17. Separating Structural Styles From Theme Styles - Making Components Themeable

Now we have working version of au-fa-input component. In css files of our component we have all the style which are applied to our component.

.icon {

width:20px;

text-align: center;

padding-left: 5px;

padding-right: 2px;

}

:host {

border: 1px solid lightgrey;

padding-bottom: 1px;

display: inline-block;

background: white;

}

:host(.input-focus) {

outline: none;

border: 1px solid #4D90FE;

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

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

}

:host ::ng-deep input {

border: none;

outline: none;

height: 100%;

box-sizing: border-box;

}

In this section we will explore component styling best practices. We will learn how to separate our styles into structural styles and theme styles. We are going to create a default theme for our component and we are going to see how to add an extra theme that we can ship togather with library. Then we are going to see how to override component css if needed and we are going to talk about multiple mode of style isolation. This section is all about component styling.

As starting point we have css of our component that is applied on it. We are going to start refactoring the style to split it up into styles that are clearly associated to overall theme for example like background of input box(whole component has white background), or color of input box when input has focus.so everything that is related to colors , we are going to isolate on separate style. In order to help with this refactoring we have split border property into 3 properties border-width, border-style and border-color (in :host and :host (.input-focus) selector)-

.icon {

width: 20px;

background: white;

padding-left: 5px;

padding-right: 2px;

text-align: center;

}

:host {

/\* border: 1px solid lightgrey; \*/

border-width: 1px;

border-style: solid;

border-color: lightgrey;

padding-bottom: 2px;

padding-top: 1px;

display: inline-block;

background: white;

}

:host(.input-focus) {

outline: none;

/\* border: 1px solid #4D90FE; \*/

/\* border-width: 1px;

border-style: solid; \*/

border-color: #4D90FE;

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

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

}

:host ::ng-deep input {

border: none;

outline: none;

height: 100%;

box-sizing: border-box;

}

Then we can see that border-width and border-style are duplicated with both selectors. So we can them from :host(.input-focus) selector. lets now start isolating everything related to colors.

Then we take color proeprties of :host and :host(.input-focus) , place them separately at top od css file. We cut paste them.

au-fa-input.component.css-

:host {

background: white;

border-color: lightgrey;

}

:host(.input-focus) {

border-color: #4D90FE;

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

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

}

.icon {

width: 20px;

background: white;

padding-left: 5px;

padding-right: 2px;

text-align: center;

}

:host {

/\* border: 1px solid lightgrey; \*/

border-width: 1px;

border-style: solid;

/\* border-color: lightgrey; \*/

padding-bottom: 2px;

padding-top: 1px;

display: inline-block;

/\* background: white; \*/

}

:host(.input-focus) {

outline: none;

/\* border: 1px solid #4D90FE; \*/

/\* border-width: 1px;

border-style: solid; \*/

/\* border-color: #4D90FE;

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

box-shadow: 0px 0px 5px #4D90FE; \*/

}

:host ::ng-deep input {

border: none;

outline: none;

height: 100%;

box-sizing: border-box;

}

Now we have separated 2 types of styles, let’s create here couple of files where we will put them. In au-fa-input folder we create common.css file. This new file will have all stytles that are structural to this component(that is styles art from theme styles). So these styles here would be common to all inut boxes that we are going to be building in this small library.

Common.css-

.icon {

width: 20px;

background: white;

padding-left: 5px;

padding-right: 2px;

text-align: center;

}

:host {

/\* border: 1px solid lightgrey; \*/

border-width: 1px;

border-style: solid;

/\* border-color: lightgrey; \*/

padding-bottom: 2px;

padding-top: 1px;

display: inline-block;

/\* background: white; \*/

}

:host(.input-focus) {

outline: none;

/\* border: 1px solid #4D90FE; \*/

/\* border-width: 1px;

border-style: solid; \*/

/\* border-color: #4D90FE;

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

box-shadow: 0px 0px 5px #4D90FE; \*/

}

:host ::ng-deep input {

border: none;

outline: none;

height: 100%;

box-sizing: border-box;

}

Au-input.compoennt.css-

:host {

background: white;

border-color: lightgrey;

}

:host(.input-focus) {

border-color: #4D90FE;

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

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

}

There are some styles here that , it’s kind of unclear if they belong to the structure of the component and not defult theme, such as the width of the border etc. it’s compromise, in most cases we are not going to change the width of input .so it’s always possible to override the styles if needed. But what we are doing here is we are identifying what are the styles that are clearly associated to a theme, that we would want alomost for sure to override in almost any web site that we use the component in. so here we have separetd 2 kinds of styles. Now in au-fa-input.css ,we import this common.css file.

@import "common.css";

:host {

background: white;

border-color: lightgrey;

}

:host(.input-focus) {

border-color: #4D90FE;

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

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

}

Now we create another file called au-fa-input-default-theme.css. In this file we cut paste in all the styles which belong to default theme(we cut paste them au-fa-input.component.css) which our component will have. Basically all color related styles.

So idea of having a default theme is, if someone uses our library and simple adds au-fa-input selector to his application then input will look like what we saw n last section. Then we import this au-fa-input-default-theme.css file in css file of our component.

au-fa-input-default-theme.css -

:host {

background: white;

border-color: lightgrey;

}

:host(.input-focus) {

border-color: #4D90FE;

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

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

}

Au-fa-input.componnent.css-

@import "common.css";

@import 'au-fa-input-default-theme.css';

Commn.css-

.icon {

width: 20px;

background: white;

padding-left: 5px;

padding-right: 2px;

text-align: center;

}

:host {

/\* border: 1px solid lightgrey; \*/

border-width: 1px;

border-style: solid;

/\* border-color: lightgrey; \*/

padding-bottom: 2px;

padding-top: 1px;

display: inline-block;

/\* background: white; \*/

}

:host(.input-focus) {

outline: none;

/\* border: 1px solid #4D90FE; \*/

/\* border-width: 1px;

border-style: solid; \*/

/\* border-color: #4D90FE;

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

box-shadow: 0px 0px 5px #4D90FE; \*/

}

:host ::ng-deep input {

border: none;

outline: none;

height: 100%;

box-sizing: border-box;

}

Now our component is working as it was working before but now styles are separated.

With this approach it is very clear what styles we need to provide to create a new theme. Theme might be provided by third party application but we can also ship our own themes togather with libarary that we build.

Next we are going to learn how to create an alternative theme and how to apply it to our application.

18. How To Create An Alternative Component Theme And Ship It With The Library

Here we will see how to create an alternative theme to our component. As we saw in last lecture we are already providing the default theme in au-fa-input-defalut-theme.css file.

Now lets create a separate css file that will contain an alternate theme that we are going to ship togather with our library. So create this file- **au-fa-input-red-theme.css.**

In order to understand how this alternate theme will work, let’s start by seeing how we will use the theme itself. In our application if we simply use this-

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

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

</au-fa-input>

Then default theme will be applied. but in lets say we want all the inputs to have an alternate theme. Then will go to wrapping element of application may be body but usually this will be div, like here we have div with class container(in app.component.html). To this div, we will add new css class, what this css class says is that any au-fa-input inside this wrapping element,is going to receive red theme instead of default theme.

<div class="container" au-fa-input-red-theme>

</div>

So lets see how we can use this css class to style further our component. Because as we know the style that we add in css file of this component(au-fa-input), because they are added in level of component, they are going to be made specific to component. So the whole goal is to have style isolation that we are going to talk more in future lesson

Right now how can we style our component depending upon on some css class that is outside of it. well that is use case for :host-context selector.

au-fa-input-red-theme.css-

:host-context(.au-fa-input-red-theme) {

border-color: red;

}

We are going to say that if some wrapping element of input in the context where input is being used.(so this is not inside component itself, this is the outside component that is using this input) if there is a wrapping element that contains this class, then these styles will be applied, but only to component itself. So, we do not need to add :host psudo selector in our selector.like this-

**:host-context(.au-input-red-theme) : host {**

**}**

These styles will only be applied to component itself.

In au-fa-input-red-theme.css-

:host-context(.au-fa-input-red-theme) {

border-color: red;

}

then apply these styles to my component. In au-fa-input.component.css-

@import 'common.css';

@import 'au-fa-input-default-theme.css';

@import 'au-fa-input-red-theme.css';

We import this css file.

In in app.component.html we add his class to wrapping element of our font awesome-

<div class="container au-fa-input-red-theme">

<h1>Font Awesome</h1>

<div class="form">

<div class="form-row">

<label>FA Input:</label>

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

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

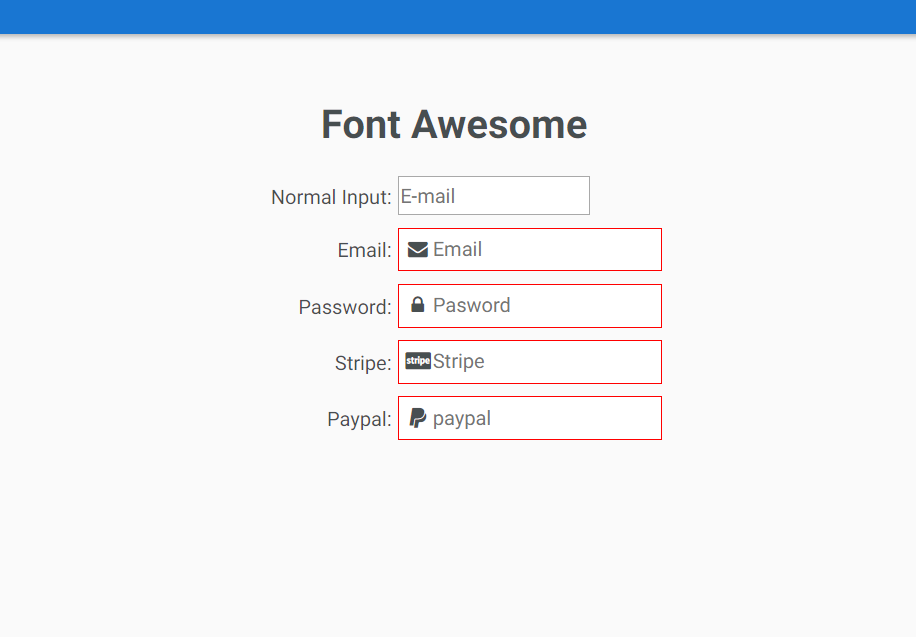
</au-fa-input>

</div>

</div>

Now we can see that borders of our component is of red color. So our style are being applied depending upon class of wrapping element.

Output-



Additional information(not covered in lecture)-

To override all colors, excat theme file should be like this(that equivalent selectors when parent has some class applied to it, is)-

au-fa-input-red-theme.css

:host-context(.au-fa-input-red-theme) {

background: black;

border-color: green;

}

:host-context(.au-fa-input-red-theme).input-focus {

border-color: red;

-webkit-box-shadow: 0px 0px 5px red;

box-shadow: 0px 0px 5px red;

}

**Refer to notes about these selectors.**

And with this we have initial implementation of theme, we are going to finish it in next lectures, where we will also look into what scss integration possibilities, angular cli brings us, that will help further to make our components more themable.  
19. Creating an Alternative Component Theme, See the CLI SaaS Support In Action

Here we will cover angular-cli sass integration.we are going to look at several features of saas and we are going to see all those features help us to write more readable and maintainable styles for our components.

Now lets try to add the situation where we have input-focus added to our au-fa-input component so we add this to our red theme css class-

:host-context(.au-fa-input-red-theme).input-focus {

-webkit-box-shadow: 0px 0px 5px red;

box-shadow: 0 px 0px 5px red;

}

This style will be added to our component only if it’s enclosing elment has this class-(.au-fa-input-red-theme and element intslf has this class- ).input-focus .We have not left any space between **.input-focus** and **:host-context()**. If we leave space , like this-

:host-context(.au-fa-input-red-theme) .input-focus {

-webkit-box-shadow: 0px 0px 5px red;

box-shadow: 0 px 0px 5px red;

}

It means we want to apply styles to child element of host element. If we want to apply it to host element then we need to remove space. Now our theme is-

:host-context(.au-fa-input-red-theme) {

border-color: red;

}

:host-context(.au-fa-input-red-theme) .input-focus {

-webkit-box-shadow: 0px 0px 5px red;

box-shadow: 0 px 0px 5px red;

}

Now we are having little repetition. We are writing :host-context(.au-fa-input-red-theme) 2 times. If we want to improve the readability of our styles we can use SAS integration of the angular cli. To do this we simply have to rename au-fa-input-red-theme.css to \_au-fa-input-red-theme.scss. we can also make it scss partial by adding \_ before name to follow the conventions of scss .you also need to changes import in component css file.

What are scss partials-

<https://dev.to/sarah_chima/using-sass-partials-7mh>

now everything is working fine as expected. But now red-theme file is scss file. Now we can use scss feature in this file.

Lets define a variable for color. We simply say that border color for this theme will be red. The we replace all ocuurences of red color with this variable.

$border-color: red;

:host-context(.au-fa-input-red-theme) {

border-color: $border-color;

}

:host-context(.au-fa-input-red-theme).input-focus {

-webkit-box-shadow: 0px 0px 5px $border-color;

box-shadow: 0 px 0px 5px $border-color;

}

You need to change your main css file to scss also.

This is one feature of scss, we can define variable. Another feature is nested styles , which usually we do not need when we are defining styles inside a component because these styles are very specific. But in our case here we are repeating :host-context(.au-fa-input-red-theme), so it would make them more clear if we neste them like this-

$border-color: red;

:host-context(.au-fa-input-red-theme) {

border-color: $border-color;

.input-focus {

border-color: $border-color;

-webkit-box-shadow: 0px 0px 5px $border-color;

box-shadow: 0px 0px 5px $border-color;

}

}

// :host-context(.au-fa-input-red-theme).input-focus {

// border-color: $border-color;

// -webkit-box-shadow: 0px 0px 5px $border-color;

// box-shadow: 0px 0px 5px $border-color;

// }

We have child style inside parent style. Now bcoz we dnt want to apply input-focus to child elements of :host-context(.au-fa-input-red-theme) but to elemnt itself we we use **&** in front of child style, like this-

$border-color: red;

:host-context(.au-fa-input-red-theme) {

border-color: $border-color;

&.input-focus {

border-color: $border-color;

-webkit-box-shadow: 0px 0px 5px $border-color;

box-shadow: 0px 0px 5px $border-color;

}

}

// :host-context(.au-fa-input-red-theme).input-focus {

// border-color: $border-color;

// -webkit-box-shadow: 0px 0px 5px $border-color;

// box-shadow: 0px 0px 5px $border-color;

// }

This will create selector similar to what we are having before.

This is similar to css we were using before, now our code is working fine.

This is small example of how we can use scss and its multiple feature to better style our components. Next we are going to have deeper look into how component styling mechanism works, how isolations of styles work and what are different isolation modes that we can have for creating angular components..how what are different isolation modes that we can have for our styles.

20. Angular Style Isolation - Emulated View Encapsulation - Learn How It Works

Lets talk about style isolation. Lets start by understanding how the default style isolation mechanism of angular works under the hood. This is going to help us a lot to troubleshoot the stylig of our compoennts and its going to help us a lot leverage the style isolation mechanism. Because it is very different from normal way css works. So lets learn about emulates encapsulation mode, which is default style isolation mechanism of angular.

Now in our component we have default theme enabled in all au-fa inputs except password field.

App.component.html-

<div class="form-row au-fa-input-red-theme">

<label>Password:</label>

<au-fa-input [icon]="'lock'">

<input placeholder="Pasword">

</au-fa-input>

</div>

Lets see how could we always overwrite the styles of stripe input if we needed to. To do that we are going to proceed normally we we would,if we want to overwrite some existing style. Normal way is to provide more specific style.

So to div containing this style we add a class stripe-field.

<div class="form-row stripe-field ">

<label>Stripe:</label>

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

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

</au-fa-input>

</div>

Now lets go to css file of app component. There we add a selector-

.stripe-field au-fa-input {

border: 1px solid blue;

}

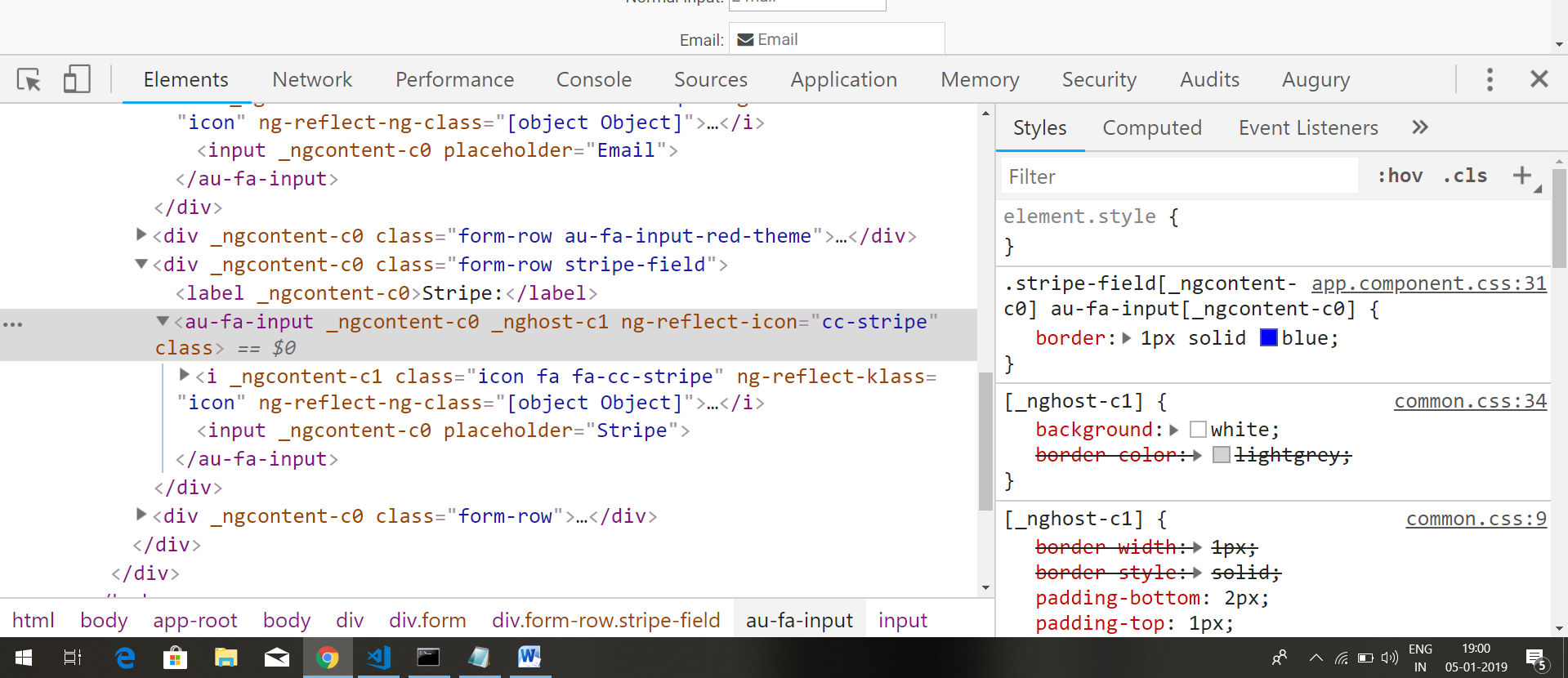
So now style are applied. but this style will be applied only to au-fa-input component that are in app.component.html, this is because we added this selector to css file of app.component. this style will be added in index.html-

.stripe-field[\_ngcontent-c0] au-fa-input[\_ngcontent-c0] {

border: 1px solid blue;

}

This is html-



We can see why that style was applied. we had 2 selectros for border but more specific was applied(in right hand side of above screenshot).

This style will be applied only to those au-fa-input compoeonts that are directly in html of app.component. lets say a au-input component is deeply nested inside app.component. then tis style wnt be applied to that.

If we try to style elments that are deeply nested in other child components of app.component then this style would not work. Lets see why.

It is because every element that is inside app.component.html, will have a unique attribute added to it. This is some cryptographic attribute. All the styles defined in app.component.css will have selector based on this cryptographic attribute. For exif we have defined this style in css of app.component-

h1 {

}

Then in final bundeled index.html, it will be added in styles sections. But it would be this-

h1 [crypto\_attribute] {

}

This makes sure that styles isolated from rest of components style. For details see lecture 13.

this type of style isolation is already very effective. We can create components that will be deployed on third party pages where we do not have control over styles of page and we can be fairly sure that our styles will not interfere with existing styles on that page.

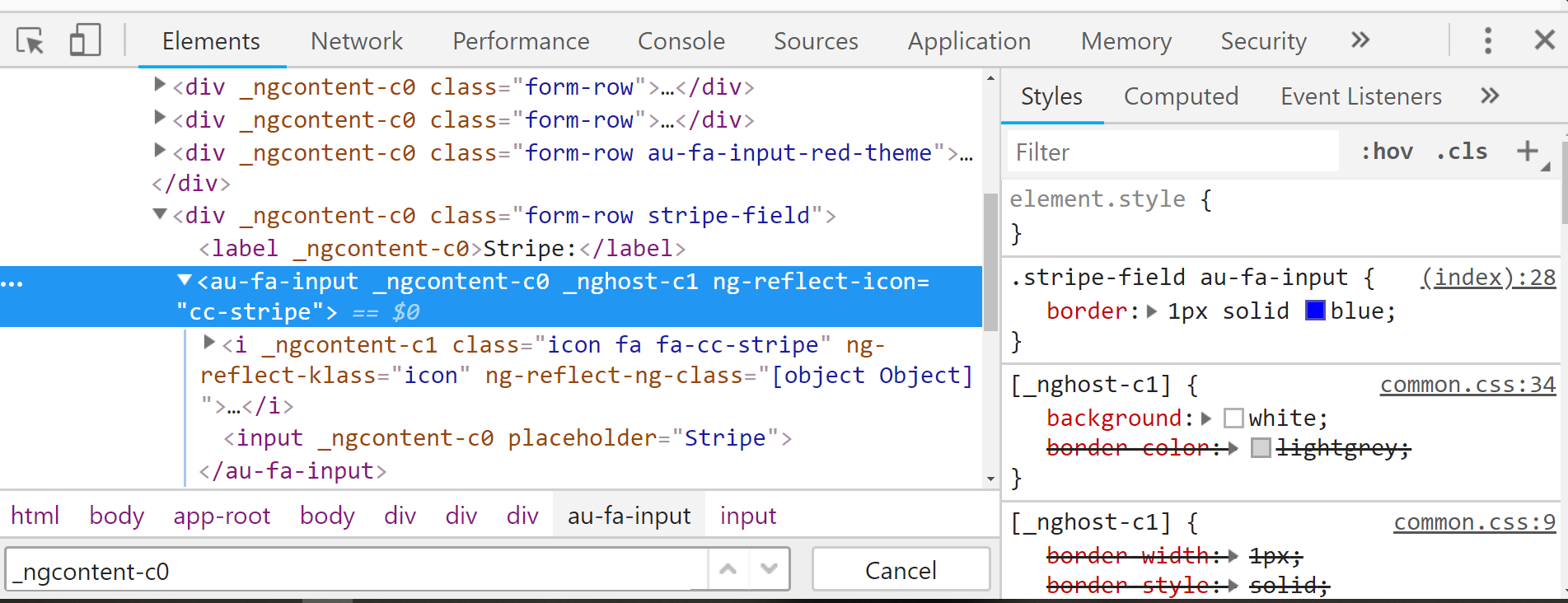
But this level of style islolation that is based on adding attribute to style is not completely bulletproof. In practice it will almost 100% work. But lets say for ex if we take a style from css of application component and we place it in index.html or if we simply load this style will external stylesheet, this style will be applied to whole application. For ex, if we place this style to index.html, it will still work-

.stripe-field au-fa-input {

border: 1px solid #4D90FE;

}

Still our stripe component receives blue border. This is why-



So this type of style isolation which is called **EmulatedViewEncapsulation** which is default view encapsulation of angular components.,This can still be overridden by third party stylesheet. The **EmulatedViewEncapsulation** mechanism is one that we will most likely be using but there are couple of other mechanism that we have avalible. Lets look at them.

We are going to see native ShadoeDOM functionality of chrome browser in action.

21. Learn 2 More Alternative Ways of Handling CSS in Angular Applications

Native view encapsulation is full style encapsulation. If we use hat even browser defaults will not be applied.

We have already covered this in angular 4. See notes

22. Setting Up A Library Module, Confirming AOT is Supported

Before starting to add testing to our library , lets do a lesson on creating library module and just make sure that everything is compatible with AOT. So in lib folder we create a module file. To module we give the same name as NPM package that we will be publishing. So name this file-

au-input.module.ts

in this file we simply create our module. We import CommonModule. Common Module. Common module is almost needed in every module. It gives us access to common directives like ngclass,ngModule ,ngfor,ngif etc.chances are there that almost all modules use this directive so we need to include this common directive.

In declaration section, we declare our component and directive. then in exports section again we add our component and directive. So any module which imports our module will be able to process these 2 directives.

au-input.module.ts-

import { CommonModule } from '@angular/common';

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

import { AuFaInputComponent } from 'app/lib/au-fa-input/au-fa-input.component';

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

@NgModule({

declarations: [AuFaInputComponent, InputRefDirective],

imports: [CommonModule],

exports: [AuFaInputComponent, InputRefDirective]

})

export class AuInputModule {}

now lets use our module in app.component. we remove component and directive that we have created from declaration section of app.module. lets use our module to access them. So we import our module in app.module. app.module-

imports: [

BrowserModule,

FormsModule,

HttpModule,

AuInputModule

]

If we run our application everything works as it was working before. Now we are going to make sure that what we have developed so far is compatible with AOT.

So we serve the application in aot mode.

**ng serve –prod –aot**

this build will take little bit longer, because we are compiling our app in aot mode. We are making sure that our app supports this mode. Once compilation successfully ends. We can load our application. We can notice that now loading of application is much faster, that’s because aot bundle is much smaller.we are going to see that everything works as it was before.

Now we are sure that our component supports the aot mode. We are going to do one last preparation step to this library which is to go to root folder of lubrary (au-inpt, which has src folder) and in order to prepre this for deployment we are going to create an index.ts file. In this file we export the module that we have just created, so that it can be easily iported by other application. Index.ts-

export {AuInputModule} from './src/app/lib/au-input.module';

so with these we will be able to very easily import the the input module into another application. The import will look like this-

**import {AuInputModule} from ‘au-input’;**

so,In target program it will be reading from index file , so this will be-

**import {AuInputModule} from ‘au-input/index’;**

so with this last preparation step is in place. Let’s start testing this library.

Lets start by making sure that our test environment is working correctly.

23. Angular Component Testing Overview

Lets start testing. So how do we approach the testing of component like this? Well essential thing to bear in mind is that we need to do an integration test to actually test the angular component, because if we simply instantiate the class using the new operator there is now way to test the functionality involved. We need a browser, we need aDOM so that template gets applied etc. so lets see what would be a good testing startergy for this particular component. So bcoz essential about testing a component is that, it’s an integaration based testing, the simpleset is to use the same sample application that we have been suing to demonstrate the component.. we are going to add testing on top of it. So lets have a look at it.the way that we would do it is we go to test application that we have been running and we are simply going to add couple of identifiers and css classes to help with testing. For ex we will be giving to id to our component selectors , so that we can easily access them in our test.

<au-fa-input id = "password-field" icon="lock" >

<input type="text" #input placeholder="password" class=”test-class”>

</au-fa-input>

We are also going to add a css class, because we are going to make sure that this input is projected inside our component. This class is test-class. It is added to input. So based on this we are ready to start write some test that are going to check is component being rendered correctly, is input elemnt being detected and projected inside our component, are correct css classes are being added to font-awesome icon and is icon present. So this will be strategy that we will adopt for view layer of our component. In next lecture we will see what tests look like.

24. The Angular CLI Testing Infrastructure - Running Our First Test

In order to writing test for ur library, lets have a look at what test infrastructure angular cli provide us working out of box.

We are going to take starting point, the jasmine configuration that was created by angular cli automatically. So lets see it line by line, it is jasmine test.

import { TestBed, async } from '@angular/core/testing';

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

describe('AppComponent', () => {

beforeEach(async(() => {

TestBed.configureTestingModule({

declarations: [

AppComponent

],

}).compileComponents();

}));

it('should create the app', async(() => {

const fixture = TestBed.createComponent(AppComponent);

const app = fixture.debugElement.componentInstance;

expect(app).toBeTruthy();

}));

});

Jasmine is testing framework.

Rest all are testing lecture, please see videos. From here