55) Want to use Angular2 instead of angular 4?

If you want to use Angular 2 instead of Angular 4, there are some tiny adjustments required.

**Here's an important note though**: I strongly recommend going with Angular 4! It's the latest version and won't break you Angular 2 code!

To use the Animation package in an Angular 2 app, you have to make sure that you import trigger , state  etc from @angular/core  instead of @angular/animations . You also **won't need to add** the BrowserAnimationsModule  to imports[] .

Besides that, the code should be the same.

56)Module Intdroduction

In this module we will see how angular animations works. When to use them and when to use css naimations and transition.

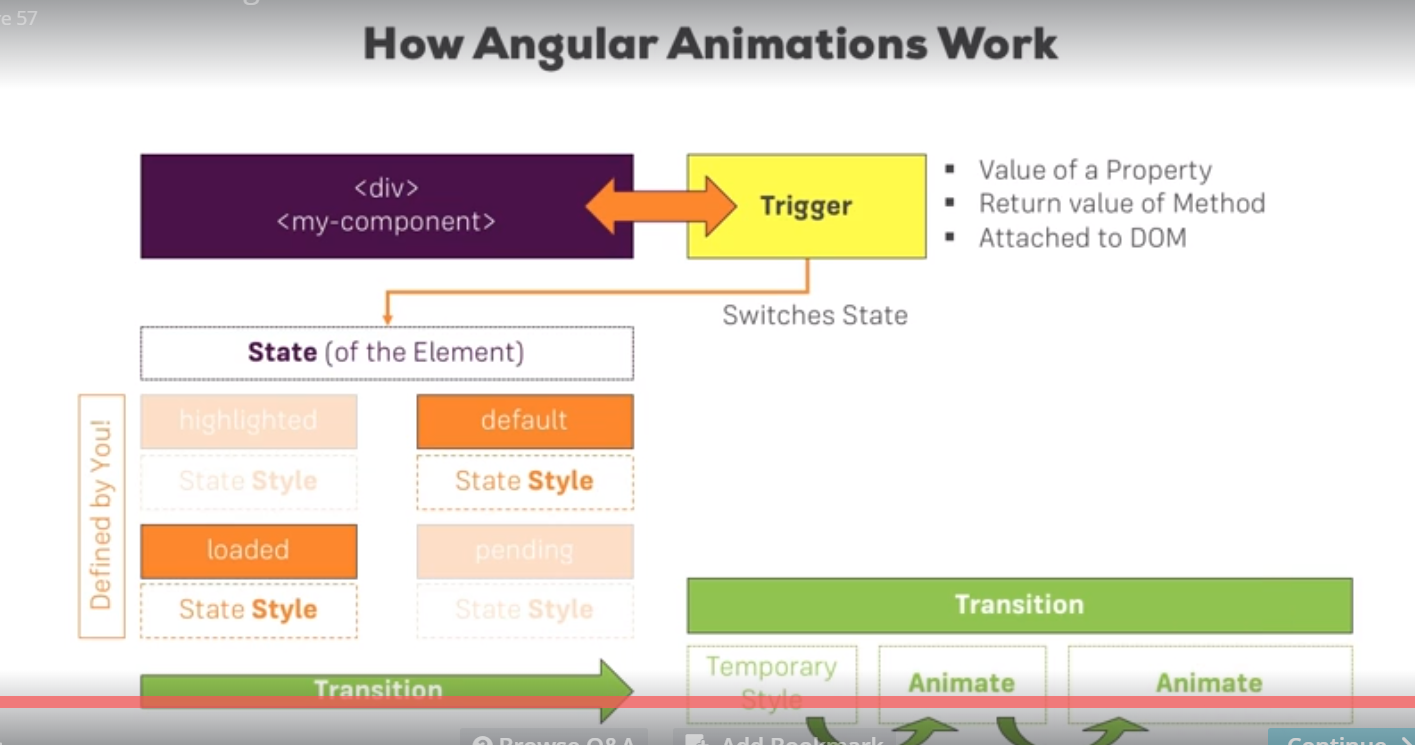
57)How Animations works in angular

It works like that- we can have any element in our template, it can be normal html element or our component selector. To animate this element or component we attach a trigger. A trigger is feature of angular’s animation library you can say. Trigger allows you to switch between states. A trigger is connected to for example,value of one of your properties defined in ts or return value of function, so it is basically connected to some value that yu connect to it.

Now,the trigger is then added to dom , to the template and therefore at run time kind of dom and trigger will detect whenever the data source which is connected to it (like the property value for example) when it changes. Why is that change important? Becaue the next step in angular animaton setup is that for this given trigger you define multiple states. Trigger can then switch between these states.

So you define states lfor ex, you can have highlighted state and each state has certain style attached to it. So highlighted can set border color to red for ex. Now the property for example,which is connected to trigger could be set to a string which says ‘highlighted’. So trigger will therefore set this highlighted state. We could have another state- defaulr for ex, and we could switch to a state by setting the property to which trigger is connected, to default and this state will have its own style- black border or no border , what ever you like. So that is how states work and you can switch between states, this is idea behind states and trigger is connected to some data source indicating which state should be current state.

Animation system does’nt stop here because switching between states and their respective styles would change appearance of elment but it won’t play any animations. We can define transition between states. You define that in transition you want to assume temporary style which only exists while transition is played which is powerful tool allowing you to attach styles which are not part of starting or ending states. We can also do this in css animations we can then also animate, that means, for example from this temporary state we animate to another animation sequence (with another temporary state maybe )then in the end we animate to final state.



58)Browser support and Polyfills

Angular animation system actually builds up on web animations api. Google ot – web animations api. In mozilla website you will find a article. You can read it see how things work in background. But important thing is it is still in draft. And not all browsers support it. To make it work in all browser we need to add polyfills.

Polyfill is basically a script package, you can say which allows you to use features which are not supported by browsers by detecting usage of these features and then catching therm basically and then writing some code your browser will understand. So if you are targeting browsers which dnt support this feature, you need to add polyfills in your project. Including them is very simple, go to polyfill.ts file in your project. Here you can say that some basuc polyfills are already included, but not for this animations package.

In that file you need to uncomment his line-

// import 'web-animations-js'

And run this comand-

npm install --save web-animations-js

this import statement will make sure that we add this polyfill to final bundle.

If you are not using cli, then steps for adding polyfill are same. First you need to download this polyfill(either by npm or from its official page), then you need to include a link to download file either to webpack or some other workflow which allows you to add imports like this by adding such a import or by adding a import in index.html file, pointing to js package. But make sure to import this before your angular code gets imported.

59)Unlocking Animations with Right Module

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

imports: [

BrowserModule,

FormsModule,

HttpModule,

AppRoutingModule,

BrowserAnimationsModule

]

Now @angular/animations might not be avalible in your default cli setup. In future this can be include in default setup. So install it-

npm install –-save @angular/animations.

60)Getting Started with Triggers and states

Here we define a div with width and give it width, height and bg color to make it visible. Now lets add a trigger to it, which is starting point to our animation journey.

Css-

#simple {

width: 100px;

height: 100px;

background-color: orange;

}

Html-

<div id ="simple"></div>

To use animation, we need to add new property to @Component decorator. This is animations property. This takes array as value, animations that we can define. What do we mean by animations that we can define? I basically mean trigger here because you can thing of one set of animations being contained in trigger because trigger will then define which part of these set of animations will be played. A set of animations can contain multiple states between which our can switch and trigger will determine which state is current state and then animations (also set up in trigger) will determine how to navigate or how to animate between these states.

So lets add a trigger. So we add a method trigger to animation array. It needs to be imported. trigger method takes 2 arguments. First is name of trigger, this is totally upto you. But since trigger will determine what the current state is , so putting state into trigger name makes sense. So we named our trigger as ‘clickedState’. This indicates with this trigger we can basically switch between different clicked states. For ex if we clicked on it, or if we double clicked it or whatever.

Second argument is array of animation metadata. It allows you to define your states with their respective styles and then transition between these states. So lets define some states. For this we use states method and we need to import it. It also takes 2 arguments, first one is name of state. Here we give it name of default for not clicked and something like that. Second argument is styles that you want to apply to states. To apply we have to use style method, we also need to import it. It takes a objects as argument, we can also pass array of objects. In this object we can define key value pair of css styles.so this is what we set up here- css styles. here can set up any style that you want. Be aware that all styles that you set might not be animatable, link to all animatable properties can be found after this lecture. Animatable properties means when you switch between 2 values of same property ,it is possible to play animation.

This is something that you have to keep in mind that if you are wondering why certain transition you want to work is not working- some css properties are not animatable. So we setup style for this sate. Hese styles are already applied to our div via id.

@Component({

selector: 'app-root',

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

styleUrls: ['./app.component.css'],

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

}))

])

]

})

So we defined styles for state. Regarding dimension units, if we dnt specify any units it taken to be %. If you want some other units, you have to mention it.

Now lets define another state. Name it click, name should specify when this state will be attached.

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

})),

state('clicked', style({

backgroundColor: 'blue',

width: '300px',

height: '50px'

}))

])

]

})

So our trigger is finished. A trigger can have multiple states, each state can has its own style.

61)Which properties are animatable

Not every CSS property may be animated but thankfully, the W3 maintains a list of animatable properties - have a look at it here: <https://www.w3.org/TR/css3-transitions/#animatable-properties>

62)Assignig triggers to elements in Template

Now we have trigger with 2 states, these 2 states has their own styles. lets use this trigger.

To use it go to html file.In html fist we remove id from our div. now it is not visible because it is not having bg color. Now lets attach our trigger to this div. Attaching a trigger is not enough, our trigger has 2 statets , how would it know which is current state. So we need to give state along with trigger. We do it by using property binding , so we pass state of trigger along with name of trigger. Value that w epass to trigger s anme of state and this needs to be a string.

app.componnet.html-

<div

[@clickedState]="'default'"

></div>

App.component.ts-

@Component({

selector: 'app-root',

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

styleUrls: ['./app.component.css'],

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

})),

state('clicked', style({

backgroundColor: 'blue',

width: '300px',

height: '50px'

}))

])

]

})

Now we have attached a trigger with state to our element. styles of states are attached to element.

In next lectures we will see how to switch between states.

63)Switching Trigger states dynamically

In Last lecture we saw how to attach a trigger to element and learned something super important- trigger in the end does one thing, it assigns the style which are set for given state to element ,to which trigger is attached to.

In last lecture we hardcoded the state to trigger .lets make it more dynamic. This how we do this-

<div

(click)="clickInfo = 'clicked'"

[@clickedState]="clickInfo"

></div>

In ts file-

clickInfo = 'default';

now when we click on div we change value of string, which in turn changes the state of trigger so different style is applied. we have’nt seen any animation because we have not set any transition yet, we will do it soon.

Anoter way can be-

<div

(click)="onClickSimple()"

[@clickedState]="clickInfo"

></div>

App.component.ts-

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

import {trigger , state, style } from '@angular/animations';

@Component({

selector: 'app-root',

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

styleUrls: ['./app.component.css'],

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

})),

state('clicked', style({

backgroundColor: 'blue',

width: '300px',

height: '50px'

}))

])

]

})

export class AppComponent {

clickInfo = 'default';

onClickSimple() {

this.clickInfo = 'clicked';

setTimeout(() => {

this.clickInfo = 'default';

}, 5000);

}

}

We can achieve the same result using ngClass and then assigning different classes . So right now this thing is nice but not huge advantagel. Lets change it, lets animate these trnasitions.

64)Adding transition between States

In last lecture we saw how to switch between different states, by changing states we changed the style. But This is not a huge advantage, as we can also do the same (change style)with ngClass. But this trigger system, this animation system makes much more sense as soon as you start animating things. In slide we had transiotions also, where we can animate things. Before I come to temporary styles and stuff like that let’s create a basic transition.

To do this we go back to our animations property, to trigger we passed 2 arguments- name of trigger and array of states. In second argument which is array, beside defining states we can also define transitons. We use transition method to define transitions. This method takes 2 arguments- first arguments defines when this transition should be played, so for which state change this transition should be applied. because you can set up multiple transitions here for different changes between different states. State change is described as such

transition('default => clicked')

this means this transition if when we move from default state to clicked state. So this is definition of when this transition should be fired, next argument is what should happen once this animation is fired. Here we got multiple options, basically here to animate it , we are going to use animate method. Here we pass 2 arguments- first one is timming. Timming means- duration,delay and timing function. first argument is like this-

transition('default => clicked', animate('200ms 500ms ease-in'))

it is duration, delay and timming function. we will come to timming functions, in next modules. Second argument is optional, it is styles that we want to assume at the end of animation. This is something which will become more important later when we have multiple animations chained togather. Default ofcourse is that at the end of animation it will take the ending sate. In our case it will be clicked state. So we won’t use this argument for now, again we will later of ofcourse use it and also learn how we can chain multiple animatons togather and things like this.

Note- ease-in means start slow and end fast.

So this is how we can set up transition, use animate function to play animation in this transition. But we have more tools than this simple setup.

65)Using Multiple Transitions

Here we added second transition-

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

})),

state('clicked', style({

backgroundColor: 'blue',

width: '300px',

height: '50px'

})),

transition('default => clicked', animate('200ms 500ms ease-in')),

transition('clicked => default', animate(300))

])

]

This for clicked to default state. Here we passes only one argument. It means transition takes 300ms, no delay and timming function will be linear ease-in(constant speed across th full animation )

So now we are using 2 transition. But what if we use more states and also might have transitions which are same for different state transiitons. We will see what it manes in next lecture.

66)Building more complex triggers with more states

Here we add a new state called mousedown, we want to use this state when user presses the mouse button but have’nt released it yet. We do this-

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

})),

state('clicked', style({

backgroundColor: 'blue',

width: '300px',

height: '50px'

})),

state('mousedown', style({

backgroundColor: 'red',

border: '1px solid black'

})),

transition('default => clicked', animate('200ms 500ms ease-in')),

transition('clicked => default', animate(300))

])

]

To attach this state with our div we do this-

App.component.html-

<div

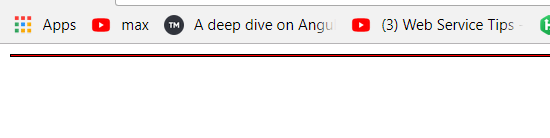
(click)="onClickSimple()"

(mousedown)=" clickInfo = 'mousedown'"

[@clickedState]="clickInfo"

></div>

When we click on div we see this-



Why this? Here is important note about state you have and styling it gets assigned. The key thing it does’nt work like this is that when we set up styles here in a state , it only assigns the style of the current state to the element it does’nt keep any of old state styles. so when we move from default to mousedown state, it does not keep the width and height of default state. styles of old states are dumped. In mousedown state we do not specify any width and height.

If you have some styles which should be applied to any of your elements independently of current animation state, you should set up these styles in normal css sheet, in a class,id then you should assign this class id to your element. This will make sure that you have these styles around even if you do not animate them. If you animate them ofcourse don’t use css to set them, set them in state styles.

Now we add the width and height to mousedown state and things work as we expect.

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

})),

state('clicked', style({

backgroundColor: 'blue',

width: '300px',

height: '50px'

})),

state('mousedown', style({

backgroundColor: 'red',

border: '1px solid black',

width: '100px',

height: '100px'

})),

transition('default => clicked', animate('200ms 500ms ease-in')),

transition('clicked => default', animate(300))

])

]

67)Configuring elegant transition when using many states

In last lecture we learned that styles of old state are not applied in current state. In this lecture we added some new transitions-

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

})),

state('clicked', style({

backgroundColor: 'blue',

width: '300px',

height: '50px'

})),

state('mousedown', style({

backgroundColor: 'red',

border: '1px solid black',

width: '100px',

height: '100px'

})),

transition('default => clicked', animate('200ms 500ms ease-in')),

transition('clicked => default', animate(300)),

transition('mousedown => clicked', animate(300)),

transition('clicked <=> mousedown', animate(300))

])

]

Here we are doing little overhead, we basically have the same animation for same transition just in both directions. In last 2 transitions we are moving between mousedown and clicked state. So instead of –

transition('mousedown => clicked', animate(300)),

transition('clicked => mousedown', animate(300))

we do this-

transition('mousedown <=> clicked', animate(300)),

68)Understand the re-usability of triggers

Here we used same trigger on 2 different element. We have attached 2 different trigger sources to these elemnts, so states can be trigger independently on these element.

Trigger can be reused on multiple element, if trigger is triggered n one lement then it does not mean that it will also be triggered on second element.

69)Using Multiple Triggers

Here we will create new trigger and will dive into more complex animations.

Here we have input, then we have some nubers displayed, depending on number enterdin input we want to select the number that we have entered. Code-

Html-

<input type="number" (input)="numberEntered = $event.target.value">

<br><br>

<span [@numberEnteredState]="numberEntered == 0 ? 'selected' : 'unselected'">0</span>

<span [@numberEnteredState]="numberEntered == 1 ? 'selected' : 'unselected'">1</span>

<span [@numberEnteredState]="numberEntered == 2 ? 'selected' : 'unselected'">2</span>

<span [@numberEnteredState]="numberEntered == 3 ? 'selected' : 'unselected'">3</span>

<span [@numberEnteredState]="numberEntered == 4 ? 'selected' : 'unselected'">4</span>

<span [@numberEnteredState]="numberEntered == 5 ? 'selected' : 'unselected'">5</span>

<span [@numberEnteredState]="numberEntered == 6 ? 'selected' : 'unselected'">6</span>

<span [@numberEnteredState]="numberEntered == 7 ? 'selected' : 'unselected'">7</span>

<span [@numberEnteredState]="numberEntered == 8 ? 'selected' : 'unselected'">8</span>

<span [@numberEnteredState]="numberEntered == 9 ? 'selected' : 'unselected'">9</span>

<span [@numberEnteredState]="numberEntered == 0 ? 'selected' : 'unselected'">0</span>

Ts-

@Component({

selector: 'app-root',

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

styleUrls: ['./app.component.css'],

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

})),

state('clicked', style({

backgroundColor: 'blue',

width: '300px',

height: '50px'

})),

state('mousedown', style({

backgroundColor: 'red',

border: '1px solid black',

width: '100px',

height: '100px'

})),

transition('default => clicked', animate('200ms 500ms ease-in')),

transition('clicked => default', animate(300)),

transition('mousedown <=> clicked', animate(300)),

// transition('clicked => mousedown', animate(300))

]),

trigger ('numberEnteredState', [

state('unselected', style({

border: '1px solid black',

padding: '5px'

})),

state('selected', style({

border: '2px solid blue',

padding: '4px',

backgroundColor: 'lightblue'

}))

])

]

})

export class AppComponent {

clickInfo = 'default';

paragraphClick = 'default';

numberEntered;

now we want to animate it.so we add this-

transition('unselected => selected', animate (300))

but now when we enter a number elements jump around a bit. We need to fix it. We will improve thus animation in next lectures.

70)Creating multistep animation

In last lecture we saw that elements jump around a liitle when we added animations. Here borer width is not animated, so we have little bit of jumping around since padding decrease is animated. To be precise , both(border and padding) are animated but does’nt work togather well here. It would be nice to have a more complex animation there, maybe bump up item, make it bigger during animation to make really clear what was selected.

Regarding border , that clearly is not optimal. It would be nice if we instantly change it, not at end of animation, which leads to this ugly jumping around. We can fix it by writing more complex transition.

Right now we pass animate as second argument to transition function. we can also pass array as second argument to transition function. in array we can define multiple transition steps. Transition steps can be, in this case animations or changes of the styles. so we can set style and now this will not be animated, this simple means that as a first step in this transition, instantly set the style .so you can now set border instantly. However we set the color of border to be black, because we want to animate the transition of color, which is animatable. We also immediately set the padding to 4px. So we wnt animate padding though we can do this but I dnt want to animate it because I dnt want to have any jumping around.

App.component.ts-

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

})),

state('clicked', style({

backgroundColor: 'blue',

width: '300px',

height: '50px'

})),

state('mousedown', style({

backgroundColor: 'red',

border: '1px solid black',

width: '100px',

height: '100px'

})),

transition('default => clicked', animate('200ms 500ms ease-in')),

transition('clicked => default', animate(300)),

transition('mousedown <=> clicked', animate(300)),

// transition('clicked => mousedown', animate(300))

]),

trigger ('numberEnteredState', [

state('unselected', style({

border: '1px solid black',

padding: '5px'

})),

state('selected', style({

border: '2px solid blue',

padding: '4px',

backgroundColor: 'lightblue'

})),

transition('unselected => selected', [

style({

border: '2px solid black',

padding: '4px'

}),

animate (300)])

])

]

Now if we save this code, we will see that now there is no jumping around but colors are still animating. There is no jumping around because we dnt try to animate change of border width. This shows us different way of suing transition instead of just having animation as second argument to transition, we have array of steps in this transition and first step is instantly change the following styles(styles that we pass to style function). these styles are temporary that is why black color of border is not kept around. We get the blue color of border in end, which is our end state. Padding also would have been temporary but as we have same padding in the end, we just set it up earlier. It is this style in first step though which is taken into account in next step, in animation. It now wnt start at our beginning state,Unselected. It now starts at these styles –

style({

border: '2px solid black',

padding: '4px'

})

So it is therefore great tool for changing certain things and making sure that we start at state we want to start an may be do some steps in advance to avoid that nasty jumping.

We can also define style at the end of animation. We can also set styles at end of animation.like this-

animations: [

trigger('clickedState', [

state('default', style({

backgroundColor: 'orange',

width: '100px',

height: '100px'

})),

state('clicked', style({

backgroundColor: 'blue',

width: '300px',

height: '50px'

})),

state('mousedown', style({

backgroundColor: 'red',

border: '1px solid black',

width: '100px',

height: '100px'

})),

transition('default => clicked', animate('200ms 500ms ease-in')),

transition('clicked => default', animate(300)),

transition('mousedown <=> clicked', animate(300)),

// transition('clicked => mousedown', animate(300))

]),

trigger ('numberEnteredState', [

state('unselected', style({

border: '1px solid black',

padding: '5px'

})),

state('selected', style({

border: '2px solid blue',

padding: '4px',

backgroundColor: 'lightblue'

})),

transition('unselected => selected', [

style({

border: '2px solid black',

padding: '4px'

}),

animate (300),

style({

backgroundColor: 'red'

})

])

])

]

})

We set background color red at end of animation. Now we see strange jumping around and we do not see background color of red. The reason for this is same as before, third step is to change the styling but all the styles that we set in transition array are temporary styles, they are not kept around. So we have beginning style, then we animate, then we try to animate to red background but since it is only temporary we instantly jump back to our final state which is why we see jumping around and nothing else. However we will see red color we add another animating after we this style. We do that we see red background, but we also see jumping around, this is because we ignore new border and padding(that we have setup before animation) and we are not yet at our ending state.so that is why we go back to our totally unstyled state where we have no border and no padding, that is why border is removed during animation. After no border state we animate again to reach end sate which has border. Lets increase the time in animation to see what is happing.

So we remove the border and padding because that is our inbetween state, then we got red background color