Introduction to Angular animations

Animation provides the illusion of motion: HTML elements change styling over time. Well-designed animations can make your application more fun and straightforward to use, but they aren't just cosmetic. Animations can improve your application and user experience in a number of ways:

- Without animations, web page transitions can seem abrupt and jarring.
- Motion greatly enhances the user experience, so animations give users a chance to detect the application's response to their actions.
- Good animations intuitively call the user's attention to where it is needed.

Typically, animations involve multiple style *transformations* over time. An HTML element can move, change color, grow or shrink, fade, or slide off the page. These changes can occur simultaneously or sequentially. You can control the timing of each transformation.

Angular's animation system is built on CSS functionality, which means you can animate any property that the browser considers animatable. This includes positions, sizes, transforms, colors, borders, and more. The W3C maintains a list of animatable properties on its CSS Transitions page.

About this guide

This guide covers the basic Angular animation features to get you started on adding Angular animations to your project.

The features described in this guide — and the more advanced features described in the related Angular animations guides — are demonstrated in an example application available as a .

Prerequisites

The guide assumes that you're familiar with building basic Angular apps, as described in the following sections:

- Tutorial
- Architecture Overview

Getting started

The main Angular modules for animations are `@angular/animations` and `@angular/platform-browser`. When you create a new project using the CLI, these dependencies are automatically added to your project.

To get started with adding Angular animations to your project, import the animation-specific modules along with standard Angular functionality.

Step 1: Enabling the animations module

Import `BrowserAnimationsModule`, which introduces the animation capabilities into your Angular root application module.

Note: When you use the CLI to create your app, the root application module `app.module.ts` is placed in the `src/app` folder.

Step 2: Importing animation functions into component files

If you plan to use specific animation functions in component files, import those functions from `@angular/animations`.

Note: See a summary of available animation functions at the end of this guide.

Step 3: Adding the animation metadata property

In the component file, add a metadata property called `animations:` within the `@Component()` decorator. You put the trigger that defines an animation within the `animations` metadata property.

Animating a transition

Let's animate a transition that changes a single HTML element from one state to another. For example, you can specify that a button displays either **Open** or **Closed** based on the user's last action. When the button is in the `open` state, it's visible and yellow. When it's the `closed` state, it's translucent and blue.

In HTML, these attributes are set using ordinary CSS styles such as color and opacity. In Angular, use the `style()` function to specify a set of CSS styles for use with animations. Collect a set of styles in an animation state, and give the state a name, such as `open` or `closed`.

Let's create a new `open-close` component to animate with simple transitions.

Run the following command in terminal to generate the component:

```
ng g component open-close
```

This will create the component at `src/app/open-close.component.ts`.

Animation state and styles

Use Angular's `state()` function to define different states to call at the end of each transition. This function takes two arguments: a unique name like `open` or `closed` and a `style()` function.

Use the `style()` function to define a set of styles to associate with a given state name. You must use *camelCase* for style attributes that contain dashes, such as `backgroundColor` or wrap them in quotes, such as `'background-color'`.

Let's see how Angular's `state()` function works with the `style()` function to set CSS style attributes. In this code snippet, multiple style attributes are set at the same time for the state. In the `open` state, the button has a height of 200 pixels, an opacity of 1, and a yellow background color.

In the following `closed` state, the button has a height of 100 pixels, an opacity of 0.8, and a background color of blue.

Transitions and timing

In Angular, you can set multiple styles without any animation. However, without further refinement, the button instantly transforms with no fade, no shrinkage, or other visible indicator that a change is occurring.

To make the change less abrupt, you need to define an animation *transition* to specify the changes that occur between one state and another over a period of time. The `transition()` function accepts two arguments: the first argument accepts an expression that defines the direction between two transition states, and the second argument accepts one or a series of `animate()` steps.

Use the `animate()` function to define the length, delay, and easing of a transition, and to designate the style function for defining styles while transitions are taking place. Use the `animate()` function to define the `keyframes()` function for multi-step animations. These definitions are placed in the second argument of the `animate()` function.

Animation metadata: duration, delay, and easing

The `animate()` function (second argument of the transition function) accepts the `timings` and `styles` input parameters.

The `timings` parameter takes either a number or a string defined in three parts.

> *`animate (duration)`* or *`animate ('duration delay easing')`*

The first part, `duration`, is required. The duration can be expressed in milliseconds as a number without quotes, or in seconds with quotes and a time specifier. For example, a duration of a tenth of a second can be expressed as follows:

- As a plain number, in milliseconds: `100`
- In a string, as milliseconds: `'100ms'`
- In a string, as seconds: `'0.1s'`

The second argument, `delay`, has the same syntax as `duration`. For example:

- Wait for 100ms and then run for 200ms: `'0.2s 100ms'`

The third argument, `easing`, controls how the animation accelerates and decelerates during its runtime. For example, `ease-in` causes the animation to begin slowly, and to pick up speed as it progresses.

- Wait for 100ms, run for 200ms. Use a deceleration curve to start out fast and slowly decelerate to a resting point: `'0.2s 100ms ease-out'`
- Run for 200ms, with no delay. Use a standard curve to start slow, accelerate in the middle, and then decelerate slowly at the end: `'0.2s ease-in-out'`
- Start immediately, run for 200ms. Use an acceleration curve to start slow and end at full velocity: `'0.2s ease-in'`

Note: See the Material Design website's topic on Natural easing curves for general information on easing curves.

This example provides a state transition from `open` to `closed` with a 1-second transition between states.

In the preceding code snippet, the `=>` operator indicates unidirectional transitions, and `<=>` is bidirectional. Within the transition, `animate()` specifies how long the transition takes. In this case, the state change from `open` to `closed` takes 1 second, expressed here as `1s`.

This example adds a state transition from the `closed` state to the `open` state with a 0.5-second transition animation arc.

Note: Some additional notes on using styles within `state` and `transition` functions.

- Use `state()` to define styles that are applied at the end of each transition, they persist after the animation completes.

- Use `transition()` to define intermediate styles, which create the illusion of motion during the animation.
- When animations are disabled, `transition()` styles can be skipped, but `state()` styles can't.
- Include multiple state pairs within the same `transition()` argument: `transition( 'on => off, off => void' )`.

Triggering the animation

An animation requires a *trigger*, so that it knows when to start. The `trigger()` function collects the states and transitions, and gives the animation a name, so that you can attach it to the triggering element in the HTML template.

The `trigger()` function describes the property name to watch for changes. When a change occurs, the trigger initiates the actions included in its definition. These actions can be transitions or other functions, as we'll see later on.

In this example, we'll name the trigger `openClose`, and attach it to the `button` element. The trigger describes the open and closed states, and the timings for the two transitions.

Note: Within each `trigger()` function call, an element can only be in one state at any given time. However, it's possible for multiple triggers to be active at once.

Defining animations and attaching them to the HTML template

Animations are defined in the metadata of the component that controls the HTML element to be animated. Put the code that defines your animations under the `animations:` property within the `@Component()` decorator.

When you've defined an animation trigger for a component, attach it to an element in that component's template by wrapping the trigger name in brackets and preceding it with an `@` symbol. Then, you can bind the trigger to a template expression using standard Angular property binding syntax as shown below, where `triggerName` is the name of the trigger, and `expression` evaluates to a defined animation state.

```
<div [@triggerName]="expression">...</div>;
```

The animation is executed or triggered when the expression value changes to a new state.

The following code snippet binds the trigger to the value of the `isOpen` property.

In this example, when the `isOpen` expression evaluates to a defined state of `open` or `closed`, it notifies the trigger `openClose` of a state change. Then it's up to the `openClose` code to handle the state change and kick off a state change animation.

For elements entering or leaving a page (inserted or removed from the DOM), you can make the animations conditional. For example, use `*ngIf` with the animation trigger in the HTML template.

Note: In the component file, set the trigger that defines the animations as the value of the `animations:` property in the `@Component()` decorator.

In the HTML template file, use the trigger name to attach the defined animations to the HTML element to be animated.

Code review

Here are the code files discussed in the transition example.

Summary

You learned to add animation to a transition between two states, using `style()` and `state()` along with `animate()` for the timing.

Learn about more advanced features in Angular animations under the Animation section, beginning with advanced techniques in transition and triggers.

{@a animation-api-summary}

Animations API summary

The functional API provided by the `@angular/animations` module provides a domain-specific language (DSL) for creating and controlling animations in Angular applications. See the API reference for a complete listing and syntax details of the core functions and related data structures.

Function name	What it does
`trigger()`	Kicks off the animation and serves as a container for all other animation function calls. HTML template binds to `triggerName`. Use the first argument to declare a unique trigger name. Uses array syntax.
`style()`	Defines one or more CSS styles to use in animations. Controls the visual appearance of HTML elements during animations. Uses object syntax.
`state()`	Creates a named set of CSS styles that should be applied on successful transition to a given state. The state can then be referenced by name within other animation functions.
`animate()`	Specifies the timing information for a transition. Optional values for `delay` and `easing`. Can contain `style()` calls within.
`transition()`	Defines the animation sequence between two named states. Uses array syntax.
`keyframes()`	Allows a sequential change between styles within a specified time interval. Use within `animate()`. Can include multiple `style()` calls within each `keyframe()`. Uses array syntax.
`group()`	Specifies a group of animation steps (*inner animations*) to be run in parallel. Animation continues only after all inner animation steps have completed. Used within `sequence()` or `transition()`.
`query()`	Finds one or more inner HTML elements within the current element.
`sequence()`	Specifies a list of animation steps that are run sequentially, one by one.
`stagger()`	Staggers the starting time for animations for multiple elements.
`animation()`	Produces a reusable animation that can be invoked from elsewhere. Used together with `useAnimation()`.
`useAnimation()`	Activates a reusable animation. Used with `animation()`.
`animateChild()`	Allows animations on child components to be run within the same timeframe as the parent.

More on Angular animations

You might also be interested in the following:

- Transition and triggers
- Complex animation sequences
- Reusable animations
- Route transition animations

Check out this presentation, shown at the AngularConnect conference in November 2017, and the accompanying source code.