## Templates, Styles & View Encapsulation

**Learning Objectives**

We’ve covered the basics of the @Component decorator in the quickstart. We explained how decorators work and both the template and selector configuration properties of the @Component decorator.

In this lecture we will go through a number of other configuration properties including templateUrl, styles, styleUrls and encapsulation.

In the section on Dependency Injection we will cover two other ways of configuring Components with the providers and viewProviders properties.

**templateURL**

We don’t have to write our template code inline with our component code. We can store our HTML template files separately and just refer to them in our component by using the templateUrl property.

Using the joke application we built in the quickstart, lets move the template for the JokeFormComponent to a file called joke-form-component.html, like so:

@Component({

selector: 'joke-form',

templateUrl: 'joke-form-component.html'

①

})

class JokeFormComponent {

@Output() jokeCreated = new EventEmitter<Joke>();

createJoke(setup: string, punchline: string) {

this.jokeCreated.emit(new Joke(setup, punchline));

}

}

① We point our component to an external template file by using the templateUrl property.

**styles**

We can also specify any custom css styles for our component with the styles property.

styles takes an *array of strings* and just like template we can use multi-line strings with back-ticks.

Let’s define a style for the JokeFormComponent so it gives it a background color of gray.

@Component({

selector: 'joke-form',

template: 'joke-form-component.html',

styles: [

`

.card {

background-color: gray;

}

`

],

})

class JokeFormComponent {

@Output() jokeCreated = new EventEmitter<Joke>();

createJoke(setup: string, punchline: string) {

this.jokeCreated.emit(new Joke(setup, punchline));

}

}



The

styles

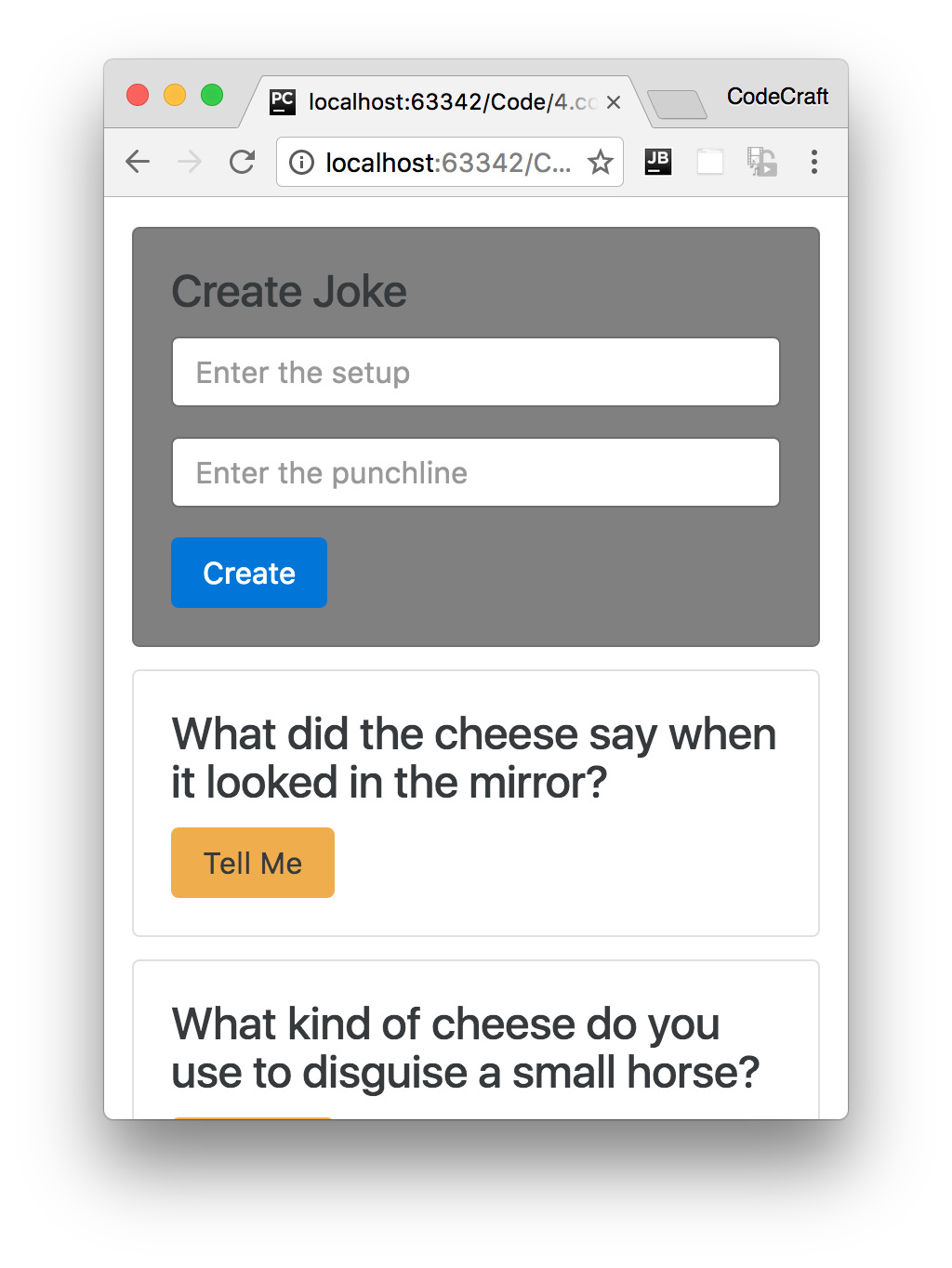
property above takes an

**array**

of strings, each string can contain any

number of CSS declarations.

The form component in our application now turns gray, like so:



**View Encapsulation**

Even though we changed the background color of .card and we have multiple cards on the page only the form component card was rendered with a gray background.

Normally if we change a css class the effect is seen throughout an application, something special is happening here and it’s called *View Encapsulation*.

Angular is inspired from Web Components, a core feature of which is the shadow DOM.

The shadow DOM lets us include styles into Web Components without letting them *leak* outside the component’s scope.

Angular also provides this feature for Components and we can control it with the encapsulation property.

The valid values for this config property are:

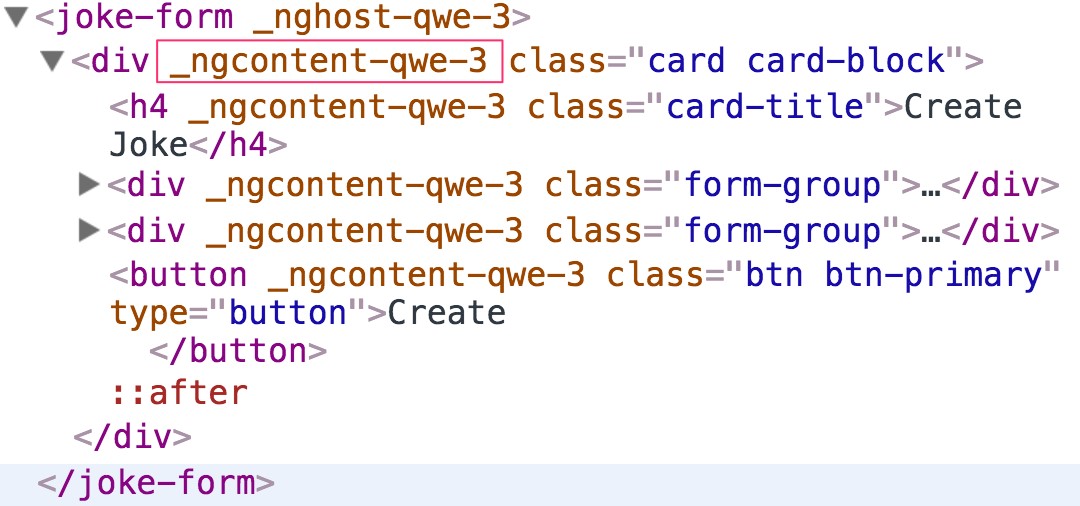
* ViewEncapsulation.Native
* ViewEncapsulation.Emulated
* ViewEncapsulation.None.

The default value is ViewEncapsulation.Emulated and that is the behaviour we are currently seeing.

### ViewEncapsulation.Emulated

Let’s inspect the form element with our browsers developer tools to investigate what’s going on.

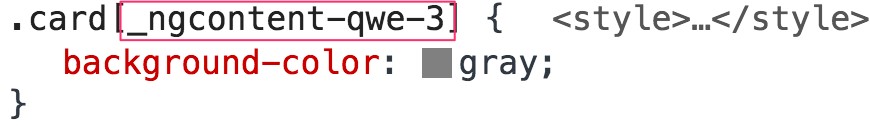
By looking at the DOM for our JokeFormComponent we can see that Angular added some *automatically* generated attributes, like so.



Specifically it added an attribute called \_ngcontent-qwe-3.

The other components on the page don’t have these automatically generated attributes, only the JokeFormComponent which is the only component where we specified some styles.

Again by looking at the styles tab in our developer tools we can see a style is set for \_ngcontent-qwe3 like so:





The css selector

.card[\_ngcontent-qwe-3]

targets

*only*

the

JokeFormComponent

since

that is the only component with a html attribute of

\_ngcontent-qwe-3

.

In the ViewEncapsulation.Emulated mode Angular changes our generic css class selector to one that target just a single component type by using automatically generated attributes.

This is the reason that *only* the JokeFormComponent is gray despite the fact that we use the same card class for all the other JokeComponents as well.

Any styles we define on a component *don’t leak out* to the rest of the application but with ViewEncapsulation.Emulated our component still inherits global styles from twitter bootstrap.

Our JokeFormComponent still gets the global card styles from twitter bootstrap and the encapsulated style from the component itself.

### ViewEncapsulation.Native

If we want Angular to use the *shadow DOM* we can set the encapsulation parameter to use ViewEncapsulation.Native like so:

@Component({

selector: 'joke-form',

templateUrl: 'joke-form-component.html',

styles: [

`

.card {

background-color: gray;

}

`

],

encapsulation: ViewEncapsulation.Native # <!>

})

class JokeFormComponent {

@Output() jokeCreated = new EventEmitter<Joke>();

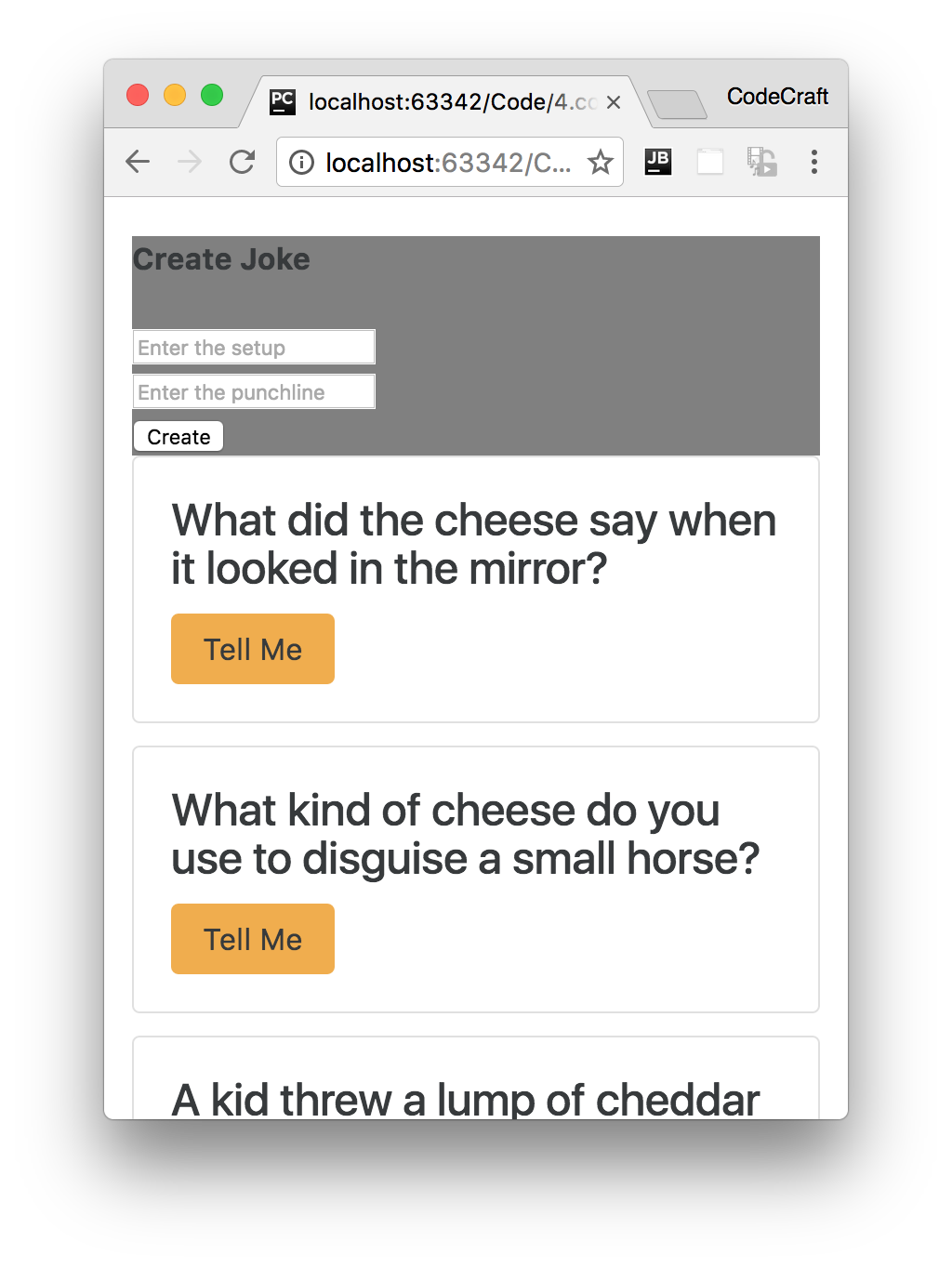
createJoke(setup: string, punchline: string) {

this.jokeCreated.emit(new Joke(setup, punchline));

}

}

But now if we look at the application although the background color of the JokeFormComponent is still gray, we’ve *lost* the global twitter bootstrap styles.



With ViewEncapsulation.Native styles we set on a component *do not leak outside* of the components scope. The other cards in our application do not have a gray background despite the fact they all still use the card class.

This is great if we are defining a 3rd party component which we want people to use in isolation. We can describe the look for our component using css styles without any fear that our styles are going to leak out and affect the rest of the application.

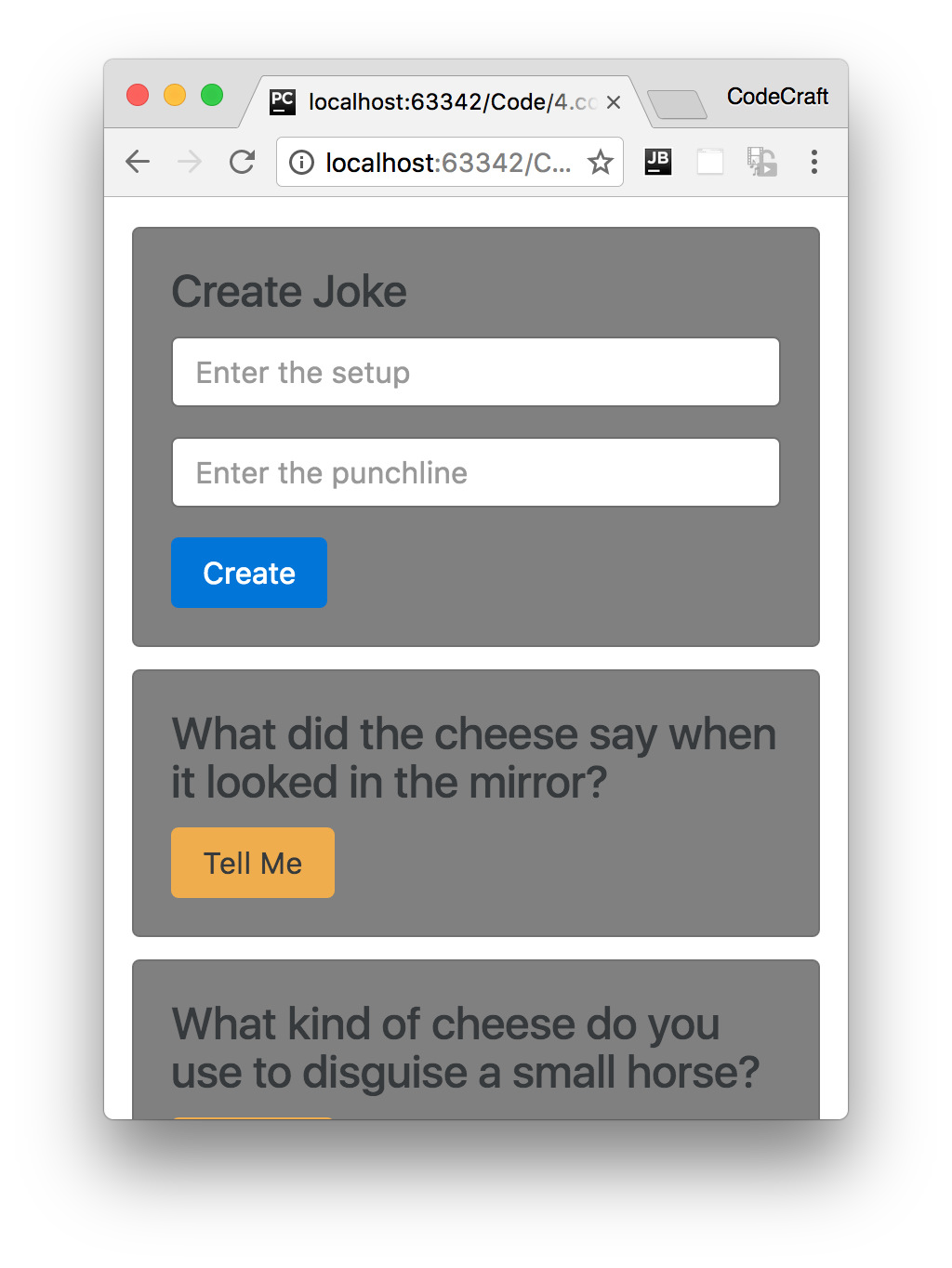
However with ViewEncapsulation.Native our component is also isolated from the global styles we’ve defined for our application. So we don’t inherit the twitter bootstrap styles and have to define all the required styles on our component decorator.

Finally ViewEncapsulation.Native requires a feature called the *shadow DOM* which is not supported by all browsers.

### ViewEncapsulation.None

And If we don’t want to have any encapsulation at all, we can use ViewEncapsulation.None.

The resulting application looks like so:



By doing this all the cards are now gray.

If we investigate with our browser’s developer tools we’ll see that Angular added the .card class as a *global style* in the head section of our HTML.



We are not encapsulating anything, the style we defined in our card form component has leaked out and started affecting the other components.

**styleURLs**

Like the templateUrl property, with the styleUrls property we can externalise the css for our component into another file and include it in.

However like the styles parameter, the styleUrls param takes an *array* of css files, like so:

@Component({

selector: 'joke-form',

templateUrl: 'joke-form-component.html',

styleUrls: [

'joke-form-component.css'

]

})

class JokeFormComponent {

@Output() jokeCreated = new EventEmitter<Joke>();

createJoke(setup: string, punchline: string) {

this.jokeCreated.emit(new Joke(setup, punchline));

}

}

**Depreciated Properties**

If you have seen code that discusses the additional @Component properties;

directives, pipes, inputs and outputs these were in the *beta* version of Angular and

were removed in the final 2.0 release. So the information you’ve read is

unfortunately outdated.

**Summary**

We can externalise our HTML template into a separate file and include it in with the templateUrl property.

We can also define styles for our component via the styles and styleUrls property.

By default styles for our components are *encapsulated*, that means that they don’t *leak* out and affect the rest of the application.

We can explicitly set the encapsulation strategy using the encapsulation property.

By default, the renderer uses ViewEncapsulation.Emulated if the view has styles, otherwise ViewEncapsulation.None. There is also a ViewEncapsulation.Native method which uses the *shadow DOM* to encapsulate the view.

**Listing**

<http://plnkr.co/edit/2MUcs44WUo2G1cS2JvX4?p=preview>

*index.html*

<!DOCTYPE html>

<!--suppress ALL -->

<html>

<head>

<link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0alpha.4/css/bootstrap.min.css">

<script src="https://unpkg.com/core-js/client/shim.min.js"></script>

<script src="https://unpkg.com/zone.js@0.7.4?main=browser"></script>

<script src="https://unpkg.com/reflect-metadata@0.1.8"></script>

<script src="https://unpkg.com/systemjs@0.19.39/dist/system.src.js"></script>

<script src="systemjs.config.js"></script>

<script>

System.import('script.ts').catch(function (err) {

console.error(err);

});

</script>

</head>

<body class="container">

<app>Loading...</app>

</body>

</html>

*script.ts*

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

import {Component, NgModule, Input, Output, EventEmitter, ViewEncapsulation} from

'@angular/core';

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

class Joke {

public setup: string;

public punchline: string;

public hide: boolean;

constructor(setup: string, punchline: string) {

this.setup = setup;

this.punchline = punchline;

this.hide = true;

}

toggle() {

this.hide = !this.hide;

}

}

@Component({

selector: 'joke-form',

templateUrl: 'joke-form-component.html',

styleUrls: [

'joke-form-component.css'

],

encapsulation: ViewEncapsulation.Emulated

// encapsulation: ViewEncapsulation.Native

// encapsulation: ViewEncapsulation.None

})

class JokeFormComponent {

@Output() jokeCreated = new EventEmitter<Joke>();

createJoke(setup: string, punchline: string) {

this.jokeCreated.emit(new Joke(setup, punchline));

}

}

@Component({

selector: 'joke',

template: `

<

div class="card card-block"

>

<h4 class="card-title">{{data.setup}}</h4>

<p class="card-text"

[hidden]="data.hide">{{data.punchline}}</p>

<a (click)="data.toggle()"

class="btn btn-warning">Tell Me

</a>

<

/div

>

`

})

class JokeComponent {

@Input('joke') data: Joke;

}

@Component({

selector: 'joke-list',

template: `

<

joke-form (jokeCreated)="addJoke($event)"></joke-form

>

<

joke \*ngFor="let j of jokes" [joke]="j"></joke

>

`

})

class JokeListComponent {

jokes: Joke[];

constructor() {

this.jokes = [

new Joke("What did the cheese say when it looked in the mirror?", "Hello-me

(

Halloumi)"),

new Joke("What kind of cheese do you use to disguise a small horse?", "Mask-a-

pony (Mascarpone)"),

new Joke("A kid threw a lump of cheddar at me", "I thought

‘

That

’

s not very

mature

’

"),

];

}

addJoke(joke) {

this.jokes.unshift(joke);

}

}

@Component({

selector: 'app',

template: `

<

joke-list></joke-list

>

`

})

class AppComponent {

}

@NgModule({

imports: [BrowserModule],

declarations: [

AppComponent,

JokeComponent,

JokeListComponent,

JokeFormComponent

],

bootstrap: [AppComponent]

})

export class AppModule {

}

platformBrowserDynamic().bootstrapModule(AppModule);

*joke-form-component.css*

.card {

background-color: gray;

}

*joke-form-component.html*

<

div class="card card-block"

>

<h4 class="card-title">Create Joke</h4>

<div class="form-group">

<input type="text"

class="form-control"

placeholder="Enter the setup"

#setup>

</div>

<div class="form-group">

<input type="text"

class="form-control"

placeholder="Enter the punchline"

#punchline>

</div>

<button type="button"

class="btn btn-primary"

(click)="createJoke(setup.value, punchline.value)">Create

</button>

<

/div

>