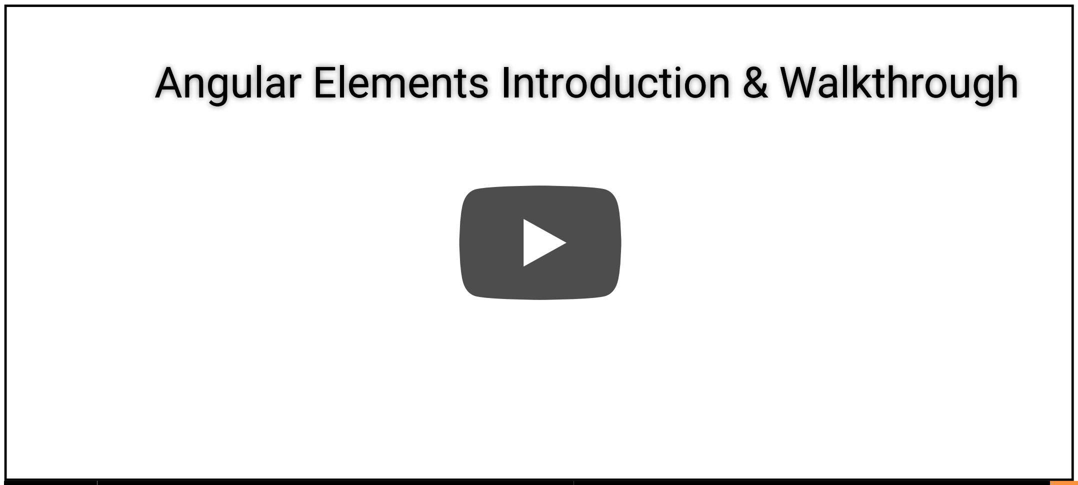
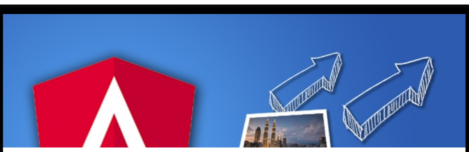
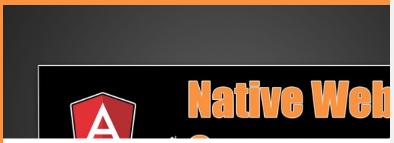

Angular Elements Introduction & Walkthrough

lar


Angular Image Upload Made Easy

Angular 6 - What's New & Upgrade Guide



Angular Elements Introduction

Content Information

Created by Maximilian Schwarzmüller

Last updated on 3. May 2018

SHOW RESOURCES

Newsletter email | REGISTER

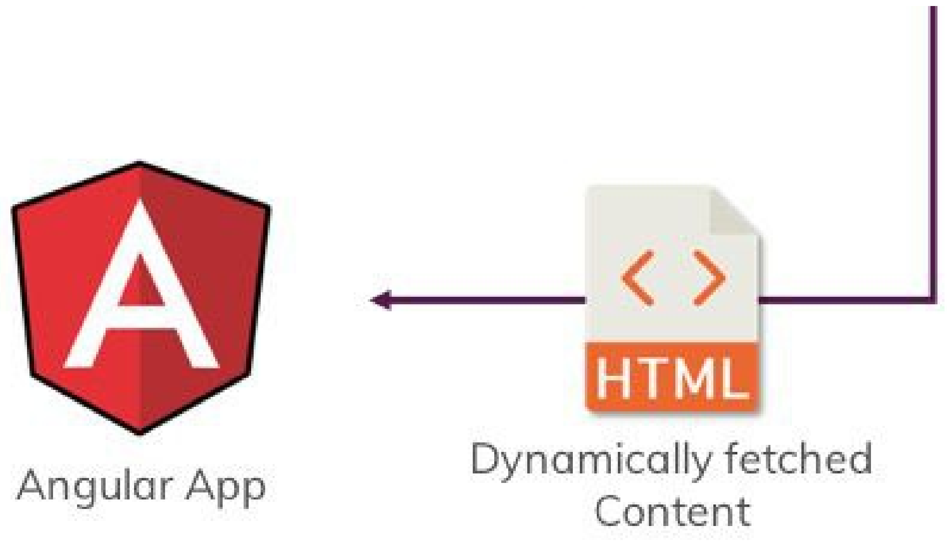
Angular Elements is a package which is part of the Angular framework: `@angular/elements` . It was introduced in Angular 6 - you can learn more about the new features introduced by Angular 6 in this article.

It offers functionality that allows you to convert a normal Angular component to a native web component.

As such, you could theoretically use it in any web page - no matter if that page uses Angular or not.

As of now, this is not really the best way to use them though. You can mainly use Angular Elements in Angular applications, not in other web pages. This will probably change in the future though. Theoretically, you can already build Angular Elements that are sharable but it requires quite a bit of manual work and wiring-up + the created bundles aren't very small. This will improve with Angular 7+.

Nonetheless, it's already very useful, especially when working with dynamically loaded HTML code.



Wouldn't it be nice if the people populating that data could also use your Angular components - and not just the default HTML tags? Maybe you want something like this in your server-side stored data:

```
<div>
  <p>The following snippet illustrates this concept:</p>
  <code-snippet></code-snippet>
</div>
```

`<code-snippet>` is **not** a default HTML element. It's probably one of your Angular components.

If you try to load the content - let's say via `innerHTML` , it will

```
  content = null;

  ngOnInit() {
    this.http.get('...').subscribe(data => this.content = dat
  }
}
```

# Understanding Angular Template Compilation

It's important to understand how Angular templates are compiled to understand why the above snippet won't render your beautiful `<code-snippet>` component when loaded like this.

Your Angular templates are compiled before they are loaded into the DOM!

This happens, no matter if you're using ahead-of-time or just-in-time compilation - this only defines if you compile during the development process or in the browser. But it always happens before the page is loading your templates into the DOM.

During this compilation step, Angular detects your component loading it dynamically.

This is where native web components can help you.

# Angular Elements Creates Self-bootstrapping Angular Components

With Angular Elements, you create native web components that essentially contain an entire Angular component. And they also contain the code that is needed to bootstrap that mini Angular app you got.

Therefore, you don't need Angular to compile and understand this selector. The Angular app is contained in the element instead. It starts as soon as its loaded into the DOM.

# Creating an Element

In order to create a custom element from an Angular component, you need to ensure that you got Angular 6 installed in your project. You might also need to install `rxjs-`

```
"@webcomponents/custom-elements": "^1.0.8"
```

This adds a polyfill which is required for custom elements to work.

You also need to adjust `polyfills.ts` :

```
import '@webcomponents/custom-elements/custom-elements.min';

/** Required for custom elements for apps building to es5. */
import '@webcomponents/custom-elements/src/native-shim';
```

With these things out of the way, let's create our own element. The following code is based on the example from the video, you find the [Github repo here].

```
// This is the component we want to turn into a web component
import { AlertComponent } from './alert.component';
import { createCustomElement } from '@angular/elements';

// This is the component where we'll do the conversion and wh
@Component({...})
export class AppComponent {
  constructor(injector: Injector) {
    // Create the custom element
```

it has to be passed into the element via the `createCustomElement` config (second argument).

You can then use `<my-alert>` anywhere in your app. It doesn't have to be inside the `AppComponent` .

The element is created there because you typically want to create the element (and register it) early in your app lifecycle.

Important: The tag you assign to your component - in the example `<my-alert>` - doesn't have to match the selector you assigned in the underlying Angular component.

You could then use the element like this:

```
// This is the component we want to turn into a web componen
import { AlertComponent } from './alert.component';
import { createCustomElement } from '@angular/elements';

// This is the component where we'll do the conversion and wh
@Component({
  ...
  template: '<div [innerHTML]="content"></div>'
})
export class AppComponent {
  content = null;
```

```
    }
}
```

`domSanitizer` is used because `<my-alert>` actually executes a bunch of JS code - that's the nature of web components. They are full managed via JS in the end.

This code will then successfully load your HTML code into the template and run `<my-alert>` as a self-containing mini Angular app. Of course you could fetch this code from an API or any other source instead of hardcoding it into the `constructor`.