Chapter 1 - Angular Animations

Objectives

Key objectives of this chapter

- What is Animation
- CSS Animation
- Angular Animation
- Triggers, Transitions, and States
- Invoking Transitions
- Animating Route Navigation
- Externalizing Animation Definitions

1.1 What is Animation?

- Angular Components include support for CSS animation
- Animation refers to the changing of CSS styles over time.
- Animation can be used to add visual interest when users:
  - Select list items
  - Click buttons
  - Scroll a list or view
- Animations can be used to highlight events such as:
  - Page transitions
  - Network downloads
- This chapter will review CSS animation and related concepts before covering how animation is implemented in Angular

1.2 Animation Configuration

- Angular 4 split animations into a separate module so there is some configuration required

    - Previously animations were in the core module which would already be configured
- Include '@angular/animations' in package.json dependencies

```
"@angular/animations": "~4.0.0",
```

- Import the 'BrowserAnimationsModule' from '@angular/platform-browser/animations' in your NgModule declaration

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

- If using SystemJS, add mapping in 'systemjs.config.js'

```
map: {
...
// angular animation bundles
'@angular/animations': 'npm:@angular/animations/...',
'@angular/animations/browser': 'npm:@angular/...',
'@angular/platform-browser/animations': 'npm:@angular/...',
```

- Import individual animations from '@angular/animations'

```
import {style, animate} from '@angular/animations';
```

Animation Configuration

The early documentation of Angular 4 does not always reflect the changes needed for configuration that were introduced with the animations changes introduced in Angular 4.

Full text of systemjs.config.js mappings:

```
map: {
...
// angular animation bundles
'@angular/animations': 'npm:@angular/animations/bundles/animations.umd.js',
'@angular/animations/browser': 'npm:@angular/animations/bundles/animations-
browser.umd.js',
'@angular/platform-browser/animations': 'npm:@angular/platform-
browser/bundles/platform-browser-animations.umd.js',
```

The BrowserAnimationsModule should also be added to the 'imports' array of the NgModule declaration.

```
  imports: [
    ...,
    BrowserAnimationsModule,
    AppRoutingModule
  ],
```

1.3 Animation Techniques

- Common Animation techniques include:
  - 2 dimensional movement
  - fade-in/out
  - grow/shrink
  - color morphing
- All of these are accomplished through CSS property changes

1.4 Animation Concepts

Time	Animations happen over time
CSS Properties	Animations modify CSS properties
Opacity	Animations can change the partial transparency of visual elements
Transforms	Refer to specific types of property changes
Key Frames	Key frames define the start and end as well as in between states of an animation
States	States refer to a particular configuration of properties and their exact values at a given point in an animation
Transitions	Transitions define changes in state

1.5 CSS Property Animation

- Moving a box across the page is a simple animation
- The box is defined as a with the following styles:

```
.animation{     width: 100px;
                height: 100px;
                background-color: green;
                display: block;
                position: absolute;
                left: 0px;
```

```
                    animation-name: boxmove;
                    animation-duration: 4s;
    }
```

- The box's starting and ending position are defined in the CSS stylesheet via @keyframes:

```
    @keyframes boxmove{  from { left: 0px; }
                         to { left: 200px; }
    }
```

- Note how the animation name "boxmove" is used to associate the keyframes with the div
- The animation starts with the div in the ***from*** position and ends up 4 seconds later with it in the ***to*** position.

1.6 Animation Property Settings

- The previous example used a minimal number of animation properties.

Property	Description/ Values
animation-name	links keyframes to element
animation-duration	how long it takes animation to run [i.e. 4s]
animation-delay	delays the start of animation [i.e. 3s]
animation-iteration-count	how many times it runs [# | infinite]
animation-direction	order keyframes run in [reverse | alternate]
animation-timing-function	sets acceleration [linear | ease-in | ease-out | …]
animation-play-state	start and stop animation [running | paused]
animation-fill-mode	sets state of animated properties before and after animation runs [backwards | forwards | both]

- Animation properties are applied in the CSS style section of the element or elements being animated.

1.7 CSS Transforms

- CSS Transforms:
  - Modify visual aspects of a given element
  - Can be animated
- CSS Transforms include: scale, rotate, and translate (change position), as well as others.
- Transform rules are set in the CSS style sheet

```
transform: rotate(45deg);
```

- Multiple transform rules can be set at once:

```
transform: translateX(50px) scale(.8) rotate(45deg);
```

- Transform rules can be used in CSS animations:

```
@keyframes xform1{
  from {transform: translateX(0) rotate(0deg);}
  to { transform: translateX(50%) rotate(45deg);}
}
```

1.8 Starting and Stopping Animation

- Animation starts:
  - When a class defining animation properties is added to an element
  - When the page loads if the class defining the animation is already assigned to the element
- Animation ends:
  - When the class defining the animation is removed from the element
  - After the last animation iteration runs
- Animation can be paused by modifying the animation-play-state property

1.9 Animation Events

- Animation activity generates various DOM events
  - animationstart - generated when an animation starts

- animationend - generated when an animation ends
- animationiteration - generated at the end of each animation iteration

- Listeners can be attached to these events:

```
box.addEventListener("animationstart",
function(event){console.log(event.type);});
```

- animationend is triggered by the completion of the last iteration. It is not triggered when the animation is paused or when it is ended by removing the animation class from the element

1.10 Browser Support

- Browser support for the Web Animations (WA) standard is mixed:
  - Firefox and Chrome include WA support out-of-the-box
  - IE, Edge and Safari support WA through the web-animations-js library
    - https://github.com/web-animations/web-animations-js
- The web-animations-js library can be installed via npm directly or as a package.json dependency:

```
npm install web-animations-js
"web-animations-js": "^2.2.2" (add to package.json)
```

- After installation the library should be added to your app's index.html:

```
<script src="./node_modules/web-animations-js/web-
animations.min.js" ></script>
```

- For the latest information on browser support see:
  - http://caniuse.com/#search=web%20animations

1.11 Angular Animations

- Although Angular animations use the same underlying technology as CSS Web Animations the way they are set up and invoked is different.
- Angular animations are added to the component's "animations" array:

```
@Component({
 selector: 'my-app',
```

```
templateUrl: "app.component.html",
animations: [
  trigger('divstate', [
    state('oovleft', style({marginLeft: '-200px'})),
    state('inview', style({marginLeft: '0px'})),
    transition('oovleft => inview', animate('300ms')),
    transition('inview => oovleft', animate('300ms'))
    ])
  ]    })
```

- The animation is applied to an element using a property binding:

```
<div [@divstate]="state_value" > … </div>
```

- Where state_value is a component property:

```
state_value: string = "oovleft";
```

- Changing state_value from "oovleft" to "inview" causes the div to slide in from the left.

Angular Animations

Note: 'oov' is short for 'out of view'

1.12 Animation Imports

- The following imports are required for Angular Animation
- Place the following statement at the top of the file where animations are defined:

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

- Prior to Angular 4, you would also need to import 'AnimationEntryMetadata' when defining animation settings in their own file and exporting them
  - This is no longer required in Angular 4
  - All of the 'AnimationXXXMetadata' classes were deprecated in Angular 4

1.13 Named Animation States

- Animation endpoints are defined by with ***state()***

```
state('active', style({color: black }) )
state('inactive', style({color: gray }) )
```

- Each endpoint state has a name and an associated style object
- State names consist of arbitrary strings
- Although state names are arbitrary it helps to use names that suggest some meaning:

```
'visible', 'hidden', 'active', 'inactive',
'highlighted', 'normal', 'pressed'...
```

- Styles are set using JavaScript object notation:
- Any number of styles can be associated with a given state

```
{ color: 'black', backgroundColor: 'white',
  borderWidth: '2px' }
```

1.14 Transitions

- Angular animations are defined by ***transitions*** between ***states***:

```
transition('inactive => active', animate('100ms'))
```

- Endpoint states make up the transition statement parameter:

```
'inactive => active'
```

- The transition above moves from the 'inactive' to the 'active' state.
- The second parameter allows you to specify animation settings which is done using the animate() function:

```
animate('100ms ease-out')
```

- The animation specified here takes 100ms to go from start to end state.
- 'ease-out' specifies an acceleration profile such that the transition visually slows as it approaches the end state.

1.15 Special States: void, *

- **Void** state
  - The void state is used to describe elements when they are not visible
  - void refers to the element when it is not attached to a view
- Wildcard (*) state
  - The wildcard state matches any state including any named states and the void state
- Transition Examples:

Transition Statement	Description
void => *	A detached element entering any attached (visual) state
* => void	An element in any visual state becoming detached and no longer visible
* => *	matches transitions between any states
active => void	Transition from the 'active' named state to the detached state
* => normal	Transition from any state to the 'normal' named state

1.16 The animate() function

- The animate() function is used to define animation behavior
- It accepts two parameters: *timing* and *styles*
- The *timing* parameter:
  - Is a string
  - Can include values for duration, delay and easing
  - Example: '1s, 500ms ease-in'
- The *styles* parameter:
  - Can include calls to styles() or keyframes() or may be left empty

- styles defined here are applied to the final state of the animation

1.17 Triggers

- Think of the trigger as a wrapper around a set of animation states and transitions.

```
animations: [
 trigger('buttontrigger', [
   state('active', style({color: 'green'})),
   state('inactive', style({color: 'red'})),
   transition('active => inactive', animate('300ms')),
   transition('inactive => active', animate('300ms'))
   ])
 ]
```

- The trigger also defines the name that will be used to bind the animation to an element.

1.18 Assigning Animations to Elements using Trigger

- Animations will only have an effect when tied to one or more elements.
- Animations are associated with elements using a property binding:

```
<input type=button [@buttontrigger]='btn_state'>
```

- This ties the input element the animation trigger 'buttontrigger'
- If needed the same trigger (animation) can be inserted into multiple elements (i.e. separate buttons)

1.19 Invoking Transitions

- In the example on the previous page the button's state is changed by modifying the component's *btn_state* property.
- The state is held in a component property:

```
btn_state: string = 'active';
```

- A transition is triggered by modifying the value of the property:

```
toggle() {
  this.btn_state =
  (this.btn_state === 'active') ? 'inactive':'active';
}
```

- The above code toggles the state property that controls the animation.

1.20 Assigning Animation to Routes

- Animations can be set up so that they are triggered by route navigation
- To set up route animation:
  - The trigger name is set to 'routeAnimation'

```
animations: [trigger('routeAnimation', [
  state('*', style({ transform: 'translateX(0)'}) ),
  transition('void => *', [ style({ transform:
  'translateX(-100%)'}),animate('0.5s ease-in-out') ] ),
  transition('* => void', animate('0.5s ease-in-out',
  style({ transform: 'translateX(100%)'})))
])],
```

  - The following property and decorator are added to the component classes:

```
@HostBinding('@routeAnimation')
anyProperty = 'anything';
```

  - Although the @HostBinding decorator needs a property to be associated with the name and value of that property does not matter.
- When animating route navigation animation definitions (triggers) need to be supplied for each component being routed to. It is helpful in this case to put these in an external file that can be reused by the various components.

1.21 External Animation Definitions

- When Animations become complex it is useful to move them to a separate file and export them for use in one or more components.

- The animations.ts file:

```
import {style, animate, transition, state, trigger }
from '@angular/animations';
export class Animations {
  static page  = [
    trigger('page1', [
      state('1', style({marginLeft: '-200px'})),
      state('2', style({marginLeft: '0px'})),
      transition('1 => 2', animate('300ms')),
      transition('2 => 1', animate('300ms'))
      ]),
    trigger('page2', [
      state('1', style({marginLeft: '0px'})),
      state('2', style({marginLeft: '200px'})),
      transition('1 => 2', animate('300ms')),
      transition('2 => 1', animate('300ms'))
      ])
    ]   }
```

1.22 External Animation Definitions

- The animation definitions exported from the file on the previous page are imported and used in the following component:

```
import { Component } from '@angular/animations';
import { Animations } from './animations';

@Component({
  selector: 'my-app',
  templateUrl: './app/app.component.html',
  animations: Animations.page
})
export class AppComponent  {
  view: string = '2';
  page1message: string = 'initial message';
  toggle() {
     this.view = (this.view === '1') ? '2' : '1';
  }
}
```

1.23 Summary

In this chapter we covered:

- What is Animation
- CSS Animation
- Angular Animation
- Triggers, Transitions and States,
- Invoking Transitions
- Animating Route Navigation
- Externalizing Animation Definitions