.html

<!-- for ng-container -->

<app-registration>

<ng-container class="headingPart"> <!-- We have replaced div with ng-container so that this div will not be visible only its data -->

Hiiii for Heading Part

</ng-container>

<ng-container class="bodyPart">

Hiiii for BODY PART

</ng-container>

</app-registration> <!-- New Component -->

Registration.component.html (same as prev.)

<div class="card">

<div class="card-header">

<ng-content select=".headingPart"></ng-content>

</div>

<div class="card-body">

<ng-content select=".bodyPart"></ng-content>

</div>

</div>

**TRICK – Parent div is replaced by ng-container inside child call i.e. <app-registration></ app-registration**

**23. Create custom Pipes in Angular 5**

**🡪** Normally angular pipes are used to transform the text like to uppercase, lowercase, number, currency, etc.

* We can write our own custom pipe and that will be used in the same way as angular built-in pipes.

Eg.

STEP1: In registration.component.ts create any property **inside class**

Registration.component.ts

// This is for the custom pipe

articles = 'This is for my youtube video';

**STEP2: registration.component.html:** Here use that property and with interpolation with custom pipe name:

<div>

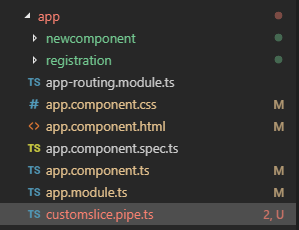
<h4>This is for the Custom Pipe</h4>

{{ articles | custompipe}}

</div>

STEP3:

Now we have to create that pipe file in app folder with .pipe.ts extension: eg customslice.pipe.ts



customslice.pipe.ts

// We have to import file Pipe and PipeTransform (Pipe Transfor is an interface which we will implement below)

import {Pipe, PipeTransform} from '@angular/core';

// Creating Decorator - this is the name of the pipe we want to use

@Pipe({

name: 'customslice'

})

export class CustomSlice implements PipeTransform {

transform(value:string, args: any) {

if(!value) {

return null;

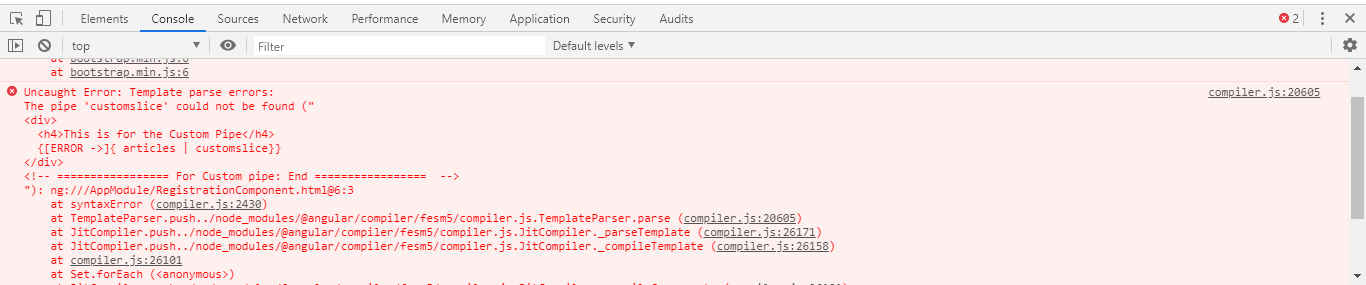
}

value.slice(0,5);

}

}

STEP4: Now we have to register this pipe in app.module.ts (i.e. registering to angular). Otherwise template parse error message will come



We will write our pipe class name (i.e. CustomSlice)

app.module.ts

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

// For using ngModel - two way binding we need this module

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

// importing this to use our custom pipe

import { CustomSlice } from './customslice.pipe'

import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';

import { RegistrationComponent } from './registration/registration.component';

import { NewcomponentComponent } from './newcomponent/newcomponent.component';

@NgModule({

declarations: [

AppComponent,

RegistrationComponent,

NewcomponentComponent

],

imports: [

BrowserModule,

AppRoutingModule,

FormsModule, // Using FormsModule included above

CustomSlice // We register our custom pipe class name after importing above

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

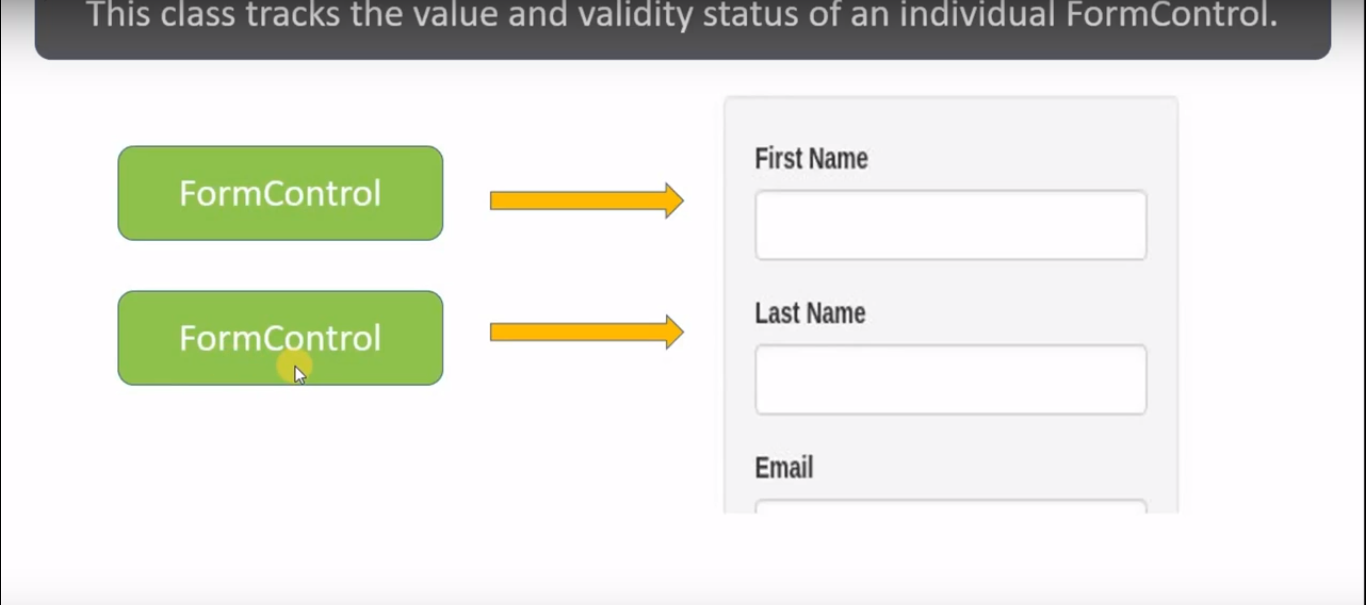
Now our pipe will run accordingly to our definition

24. Form Control vs Form Group in Angular 5

(A) Form Control

🡪 Form Control class tracks the value and validity status of an individual Form Control

* We can know the status of the i/p field like when click (touched), not touched (pristine), written something (dirty). So we can check various validations
* This works only for individual input field. So for different i/p field we will create different instance.
* So we will create instance of Form Control class and use them

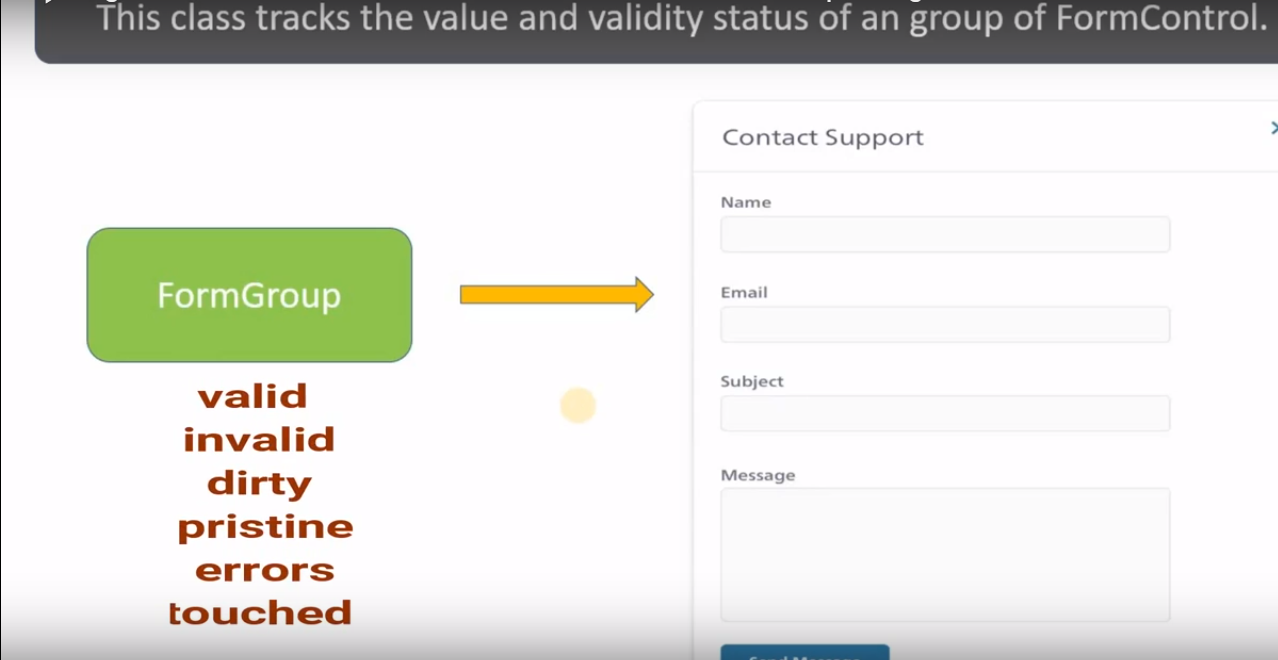


(B) Form Group

🡪 This class tracks the value and validity status of group of Form Control

🡪 When All the submit button is clicked I need to validate all the i/p field at once, for this reason we use it and on the basis of that we will show validation error message.

🡪 So we will create instance of Form Group class. We get same properties as was provided by form control like valid, invalid, dirty, pristine, errors, touched, etc.



So we will use both Form Control for individual and form Group when final submit button is clicked.

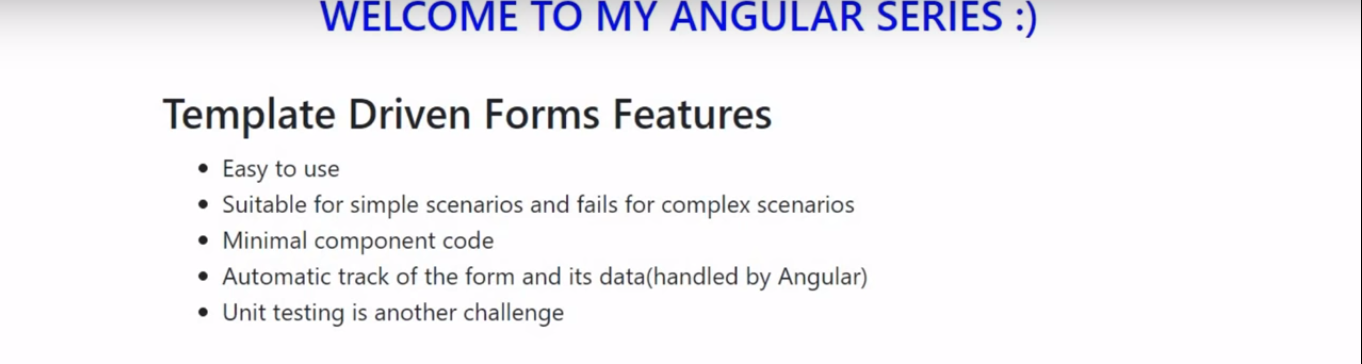
* So for coding we can use 2 ways in which we can use:

Directives and codes

* For **Directive** we will use **template driven form**
* For **Codes** we will use **Reactive Forms (or Model Driven called in Angular 2)**



**25 Template Driven Form in Angular 5/6**

****

* When there is simple form where there is not much i/p field we will use template driven form
* For complex form we will use reactive forms
* The NgModel directive created the Form Control instance to manage the template form control.
* We write minimum code in component as ng model does most of the work.
* Unit testing is little long for this case.

So let’s create a simple-form component

Simple-form.component.html

<div class="container">

<form action="">

<!-- writing ngModel trigers the Form Control method, writing name is must to that i/p field otherwise error in console -->

<div class="form-group">

<label for="firstname">First Name</label>

<input ngModel name="firstname" type="text" id="firstname" class="form-control" #fname = "ngModel"

(click)="showValue(fname)">

</div>

<div class="form-group">

<label for="password">Password</label>

<input ngModel name="password" type="text" id="password" class="form-control">

</div>

</form>

</div>

Simple-form.component.ts

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

@Component({

selector: 'app-simple-form',

templateUrl: './simple-form.component.html',

styleUrls: ['./simple-form.component.css']

})

export class SimpleFormComponent {

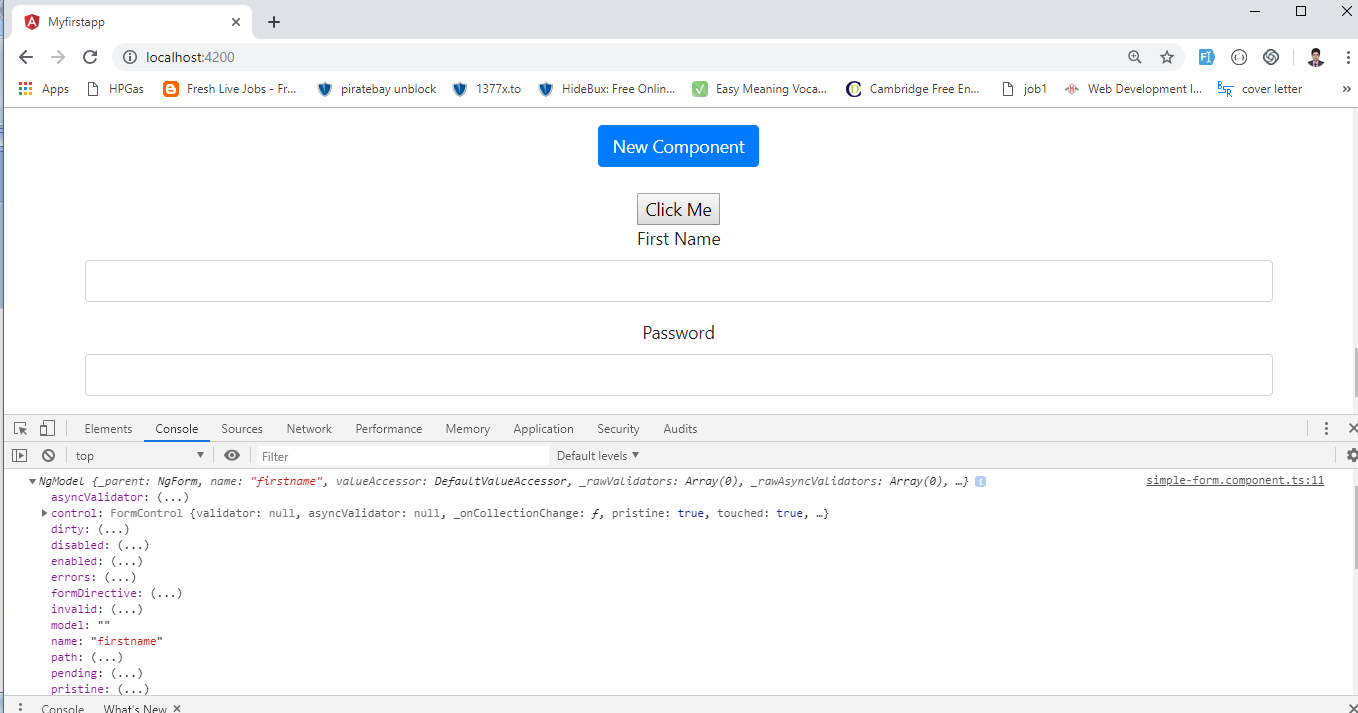
showValue(fname) {

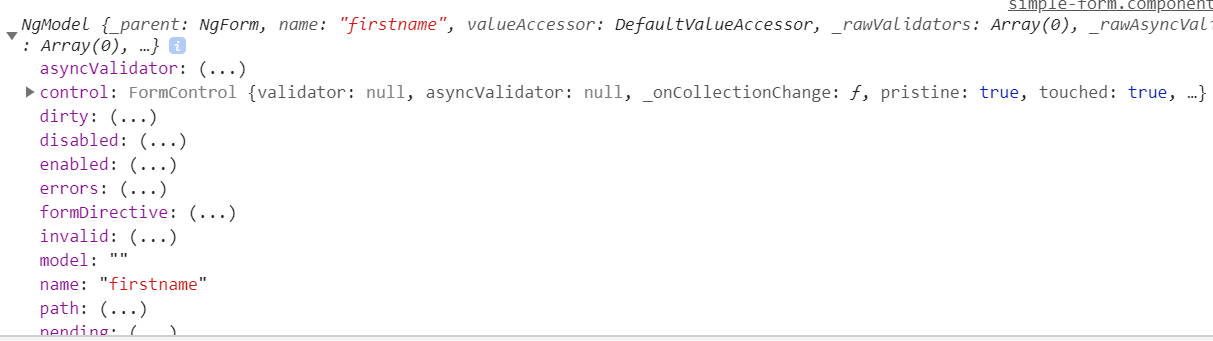
console.log(fname);

}

}

So on clicking the form will trigger the value in console





26. Form Validation in Angular 4,5,6

🡪 We have seen above on clicking an i/p field we get different properties. Most important property is **errors** for checking validation. Next is **touched** and **valid. We also get same name in property as we have provided in i/p field.**

* Create another div inside form control. Put required in input (this is angular required) and in error div put ngIf condition and valid property to check if valid or not.
* If we write something in div error will not come and remove it error will come

Simple-form.component.html

<br><br>

<div class="container">

<form action="">

<!-- writing ngModel trigers the Form Control method, writing name is must to that i/p field otherwise error in console -->

<div class="form-group">

<label for="firstname">First Name</label>

<input required ngModel name="firstname" type="text" id="firstname" class="form-control" #fname = "ngModel"

(click)="showValue(fname)">

<div class="alert alert-danger" \*ngIf = '!fname.valid && fname.touched'>

Please fill the First Name..

</div>

</div>

<div class="form-group">

<label for="password">Password</label>

<input ngModel name="password" type="text" id="password" class="form-control">

</div>

<div>

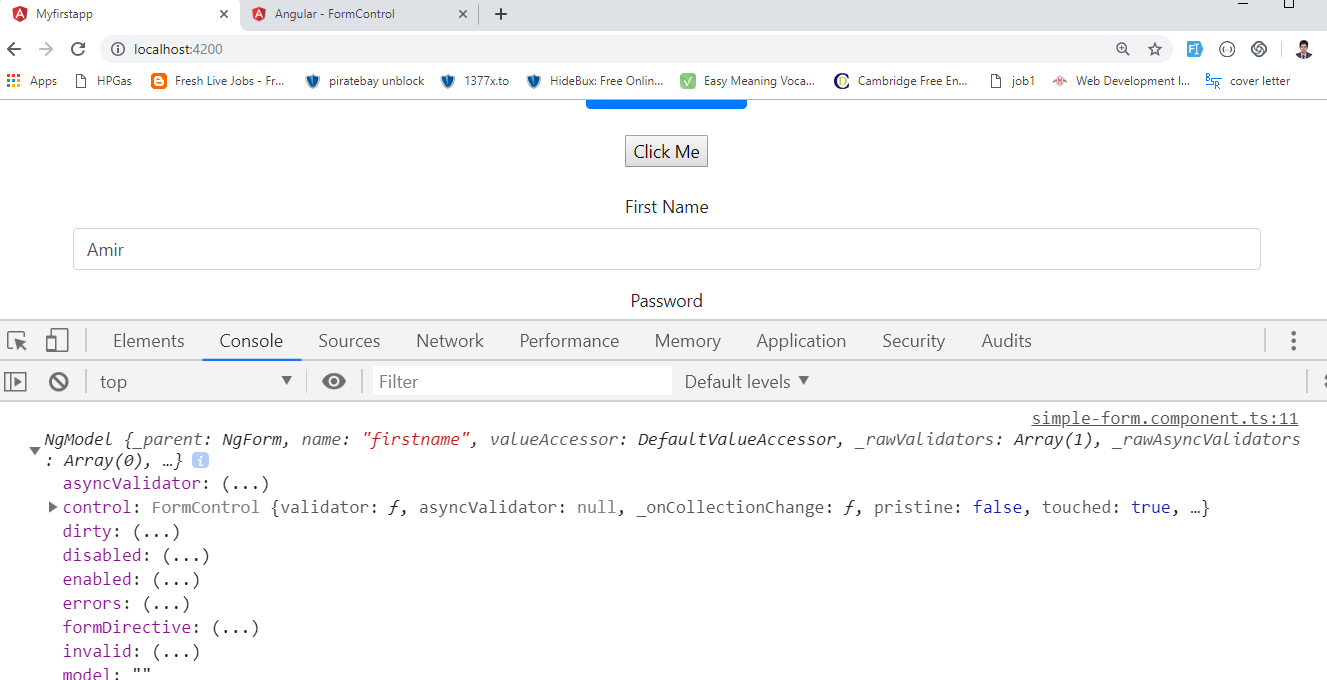
<input type="submit" name="submit" class="btn btn-success">

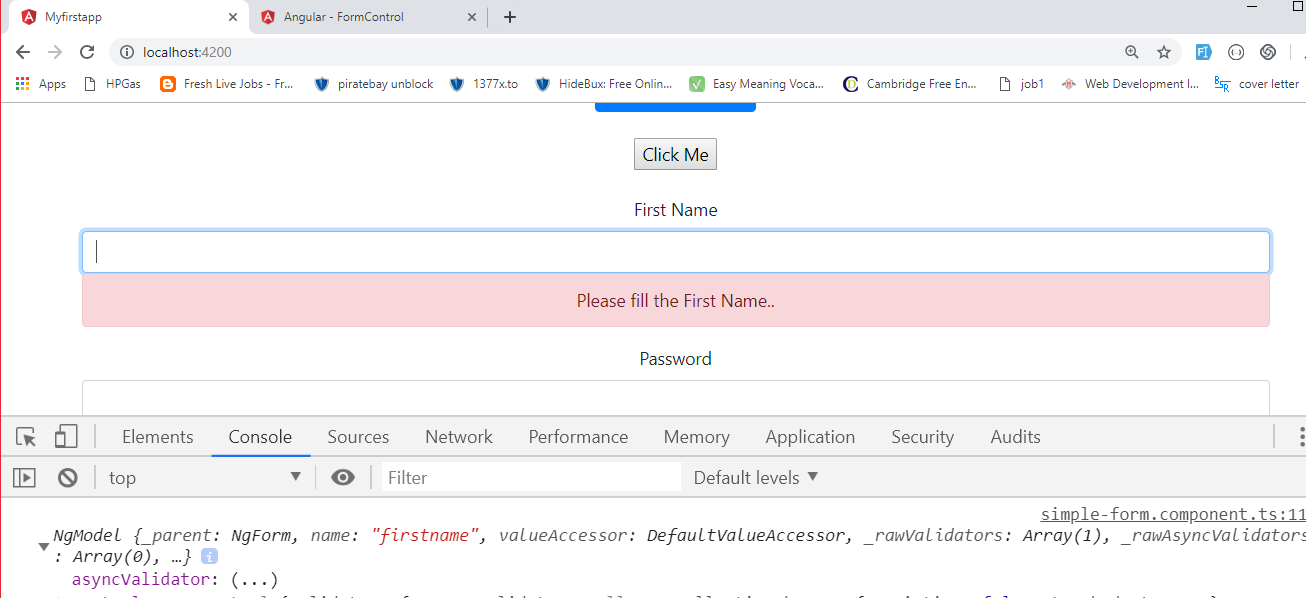
</div>

</form>

</div>

<br><br>





* Whatever we used to write HTML properties like required, title, etc angular have redefined it itself

Simple.component.html

<br><br>

<div class="container">

<form action="">

<!-- writing ngModel trigers the Form Control method, writing name is must to that i/p field otherwise error in console -->

<div class="form-group">

<label for="firstname">First Name</label>

<input required minlength = "5" maxlength="20" pattern="AmirMustafa" ngModel name="firstname" type="text" id="firstname" class="form-control" #fname = "ngModel"

(click)="showValue(fname)">

<div class="alert alert-danger" \*ngIf = '!fname.valid && fname.touched'>

<div \*ngIf='fname.errors.required'>

Please fill the First Name..

</div>

<!-- When we write name.errors.minlength, this property will now be seen in ngModel object hence error message comes -->

<div \*ngIf='fname.errors.minlength'>

Minimum Length is 5

</div>

<div \*ngIf='fname.errors.pattern'>

Pattern is not matching

</div>

</div>

</div>

<div class="form-group">

<label for="password">Password</label>

<input ngModel name="password" type="text" id="password" class="form-control">

</div>

<div>

<input type="submit" name="submit" class="btn btn-success">

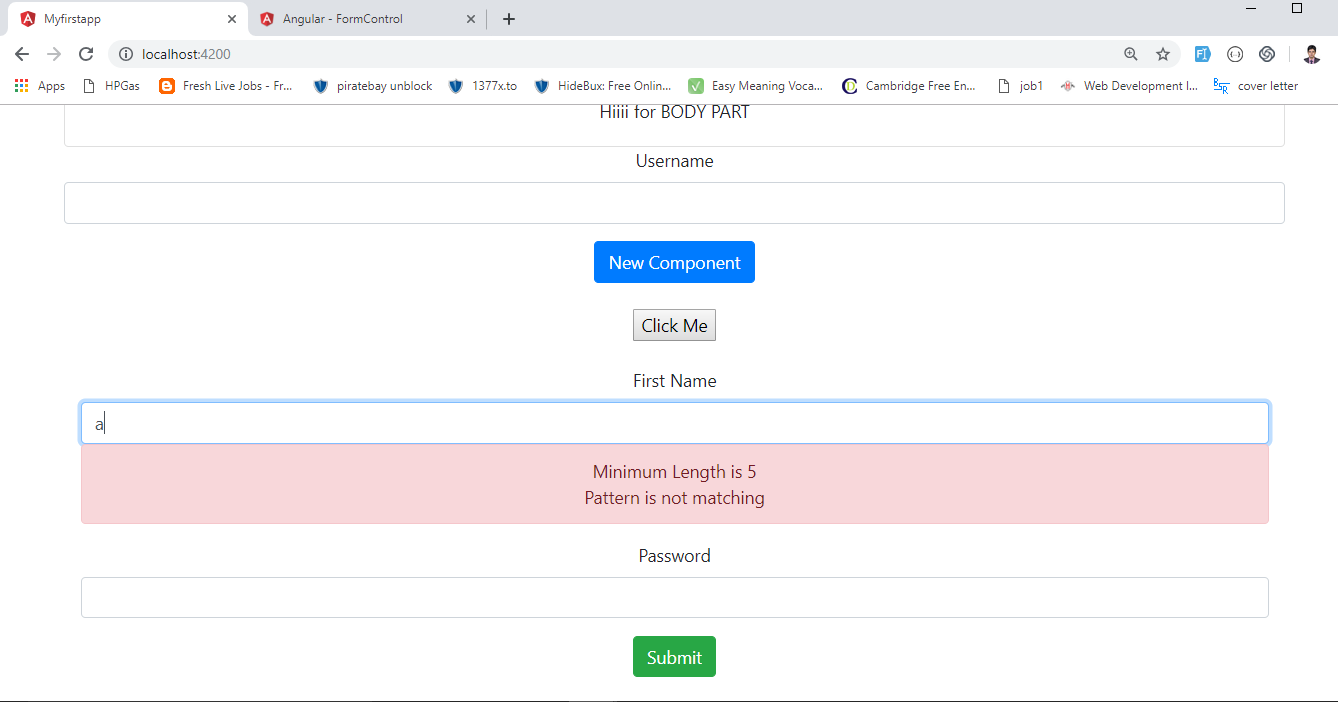
</div>

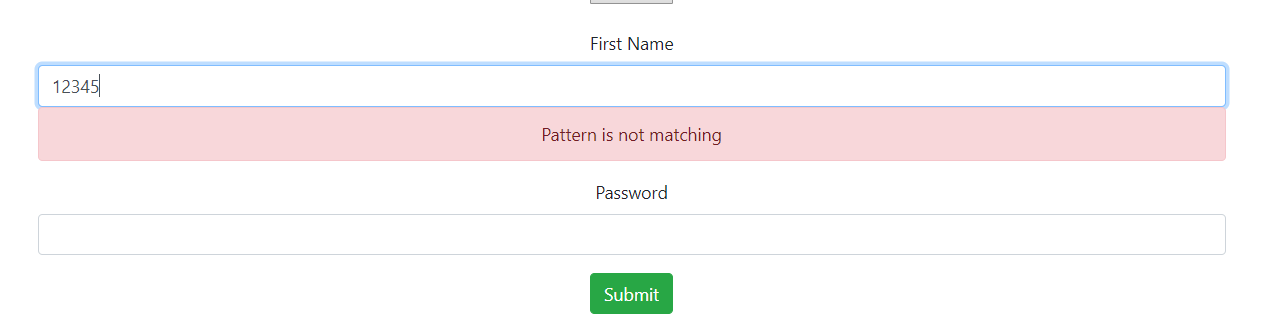
</form>

</div>

<br><br>

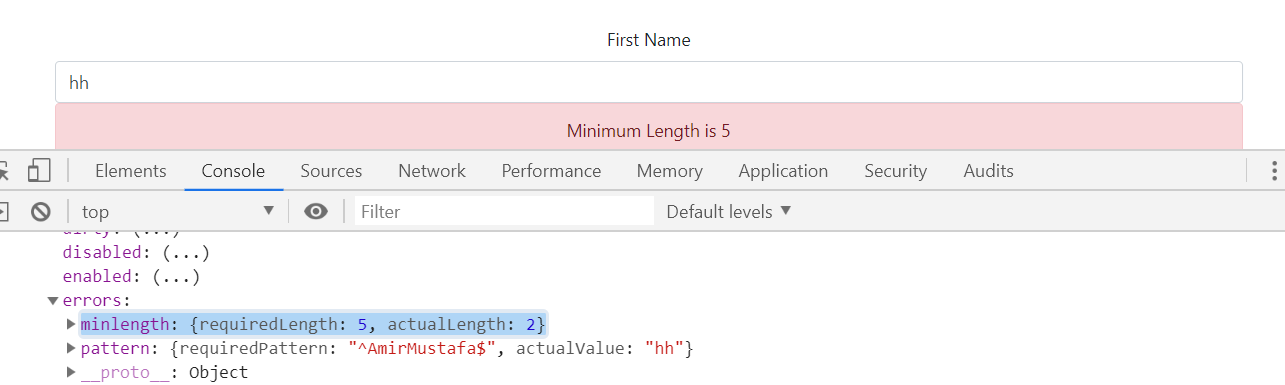
o/p







To see in onject property what comes in object:



27. Services in Angular 5

🡪 Lets see by an eg

Registration.component.ts

public bioData = [

{"name" : "Amir 1",'age': 26, 'profile' : 'Web developer' },

{"name" : "Amir 2",'age': 26, 'profile' : 'Web developer' },

{"name" : "Amir 3",'age': 26, 'profile' : 'Web developer' }

];

Registration.component.html

<ul \*ngFor = 'let data of bioData'>

<li>{{data.name }}</li>

<li>{{data.age }}</li>

<li>{{data.profile }}</li>

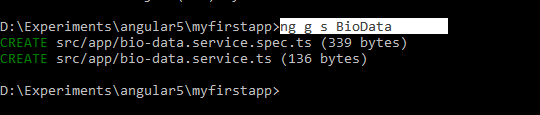
</ul>

We have created an array of objects and used ngFor or printing it.

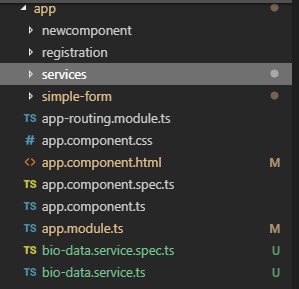
* Now suppose there are 15 or more components and we need this data in all components. So should we have to copy this object in all components (Assume large objects data as well). So this becomes repetitive. Here comes the role of services. Write once use many times.
* STEP1: So to create service in terminal/command prompt

ng g s <service name>

eg ng g s BioData



* Service will be created inside app folder like bio-data.services.ts and bio-data.service.spec.ts



* STEP2: We will create property which we want to reuse. Here we will copy the

Eg. app/bio-data.service.ts (the service page generated)

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class BioDataService {

constructor() { }

// this is the data to send to the component

howServiceWorks() {

return "this is how service works";

}

}

* STEP3: We will register this in app.module.ts . Angular understands only those services, components or pipe which are registered to it. We write inside **import the service** and write in **provider**

app.module.ts (Register the service in app.module.ts)

import { BioDataService } from './bio-data.service';

providers: [BioDataService],

* STEP4: In whatever class we want to use we will add this as the dependency injection where is the need to this service or property

services.component.ts (our service where we want to use it)

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

// Importing our service here as well. (We imported in angular.module.ts as well)

import { BioDataService } from '../bio-data.service';

@Component({

selector: 'app-services',

templateUrl: './services.component.html',

styleUrls: ['./services.component.css']

})

export class ServicesComponent implements OnInit {

public data; // will save data fetched from service here

constructor(private biodataService : BioDataService) { } // accessing the service data alias : Service Name

// now we have access to all he variable and function of the service class

ngOnInit() {

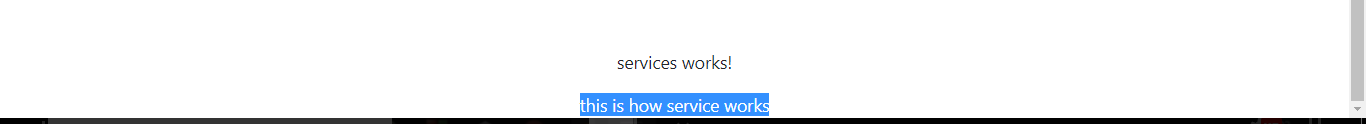
this.data = this.biodataService.howServiceWorks(); // acceessing the data we need(i.e. reqd funcn)

}

}

* STEP5 : Restart ng serve (angular server)

O/P:



Eg2:

Biodata-service.ts (service page)

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class BioDataService {

constructor() { }

// this is the data to send to the component

howServiceWorks() {

return "this is how service works";

}

// method2

biodataService() {

return [

{"name" : "Amir 1",'age': 26, 'profile' : 'Web developer' },

{"name" : "Amir 2",'age': 26, 'profile' : 'Web developer' },

{"name" : "Amir 3",'age': 26, 'profile' : 'Web developer' }

];

}

}

Service.component.ts (our component ts page)

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

// Importing our service here as well. (We imported in angular.module.ts as well)

import { BioDataService } from '../bio-data.service';

@Component({

selector: 'app-services',

templateUrl: './services.component.html',

styleUrls: ['./services.component.css']

})

export class ServicesComponent implements OnInit {

public data; // will save data fetched from service here

public data2; // will save data fetched from service here

constructor(private biodataService : BioDataService) { } // accessing the service data alias : Service Name

// now we have access to all he variable and function of the service class

ngOnInit() {

this.data = this.biodataService.howServiceWorks(); // acceessing the data we need(i.e. reqd funcn)

this.data2 = this.biodataService.biodataService();

}

/\* public bioData = [

{"name" : "Amir 1",'age': 26, 'profile' : 'Web developer' },

{"name" : "Amir 2",'age': 26, 'profile' : 'Web developer' },

{"name" : "Amir 3",'age': 26, 'profile' : 'Web developer' }

]; \*/

}

Service.component.html (our component html page)

<p>

services works!

</p>

{{data}}

<ul \*ngFor = 'let data of data2'>

<li>{{data.name }}</li>

<li>{{data.age }}</li>

<li>{{data.profile }}</li>

</ul>

Ref - <https://www.youtube.com/watch?v=NsxUtDaqgCg&list=PLwGdqUZWnOp0NUuwR85Kq-zwrDA2AVY1Ku&index=12>