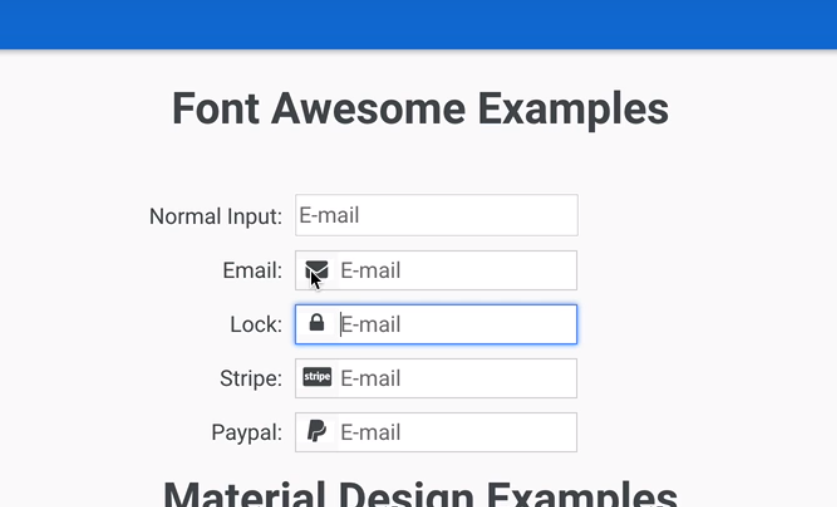
7)Demo of Our First Library - Font Awesome And Google Material Icons Input

Here we saw what we are going to build at last. First input is normal input, then we have inputs with icons. These are font awesome icons. They are open source. Link-

<https://fontawesome.com/v4.7.0/icons/>



You can use them in anywhere. Input fields where we use them (au-inputs) are similar to normal inputs, behaviour that this component has is identical to normal input. That is the idea. We can press tab to navigate through them. We can use any of font awesome icons avalibale.

Then we have some material design examples. This is another directive that we are going to build.this is very similar to font awesome examples, but some differences are there.in this case we are using icon from material design google library. Link –

<https://material.io/tools/icons/?style=baseline>

so this comes with set of SVG Icons. So these are icons that are defined using SVGA and not images. they are ideal suited for mobile because download of svg does’nt take lot of bandwidth.

So this is what we are going to build. this is deceiveingly simple looking component that will bring a lot of angular functionality that we will be using to build more complex libraries. We are going to see things like content projection, we are going to see how to how to style components. We are going to talk about API design of components, how to override encapsulated styles. We are going to cover many of most important notions of component design and content projection with this example. There are couple of tricks that we will need to use to get behaviour that is basically identical to normal input behaviour, so compatible with tab mechanism and shift tab, with focus behaviour. So we are going to use quite some tricks to implement these small components.

We will introduce testing. Being a small example its ideally suited to get introduction to testing ecosystem of angular and we are going to finish off by publishing this into npm and looking at best practices for doing so. We will then increase the complexity of our components that we are going to build.

8. Beginning The Implementation Of The Font Awesome Input Box

We have starting code ready, now lets decide what will be the api of our font awesome component. lets have a look at it. we are going to implement several examples in app.component.html. so it is main template of our application. Open it. Right now we have a simple input in it. So lets get started with our component right away. We are going to create it inside lib folder, which is in app folder.

Create a new component-

ng g c lib/au-fa-input

now this component is ready to use. So we are going to start defining component api straight away. We place normal input in our new component, then we use selector of this component in app.component.html

so with this in place lets now start discussing component api design.one of the properties that we want to define in this component is icon that we want to display. Icon inside this component will be simplyI tag with some classes. One of them will be fa, second class will depend upon icon that we want to display.

Ex-

Au-fa-input.component.html

<i class="fa fa-envelope"></i>

<input class="normal-input" type="email" name="email" placeholder="E-mail">

So go to <https://fontawesome.com/icons>

Here we are displaying the envelope class. So icon that will show different icon is envelope.

So one of inputs of component is going to be name of icon that we want to display.so lets start implementing it. this will be starting point of our api. So we define a property icon, we want to set value of this property from parent component so we use @Input-

@Input() Icon: string;

Lets see how we can achieve it.

Instead of using attaching class manually, we use ngClass directive. We are going to pass here as an expression, we are going to do a call to a method. we are going to call it simply, classes. This will a getter method in our component that is going to return an object that will tell ngClass which classes to display and when. Code-

au-fa-input.componnt.ts-

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

@Component({

selector: 'au-fa-input',

templateUrl: './au-fa-input.component.html',

styleUrls: ['./au-fa-input.component.css'],

encapsulation: ViewEncapsulation.None

})

export class AuFaInputComponent implements OnInit {

@Input() icon: string;

constructor() { }

ngOnInit() {

}

get classes() {

const cssClasses = {

'fa': true

};

if (this.icon) {

cssClasses['fa-' + this.icon] = true;

}

return cssClasses;

}

}

So this getter method returns the object that ngClass expects. Properties of object are going to be name of classes and value of properties is expression returning true or false.

Firstw e check that whther icon property is defined or not. If it defined then we include it in our object.

au-fa-input.component.html-

<i [ngClass] ="classes"></i>

<input class="normal-input" type="email" name="email" placeholder="E-mail">

app.component.html-

<div class="form-row">

<label>FA Input:</label>

<au-fa-input icon="envelope"></au-fa-input>

</div>

<div class="form-row">

<label>Password</label>

<au-fa-input icon="lock"></au-fa-input>

</div>

<div class="form-row">

<label>Stripe:</label>

<au-fa-input icon="cc-stripe"></au-fa-input>

</div>

<div class="form-row">

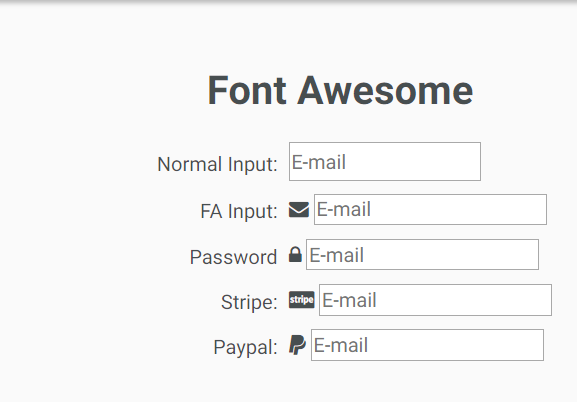
<label>Paypal:</label>

<au-fa-input icon="paypal"></au-fa-input>

</div>

</div>

Output-



Now our components has one api, icon which is icon that we want to display.

This is just a beginning. We are going to use this component to learn about content projection. We are going to discuss components styling, how to separate structural styling from themes that give certain look and feel to component and we are going to introduce testing among several other things.

9) Angular Component Styling - Watch Style Isolation In Action

Lets add style to our component-

<i class="icon" [ngClass]="classes"></i>

<input class="normal-input" type="email" name="email" placeholder="E-mail">

In css-

.icon {

width: 20px;

background: white;

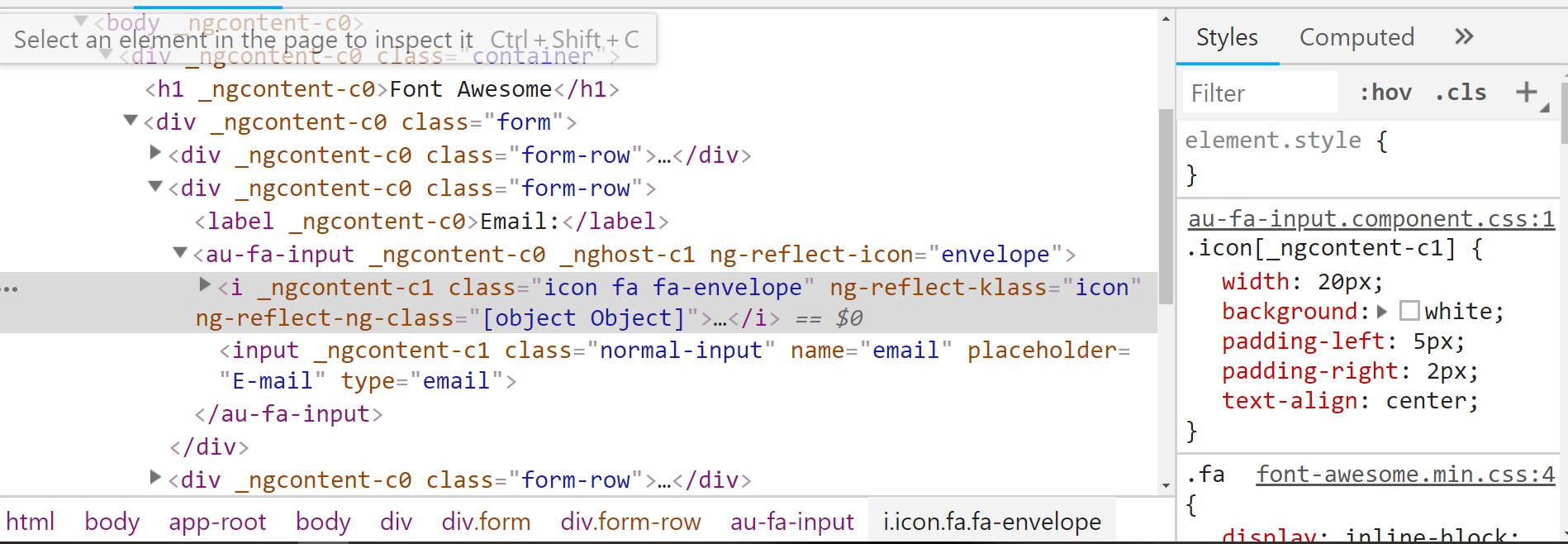
padding-left: 5px;

padding-right: 2px;

text-align: center;

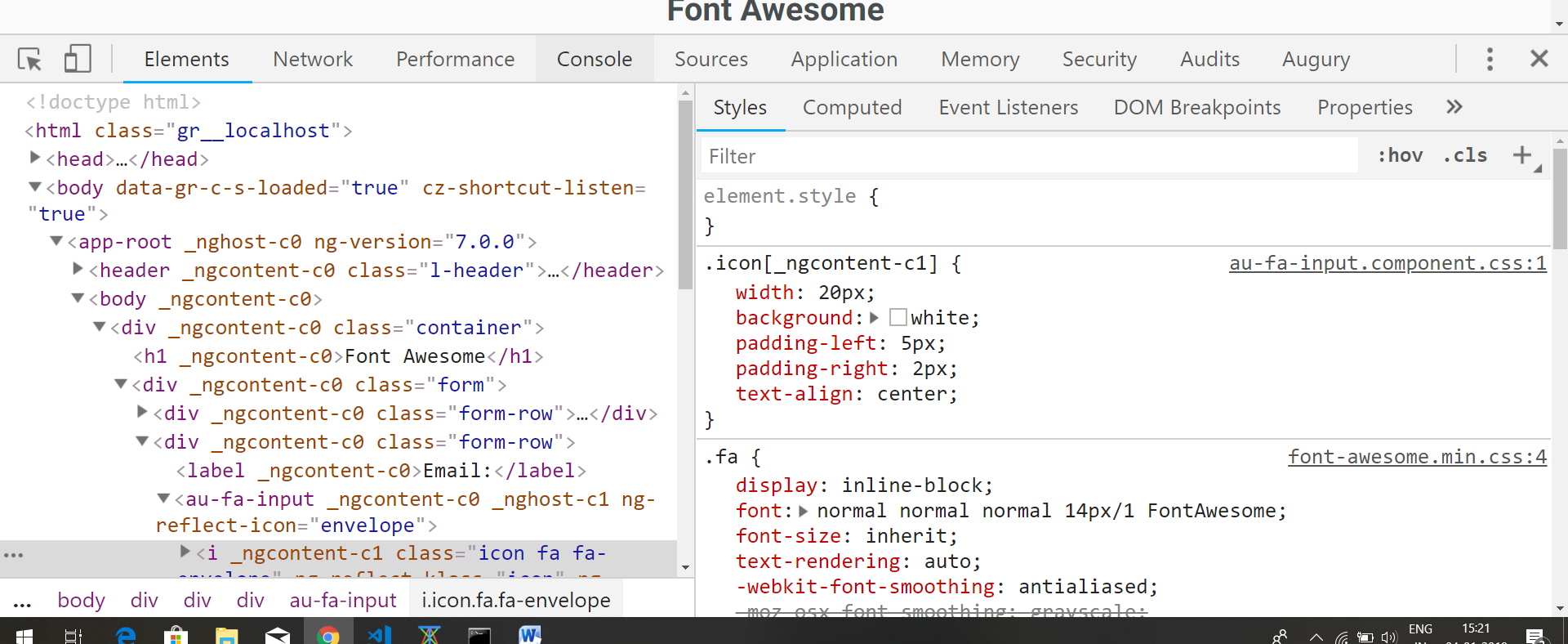
}

I chrome dev tools-

 we can see here that each component has a unique attribute added to it. Then each elemnt inside that component has another unique attribute attached to it(both are not same). But all elements inside a component will have same attribute.as you can see icon has \_ngcotent-c1 attribute. Attached to it. All elements inside au-fa-input will have this attribute.

Now au-fa-input has 2 attributes. \_ngcontent-c0 is added because it is part of app-root(all elements inside app-component will have this attribute). While \_nghost-c1 is just attribte for this component. note that \_nghost-c1 is not attached to all elemnts insie our component. for that purpose we have different selector.

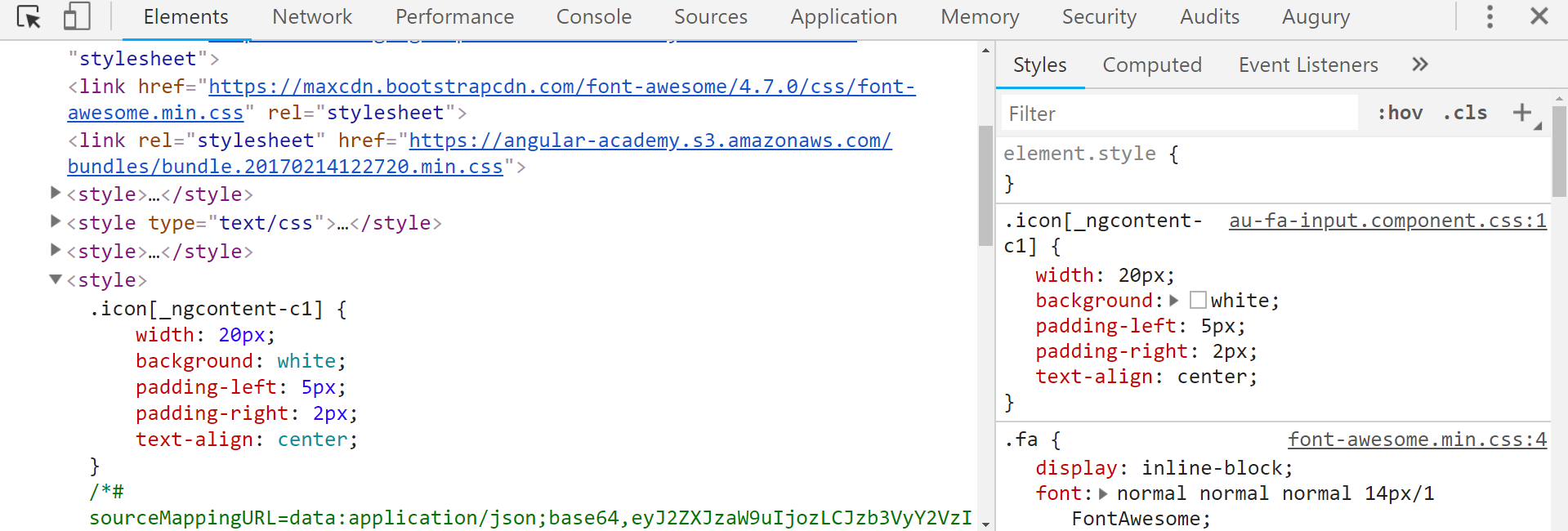
This attribute are used to define css selectors , which only targets elements of that component.as we can see-



So this icon class style will only be applied to elemnt with icon class which is inside the au-fa-input component. this is how angular achieves the style isoloation. As this icon class style is more specific(as we have added attribute in class selector) , it is less likey to be overridden by outside style.

This is not full component isolation by using shadown DOM, but it very good level of style isolation that in practice tends to work exactly as expected.

. Now question is how styles defined at component level are included in final index.html. they are added in index under style tag by webpack. If we go to top of index.html which is generated by webpack, we can find these styles under style tag. in selector of these styles we have cryptographic attributes, to make style more specific.

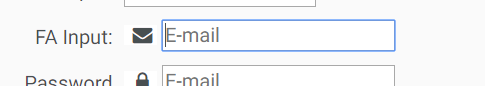


So lets keep in mind where to find these styles auto generated by angular, if we need them for trouble shooting. These will come in handy for us later in the course.

So this explain how angular style isolation mechanism works.it works by increasing the specificity of our components stylesheets , adding them attributes to those styles that will make sure that style is only applicable to that component or inside that particular component. Now styles are not 100% isolated. We will learn more about style isolation further in course.

10) Learn An Angular CSS Extension Feature - The Host Pseudo Selector

Lets continue learning about angular component styling.



What we want to do there is, we want to include our icon in blue border around input field. We want to wrap our complete component inside this border. We know that icon class that we defined in css file of our component will be applied to elements inside our components. Here we dnt want to style any element inside component but we want to style host element of component itself i.e we want to style selector of our component.one approach will be wrap it inside div then style that div, but angular provides us better approach.

Angular give us special css selector-

**:host {}**

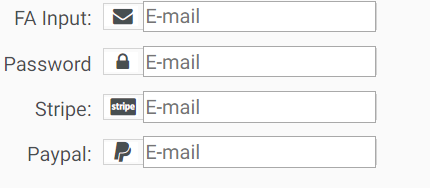
This selector will allow us to style the host element of component itself (elector of component) in css file of our component we do this-

:host {

border: 1px solid lightgrey;

}

But still we get this separation-

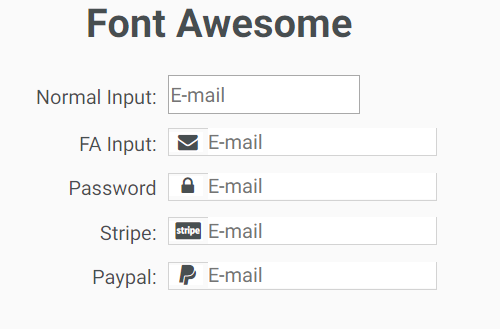


We dnt want this separation. Now we have border around our component but we also have border that browser defines for input. We want to delete it.so we do this-

input {

border: none;

}



Border at bottom is not visisble so, lets add little bottom padding. Now we have a single border. But when we click on input filed, only textfiled is highlighted. Border around the icon is highlighted. So we set outline equal to none for input. Overall css file-

.icon {

width:20px;

text-align: center;

background: white;

padding-left: 5px;

padding-right: 2px;

}

:host {

border: 1px solid lightgrey;

padding-bottom: 1px;

}

input {

border: none;

outline: none;

}

Now we do not focus functionality but atleast we dnt have that outline that was making the inout look not look like normal input. We will add that functionality later.

Right now lets continue styling our component and lets talk a bit about about component styling best practices and how to split up styles into multiple different types of styles and learn another extended css features of angular.

11. Component Styling Best Practices - Ensure Solid Styles For Multiple Widget Sizes

Here we will talk about component styling best practices , how to organize style and we are going to learn a couple of techniques that will help us ensure that styles that we put inside inside this component will work well in third party applications and that they are solid in different sizes of widgets.

Lets see how we can style our component so that we can make input little bit higher. One way will to put the height in input selector in css of our component. Here-

.icon {

width:20px;

text-align: center;

background: white;

padding-left: 5px;

padding-right: 2px;

}

:host {

border: 1px solid lightgrey;

padding-bottom: 1px;

}

input {

border: none;

outline: none;

}

That is something we want to avoid. we want to make these styles that we use inside the component as much as possible structural styles. So what we mean by structural styles? These are styles that are inherenly linked to the way component is designed. For ex this component is designed to have an input inside of it, that for ex should never display it’s border(input’s border). So external border of the component is meant to replace the actual browser rendered border of input. So these styles-

input{

border: none;

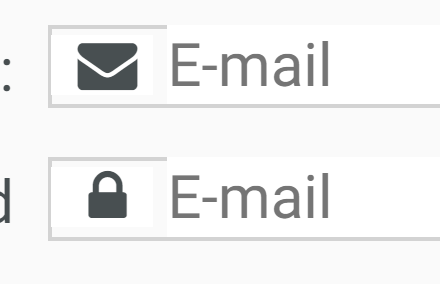
outline: none;

}

never need to be overridden because they are valid for any instance of component. But for example the background, the white background of the input or colour of icon or may be color of border, those are more theme related styles. So those could be easily overwritten depending on the overall feel of the application where we deploy our component.

in icon clas selector we have background white property, this is not the case in many pages. In many pages the input have other color than black, they might have light tone of yellow for example. So if use this component in those pages we have to make background property in incon class selector overwritable.(see css file in starting of lecture). We will see how to do that in a second.

Right now we want to make sure that structural style that we have here is solid and work well with different sizes of the component.one way to ensure that is to use browser zoo functionality. Upon zoomig we noticed that there is small border around icon, background is not completely white.



So this is one way of seeing how styles will scale up.

Another is to simply change the style of the element. Different pages will need element of different size. Go to index.html

au-fa-input {

height: 30px;

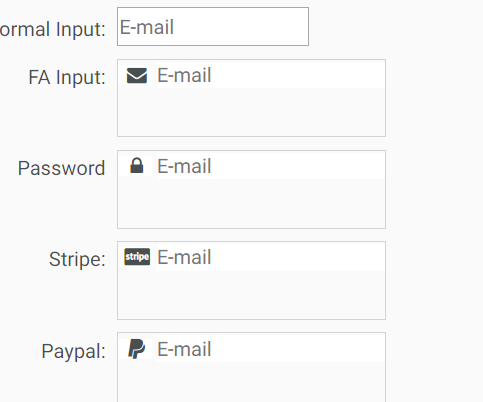
}

So we use incease the size to very large size then we will need,/ this is just to see how style will scale. Now this style wnt be applied. this already indicates that these style needs some reworking. To let this style applied , to css od component-

Add this property to :host selector-

display: inline-block;

so this means we can define properties like height etc. now we can see that there is difference of color in background.



So we remove

background: white;

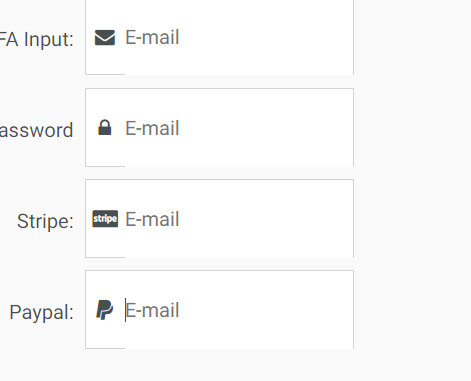
from icon class and apply it to :host

now whole component is white. So in this way we will have color for input background and icon background.

Now we can note that input is not correctly centered . for this we give

height: 100%;

to input .



Now we have rendering problems in bottom of the border box. For this give this property to input-

box-sizing: border-box;

it means height incudes content, borders and padding. By default height does not include padding and border.

Now if we zoom it in, zoom it out, we can see that our style is scaling well for different sizes. Now these structural sizes are much solid.

Next, we are going to talk about component api design best practices and we are going to talk about content projection.

12. Component API Design - Simpler and More Reusable Components With ng-content

Let’s talk about content projection and best practices for design of the api of our components. We will get back to component styling later in the course. Right now lets talk about component design. We can see that placeholder is hardcoded in the our component. We did this-

In ts file of au-input -

@Input() placeholder = '';

In html of au-input-

<input class="normal-input" type="email" name="email" [placeholder]="placeholder">

In app.component-

<div class="form-row">

<label>Password</label>

<au-fa-input icon="lock" placeholder="password"></au-fa-input>

</div>

This works fine as expected. So maybe we wuld thing that it would be a good design, to do same thing for all other properties of input. But let me give you an exmpale why this will not be a good idea. because an input html tag might have lot of properties in production.

<input type="email"

size="20" data-stripe="email"

name="email" placeholder="E-mail"

required="required" autocomplete="email">

it will have multiple validators, it can have custom stripe attribtes. Like this it can have lot of properties. So at that point vast majority of code of component would be to just to duplicate the properties that are already part of normal html input. There would not be much functionality to component and also it will create other problem.

Another problem will be what if this input is part of angular form? Then we would want to add formControl directive to our input. we would not be able to do so here at level of template (app.component.html)where this component is being used. So if we want these to be integrated with angular form, then we will have to duplicate all angular form properties on input and that is not practical. So instead lets look at alternate component design.

This is general design practice that if possible we should avoid wrapping plane html elements especially liv elements like input or select boxes, These elements are better if they are included here in original component(app.component.html in our case) and not being wrapped around in custom component because of this problem that we have just seen. So how could we do this? Angular provides us a feature which is meant precisely for this, it is called **content projection**.

The way this works is,Between selectors of our custom component we can pass some content and this content can be used in template of our component.

Now we do this-

app.component.html-

<div class="form-row">

<label>FA Input:</label>

<au-fa-input icon="envelope">

<input type="email" placeholder="Email" name="email">

</au-fa-input>

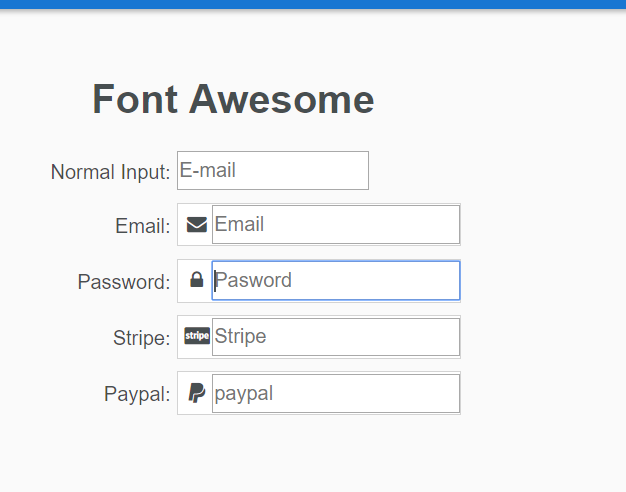
</div>

au-fa-input.component.html-

<i class = "icon fa" [ngClass] ="classes"></i>

<ng-content></ng-content>

Now we dnt have to duplicate all basic html properties. But there are some styling issues, we will look into that.



We will see more advanced uses of ng-content throughout the course. For for now lets look at this styling issue.

Our component was styled correctly but now with the use of ng-content, we can see that border:none style is no more applied, for example. So how is this working, what is going on, isit possible to style using ng-content? we are going to solve these questions.

Questions-

1)about input being part of form, but not using content projection

<https://www.udemy.com/angular-advanced-masterclass/learn/v4/questions/5998512>

2)making projected content part of angular form

<https://www.udemy.com/angular-advanced-masterclass/learn/v4/questions/6005350>

13. Understanding ng-content and Style Isolation - Learn The Deep Style Modifier

our styles were working fine initially ,but now they are not working fine now. Somehow styles that we defined inside au-fa-input.component.css is not being applied to elements that injected from ourside(by use of **content projection**). But it was getting styled when this input was in au-fa-input.component.html. so what happened.firs lets see styles which are working. Icon is working(it is in our template only, nit projected from outside) If we inspect icon in chrome, we can see this-

**<i \_ngcontent-c1 class="icon fa fa-envelope" ng-reflect-klass="icon fa" ng-reflect-ng-class="[object Object]"></i>**

So this cryptographic property **\_ngcontent-c1** is applied to all elements that we use in au-fa-input.html. in au-fa-input.css we applied these styles to icon-

.icon {

width:20px;

text-align: center;

padding-left: 5px;

padding-right: 2px;

}

In final index.html generated we have this selector added in style tag-

.icon[\_ngcontent-c1] {

width:20px;

text-align: center;

padding-left: 5px;

padding-right: 2px;

}

So it is by this cryptographic attribute styles are being applied. in au-fa-input.css we also have these styles for input tag-

input {

border: none;

outline: none;

height: 100%;

box-sizing: border-box;

}

We have also applied styles to input.In index.html we can find these styles as-

input[\_ngcontent-c1] {

border: none;

outline: none;

height: 100%;

box-sizing: border-box;

}

So everything is fine why these styles are not being applied to input? Problem is,when input tag was in au-fa-input.html then this crypto attribute was added to it. Now input tag is not inside that compoent. We use ng-content to get it inside our html file. Now that input element is inside app,component.html so it has crypto attribute for that app.component.

**<input \_ngcontent-c0="" name="email" placeholder="Email" type="email">**

So now we know why our styles are being applied to input(which previously was getting applied). so the reason that this happens is to make sure that any html that gets passed on from outside of component itself, gets their styles preserved. If we have added any styles to input in app.component , then it wnt be overridden when we pass this input from app.component.html to au-fa-input.component.html via projection(by using <ng-content></ng-content>). This is actually a feature. This is why to ensure that if we add some styles in this component and we pass it on to another component via projection, then the styles that we have added here are also preserved in projected component.

So this is how angular’s style mechanism works by default and reason why it works like that.

But in our case of projected input using ng-content, we want to do something little bit different. We want to override the styles of projected component. normally we do not want to do that ,but in particular case of this component we are expecting an input for sure that we want to apply some very specific structural styles.lets see how we can do that. We add :host to our input selector in css of our au-input. It means any input that is part of this template should have this style. So-

:host input{

border: none;

outline: none;

height: 100%;

box-sizing: border-box;

}

But still it does solve our problem. It is not applied to elements that come via projection. This style is still going to be applied to inputs that are put explicitly inside the template of out fu-input component.

So in order to solve that double border issue, it is relative rare occasions where we want to break the syle isolation barrier.we w ant to mke sure that any input inside of this componentindependent of whether it was projected or not, should have some styles. We do this-

::ng-deep input {

border: none;

outline: none;

height: 100%;

box-sizing: border-box;

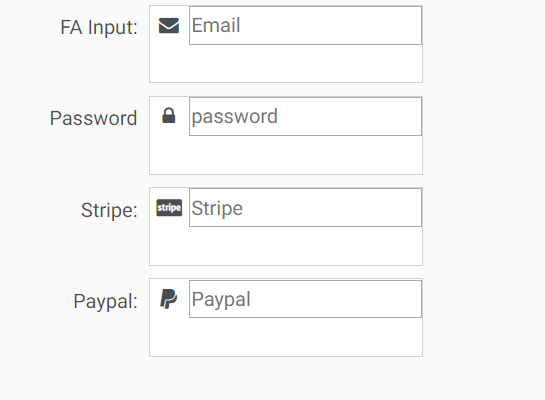
}

It means apply these styles to any input which is in this component, either it is there explicitly or it has come via projection. Earlier we used-

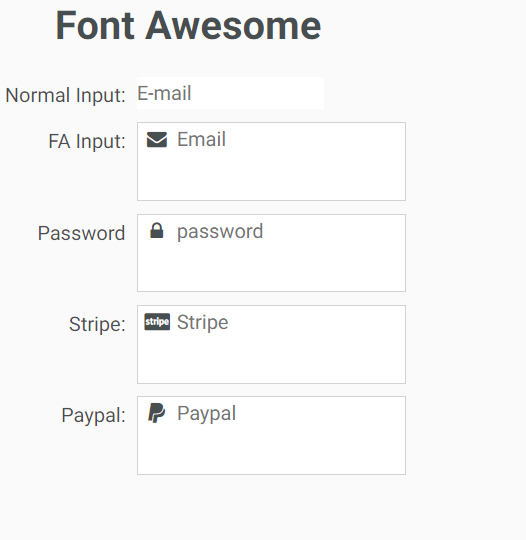
/deep/ input {

}

Initially our inut was like-



Now it is like this-

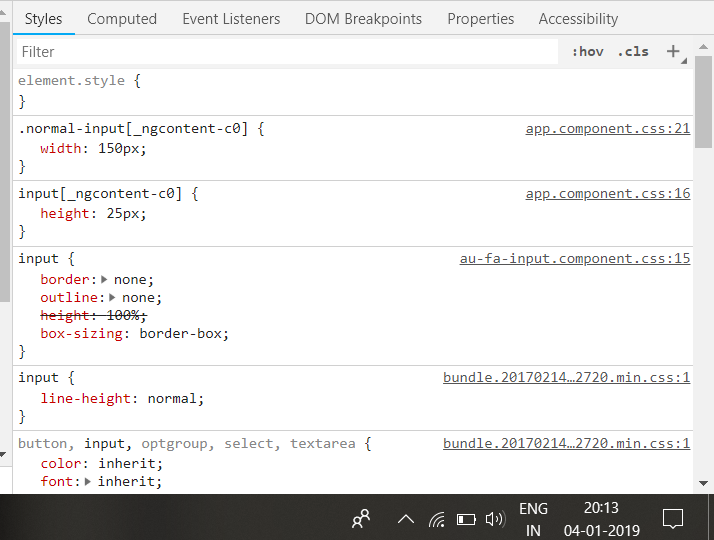


With this solution we have created a another side effect which is that normal input(which is inside au-input component) is not being styled properly. It do not have any border. If we inspect in chrome dev tools(inspect normal input in au-input component whch do not have boreder now) we can see that style that we provided to ::ng-deep input {}, has been added to normal input also. So in index.html, we can find this selector-

Input {

All styles in ::ng-deep input

}



So what ::ng-deep did was, instead of adding styles to inputs of au-fa-compoent(using cryptographioc attribute), it added styles to input selector in general. So style is not isolated. So instead of adding, this selector-

Input[cryptographic attribute of component] {

}

It added this-

Input {

}

So now this style will be added to input in any level of component tree, hence name ::ng-deep. But we do not need this effect. We want this styles (added with selector **::ng-deep**), to be applied only to input in au-fa-input.compoent. to do this use this-

**:host ::ng-deep input {**

**}**

:host ::ng-deep input{

border: none;

outline: none;

height: 100%;

box-sizing: border-box;

}

Now this style will be applied only to inputs inside au-fa-component(to projected also). But how?

Now this style is added in index.html-

[\_nghost-c1] input {

1.  border: none;
2.  outline: none;
3.  height: 100%;
4.  box-sizing: border-box;

}

Now our app.component has this crypto attribute-  **\_nghost-c0,** it is placed on selector of this component like this -

<app-root **\_nghost-c0** ></app-root>

All elemnts that that are inside this app-root will have this attribute- **\_ngcontent-c0.** Like this-

<app-root **\_nghost-c0**  >

<input type=”text” - **\_ngcontent-c0>**

</app-root>

Similarly au-fa-input.component has this ,atribute- **\_nghost-c1,** like this**-**

**<au-fa-input \_ngcontent-c0 \_nghost-c1></au-fa-input>**

But it will also have **\_ngcontent-c0**  as it is element which is inside of app.component.

all elements inside this component will have this attribute- **\_ngcontent-c1,** like this-

**<app-root \_nghost-c0 >**

**<au-fa-input \_ngcontent-c0 \_nghost-c1>**

**<input \_ngcontent-c1="" placeholder="password" type="text">**

**</au-fa-input>**

**</app-root>**

Now this style-

[\_nghost-c1] input {

1.  border: none;
2.  outline: none;
3.  height: 100%;
4.  box-sizing: border-box;

}

Will be added to input which is inside the element with attribute **\_nghost\_c1.**

So this is how angular styling works and how it interacts with content projection, ng-content functionality.

Next we are going to learn further features of angular core. We are going to talk about content children and being able to inject configurator elements in our component. we are going to look how we can interact with projected elements and we are going to to see how we can add auxiliary directive to interact with our input. We are going to see how we can use that to simulate the focus functionality of the component. because gaol here is to make these input really look like a native input. We will also get back later to component styling and how to create theme for the component.

14) The ContentChild Decorator, How Does It Work ? Component Design Best Practices

Here we will see how to interact with component’s content using content child., how to use contentChild to inject DOM elements directly from content, how to use it to inject other directives and how afterContentInit lifecycle hook works and which use cases we want to use it.

Now when we focus on input we have blue border around it. So now what we are going to do is, we are going to add this functionality of simulating here a blue border around the au-input component. This is to simulate the focus of the component. So idea here is to detect focus and blur events here at the level of native element which is input that is hidden here without border, to detect all the native browser events at the level of that component. So focus and blur and react to them by modifying the border outside wrapping element.

Now if we had this input elemnt,inside the au-fa-input.component.html, then we can simply add event handlers for focus and blur like this-

**<input (blur)=”function1()” (focus)=”function()” type=”text”>**

But here we are using content projection. This our html file-

<i class = "icon fa" [ngClass] ="classes"></i>

<ng-content></ng-content>

What we need in this case is to grab a refrence to the native dom element itself.lets try to do that in au-input.compoennt. in angular we have functionality to gran refrence from content part of component. Content part of element is anything that exist inside it’s selectors, like in our case-

<au-fa-input icon="lock" >

<input type="text" placeholder="password">

</au-fa-input>

Content part of au-fa-input component is input element. So the functionality that angular provides for interacting with components content is ContentChild and ContentChildern decorator.

In app.component.html-

<div class="form-row">

<label>Password</label>

<au-fa-input icon="lock" >

<input type="text" #input placeholder="password">

</au-fa-input>

</div>

Here we have placed local refrence on input element.

In au-fa-input.component.ts-

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

@Component({

selector: 'au-fa-input',

templateUrl: './au-fa-input.component.html',

styleUrls: ['./au-fa-input.component.css'],

encapsulation: ViewEncapsulation.Emulated

})

export class AuFaInputComponent implements OnInit, AfterContentInit {

@Input() icon: string;

@ContentChild('input') Input: HTMLInputElement;

constructor() { }

ngOnInit() {

}

ngAfterContentInit() {

console.log('input' + this.Input);

}

get classes() {

const cssClasses = {};

if (this.icon) {

cssClasses['fa-' + this.icon] = true;

}

return cssClasses;

}

}

Here we sue the type HtmlInputElement, this added implicitly for us due to the configuration of the angular cli, that is bringing series of very useful definations from typescript compiler itself.

So using ContentChild we got the access to input element. To ContentChild we have passes string as a value. this string is local refrence. We can also pass it class of components to get access to first instance of component. we can di same thing for directive.

So this is one way of getting the native DOM element injected here at level of our component even if it’s coming via projection via <ng-content>. But as we can see it is not very convenient solution. We have to add #input to input that we pass between selector of our component.

We will prefer to do instead is to have here some type definition(where we pass local refrence to ContentChild) so that we can try it with HTMLInputElement-

So instead of using this-

@ContentChild('input') Input: HTMLInputElement;

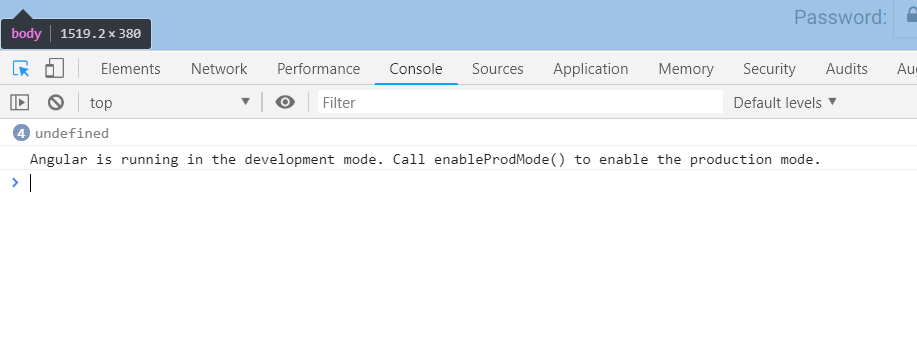
We replace local refrence with with typeDefination, lets try with HTMLInputElement-

So we do this-

@ContentChild(HTMLInputElement) Input: HTMLInputElement;

Now we can remove all local refrences in app.component.html. we don’t want to use these local refrences because it is easy to forget and then we dnt have any idea why this is not working. So we would prefer to have an api that is somewhat similar to what we have here, where we simply pass in plane HTML Input(without local refrences) and and it would work out of box.

But get undefined on console. So this is not working.



So lets see how we can use Content Child to make this simpler component directive where we dnt have to mark the input(with local refrence) that we pass on to component with a special refrence or anything similar, we can just pass th plain html input and it will just work.

15. ContentChild, Directives and HostListener - Implementing the Input Focus Feature

Here we will implement focus functionality that will simulate that this input element really looks like true DOM native element. We are going to learn more about ViewChild amd we are going to see how it can be sued to interact with auxiliary directives.

So now lets try to grab a refrence to input element so that we can detect the focus and blur events and react to it to create border around our au-fa-input component similar to normal input. So if we want to detect elements inside a fa-input component, what we need is an auxiliary directive.

So in lib/common we create input-ref directive

**ng g d lib/common/input-ref**

by default selector of directive is-

selector: '[inputRef]'

it means it is applied to any element that has attribute inputRef.

We change selector to-

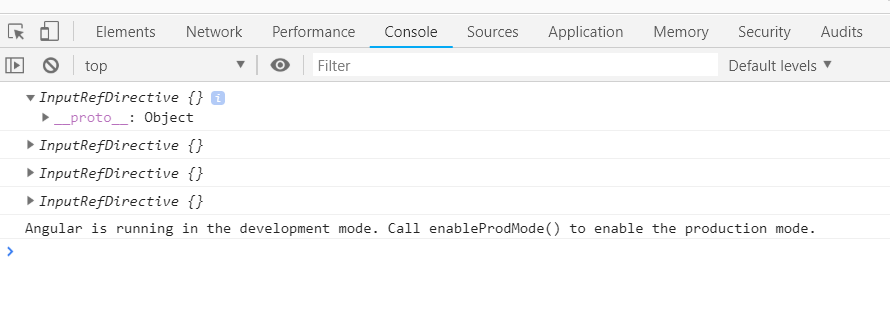
selector: 'au-fa-input input'

now this directive will be applied to any input which is nested inside au-fa-input. We copied the selector ofau-fa-input component.

So with this in place we have here a directive that angular is going to associate at component bootstrap time to any input inside au-fa-input element. So we can use this type(directive) to have input directive injected in our au-fa-input component. So we use this refrence directive here. So in au-fa-input.component.ts-

@ContentChild(InputRefDirective) Input: InputRefDirective;

So now refrence to input directive is injected into our component. let’s see what we get in console, when we print input variable on console-



So unlike the injection of template refrences, this is not a refrence of DOM element, this is refrence to directive that is applied to DOM element. So we still do not have access to DOM element. And althrough we could, we are going to show later in course that we are going to try to keep doing this in angular way without directly manuplating the Dom element as much as it is practical. Whenever it is not practical we are going to do it. This is typical case of everything that relates around keyboard events.

But right now we can still use here the functionality of directives of angular to detect blur and focus events here in host element of this directive(input element that is projected into our component).

In our directive we use HostListener directive to listen to focus and blur events on input. Now we also want to keep track of focus status of input. We want to know if the focus is active or not. So we define a Boolean variable focus. It is public so au-fa-input component can access it. Now we have focus state at the level of refrence directive, lets see how we can use it at level of au-fa-input component, to add focus borders around my component, simulating a true native input element.

Input-ref.directive.ts-

import { Directive, HostListener, HostBinding } from '@angular/core';

@Directive({

// tslint:disable-next-line:directive-selector

selector: 'au-fa-input input'

})

export class InputRefDirective {

focus = false;

constructor() { }

@HostListener('focus') onFocus() {

this.focus = true;

}

@HostListener('blur') onblur() {

this.focus = false;

}

}

Question-

1. just getting value of input-

<https://www.udemy.com/angular-advanced-masterclass/learn/v4/questions/6005122>

16. Implementing the input Focus Functionality - The HostBinding Decorator

We will add a state class to our component whenever focus attribute(in directive) is true. This state class is going to be called input-focus. So this class will be added or removed using value of focus variable. Lets define styles when this class is attached to component.

au-fa-component.css-

:host(.input-focus) {

outline: none;

border: 1px solid #4D90FE;

-webkit-box-shadow: 0px 0px 5px #4D90FE;

box-shadow: 0px 0px 5px #4D90FE;

}

Later we will see how to extract these into theme because many of these style here like the background : white etc, this is sort of a default component theme that is embedded here in idle of other styles that are not necessary theme related.

Right now lets see how we can we have input-focus class on host element(fa component) depending upon value of focus variable. Simplest way to do is to use HostBinding decorator, but this time at level of au-fa-input component itself. So we add this decorator to method which determines whether input is focussed or not. So we define a getter. It needs to return an input property true or false, that is then going to activate or remove class in host element.

@HostBinding('class..input-focus') get isInPutFocus() {

return this.input ? this.input.focus : false;

}

So we are going to define here that class is going to be called input-focus, like we defined in css. With this how we can implement this. We are going to use this.input, then we see if that input is present because, if something goes wrong may be this input is undefined, so we want to guard against this condition. For ex if user accidentally forgot to add the input to font-awesome component, we dnt want our application to crash. So we check here if input is present , if yes, then we add or remove the class depending upon on truthfulness of focus member variable.

Now in ngAfterContentInit we want to add some functionality to make sure that input is present and if it’s not then we don’t to abort the application, but we want to inform developer that component is not correctly initilaized.

ngAfterContentInit() {

if (!this.input) {

console.error('the au-input needs an input inside its content');

}

}

Au-fa-input.component.ts-

import { Component, OnInit, ViewEncapsulation, Input, ContentChild, AfterContentInit, HostBinding } from '@angular/core';

import { InputRefDirective } from 'app/lib/common/input-ref.directive';

@Component({

selector: 'au-fa-input',

templateUrl: './au-fa-input.component.html',

styleUrls: ['./au-fa-input.component.css'],

encapsulation: ViewEncapsulation.Emulated

})

export class AuFaInputComponent implements OnInit, AfterContentInit {

@Input() icon: string;

@ContentChild(InputRefDirective) input: InputRefDirective;

constructor() { }

ngOnInit() {

}

ngAfterContentInit() {

if (!this.input) {

console.error('The au-fa-input needs input inside its cotent');

}

}

@HostBinding('class.input-focus') get isInputFocus() {

return this.input ? this.input.focus : false;

}

get classes() {

const cssClasses = {};

if (this.icon) {

cssClasses['fa-' + this.icon] = true;

}

return cssClasses;

}

}

Now if we run our code we can see that focus functionality is working fine.

Lets now discus how to organize the styles in order to make the component easy to theme.lets produce an alternate theme to demonstrate that , lets show saas support that we can use via angular cli to build our components. Lets see how to override that styles if we need to.

We are going to create an ng module to wrap up our library and we are going to test the library. After that we are going to prepare the library for publication and deployment on npm.

Key take away is

1)use of getter in component

2)use of host binding in component

3)styling in angular, css selectors that angular provides

4)