Advanced Topics in Internet Application Programming

Angular 2

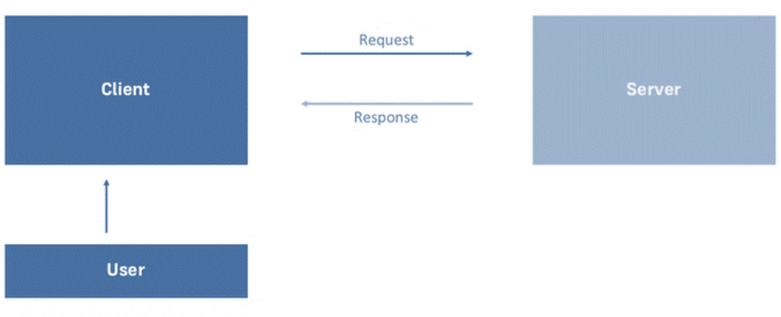
Dr. Shay Horovitz

Angular 2

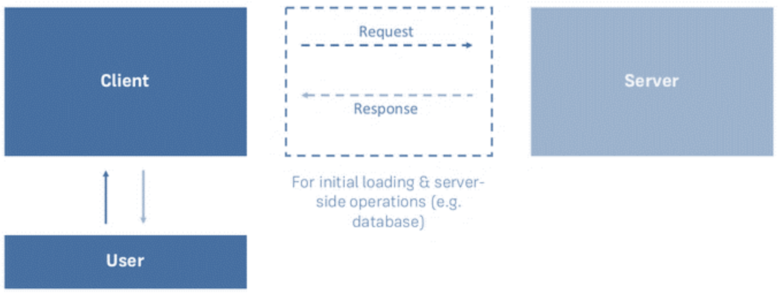
- Angular 2 is a framework for creating Single Page Applications (SPA)

Traditional vs SPA

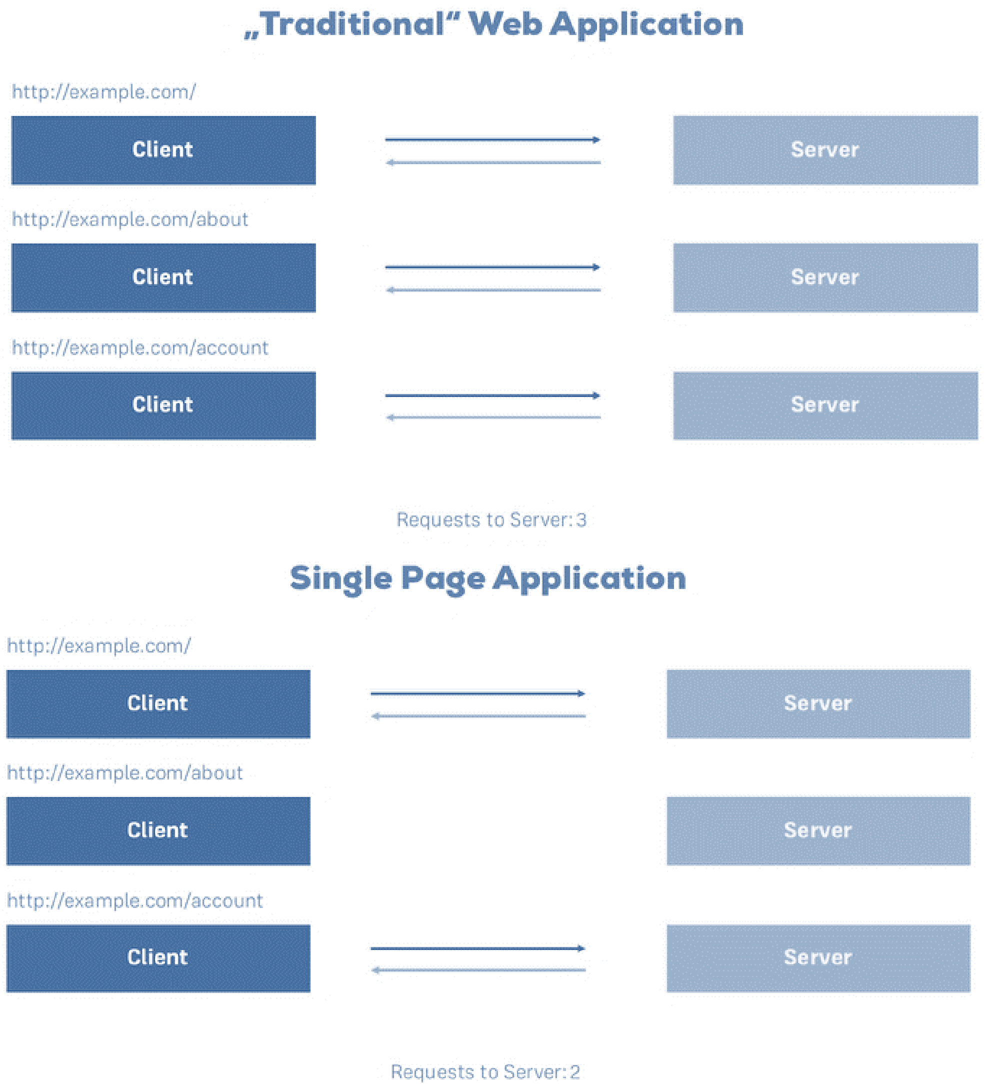
„Traditional“ Web Application



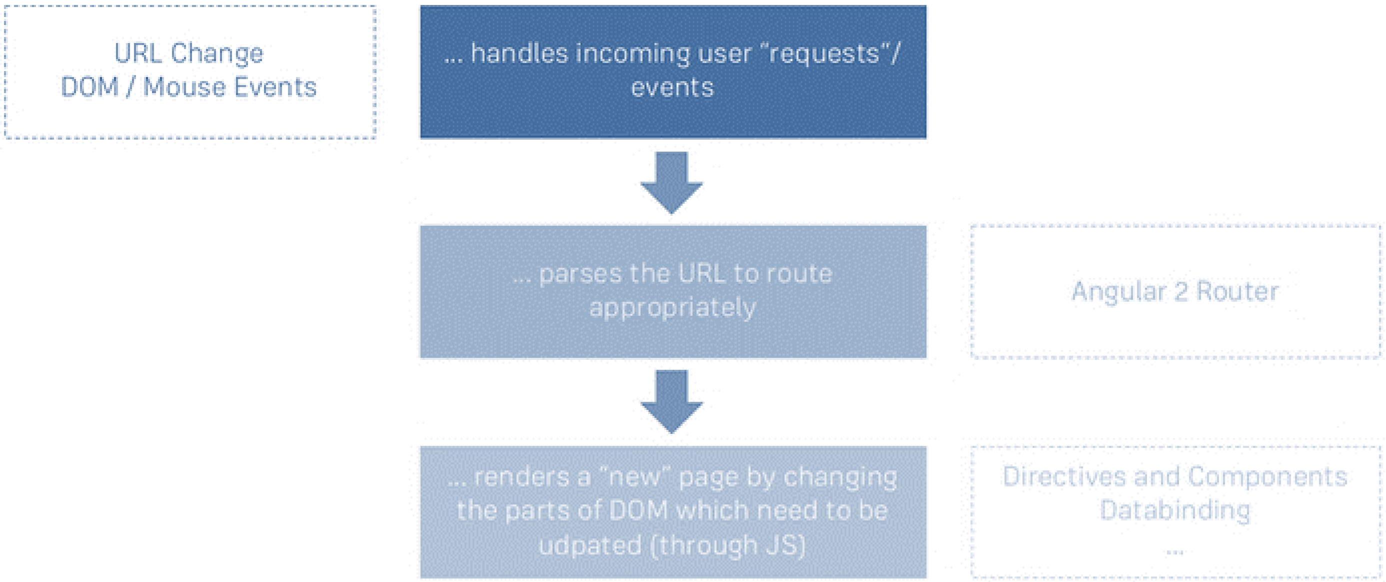
Single Page Application


Client
Request
Response
Server
User
For initial loading & server-side operations (e.g. database)

Traditional vs SPA Messages



Angular 2



URL Change
DOM / Mouse Events

... handles incoming user "requests"/ events

... parses the URL to route appropriately

Angular 2 Router

... renders a "new" page by changing the parts of DOM which need to be udpated (through JS)

Directives and Components
Databinding
...

Creating a Project

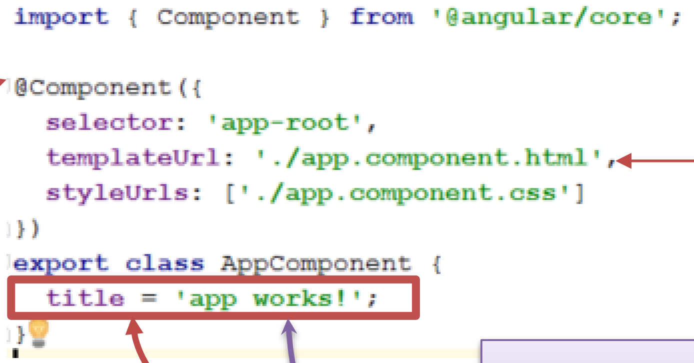
- Download & install Node.js (we will use its package manager NPM)
- Install Angular-Cli
  - **npm install –g angular-cli**
- Within your project folder, create a new angular project
  - **ng new first-app**
    - *Creates the files for our app to start with*
  - **cd first-app**
  - **ng serve**
    - *ng serve server will keep running and watch for changes in your code*
- In Google Chrome:
  - http://localhost:4200
- **That’s it! That is our first Angular 2 app working!!!**

Angular CLI

- important commands
- Create new project with new folder:
  - **ng new PROJECT_NAME**
- Create new project in existing folder:
  - **ng init**
- Build project:
  - **ng build**
- Serve project (will auto-reload upon changes to code):
  - **ng serve**
- Create a new component:
  - **ng generate component**

Files overview

- Open the project in WebStorm (or other framework)
- In \src\app\ notice:
  - app.component.ts :



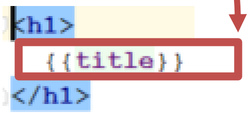
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app works!';
}
```

Create a component

Html to display

Try to modify the text and check the browser!

  - App.component.html :



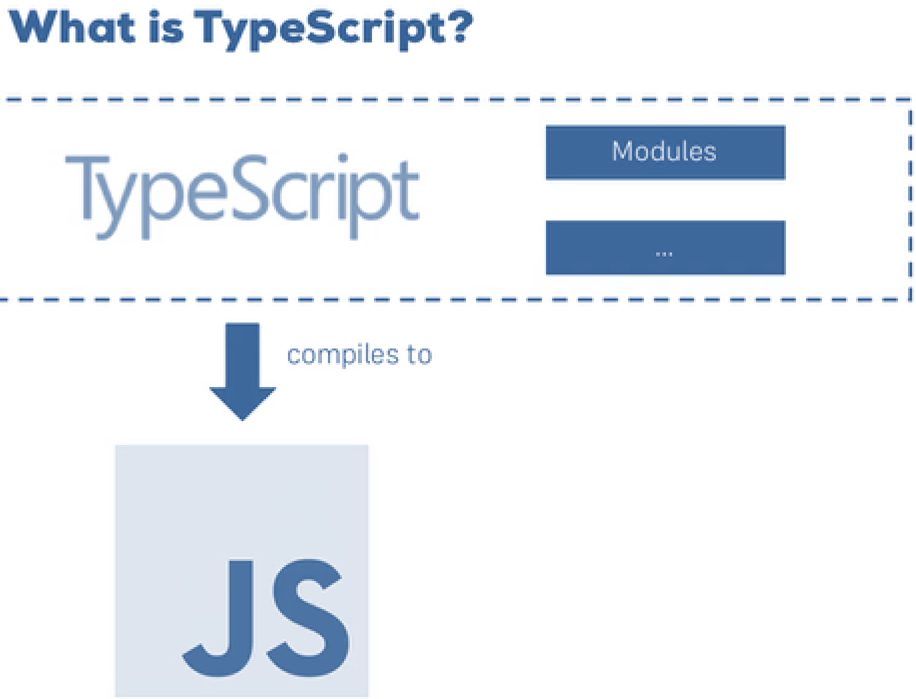
```
<h1>
  {{title}}
</h1>
```

Angular 2 Project Structure

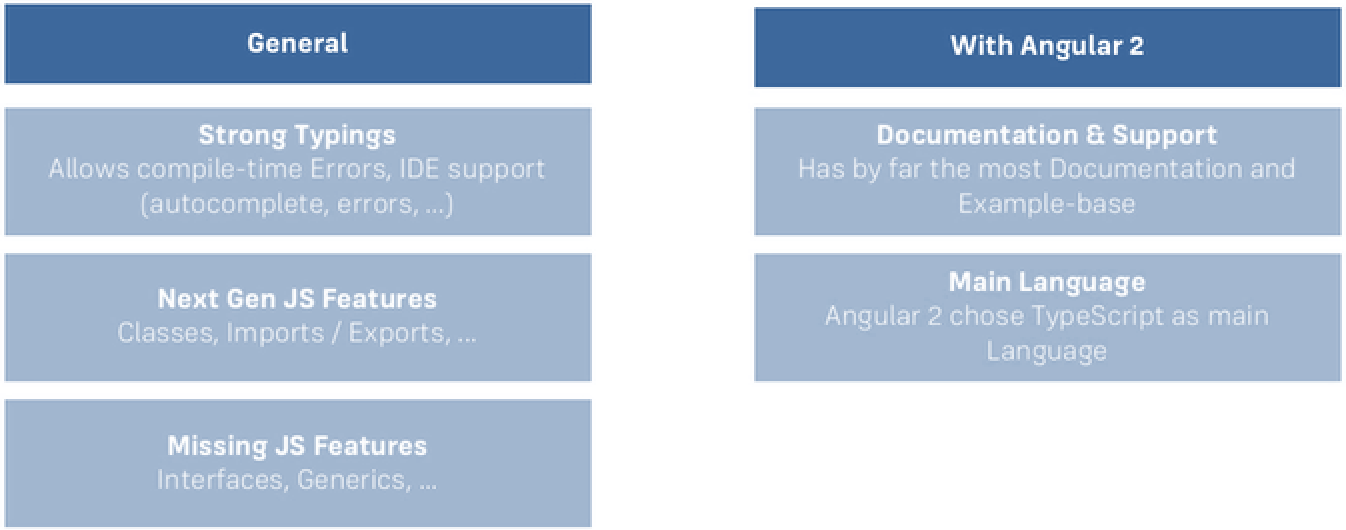
- App Root folder
  - **e2e -** *end to end testing*
  - **src** - *our code goes here*
    - **app -** *our angular 2 related files (typescript, htmls,…)*
      - **Share** – *shared components folder*
      - ***App.component.ts*** – *our component file*
      - ***App.component.css*** – *our component's css styling file*
      - ***App.component.html*** – *our component's html file*
      - **App.component.spec.ts** – *for unit testing, you can delete it*
      - **App.module.ts** – *a bundle telling angular 2 which parts our app has*
      - **Index.ts** – *keeping track over our files, kind of a mgmt. file*
    - **environments** – *environment configurations for the compilation of the project*
    - **Main.ts** – *the file that starts the angular 2 app*
    - **Styles.css** – *can define css settings for the whole app*
  - The other files outside the src folder – config files of the cli
    - Angular-cli.json – *where you store your code, what is the compile target*
    - Karma.conf.js – *for testing*
    - Protractor.conf.js – *for testing*

TypeScript

- A superset of JS that adds features to JS
- browsers don’t run TypeScript – we compile to js first



Why using **TypeScript** and not **JS**?



General

Strong Typings
Allows compile-time Errors, IDE support (autocomplete, errors, ...)

Next Gen JS Features
Classes, Imports / Exports, ...

Missing JS Features
Interfaces, Generics, ...

With Angular 2

Documentation & Support
Has by far the most Documentation and Example-base

Main Language
Angular 2 chose TypeScript as main Language

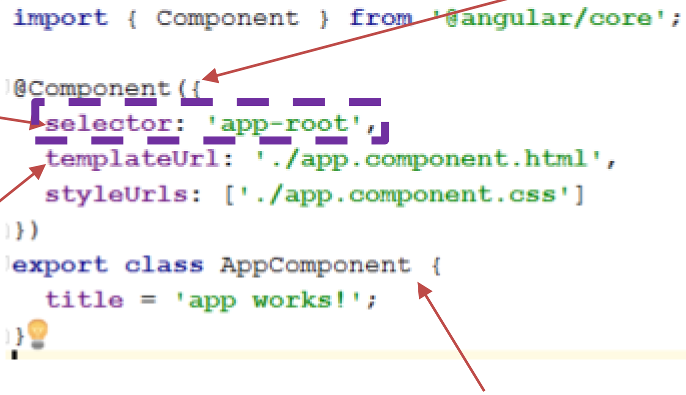
Install TypeScript

```
npm install typescript -g
```

Back to Components

Import code from another module

app.component.ts



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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app works!';
}
```

A **Decorator –** a function that is attached to a code (in this case to a class)

Selector – (like css selector) where to attach that component In the html

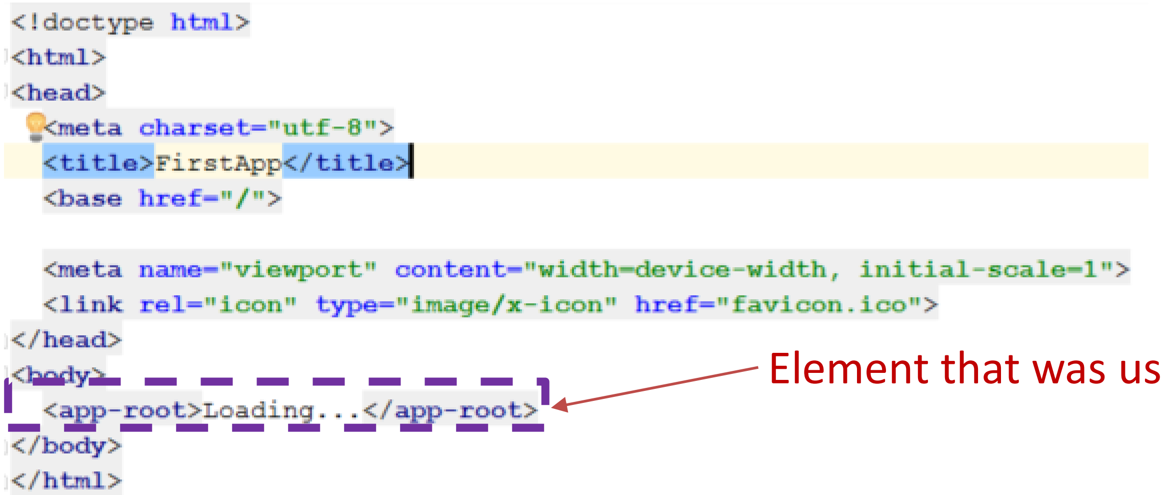
The @Component decorator takes a JS object as an argument and uses it to add some metadata to the class

Html Template for that component

TypeScript Class

Using @Component allows Angular 2 to work with that class

Index.html

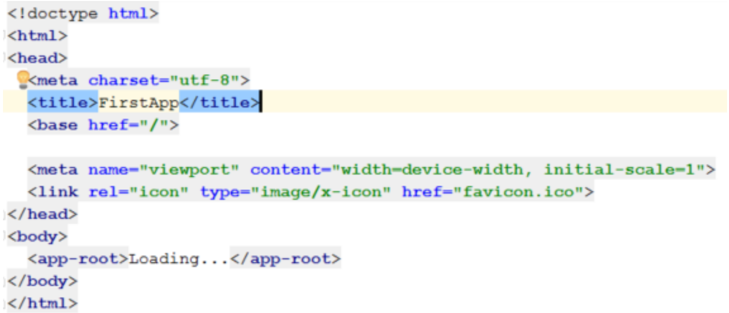


```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>FirstApp</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

Element that was used by the component’s selector

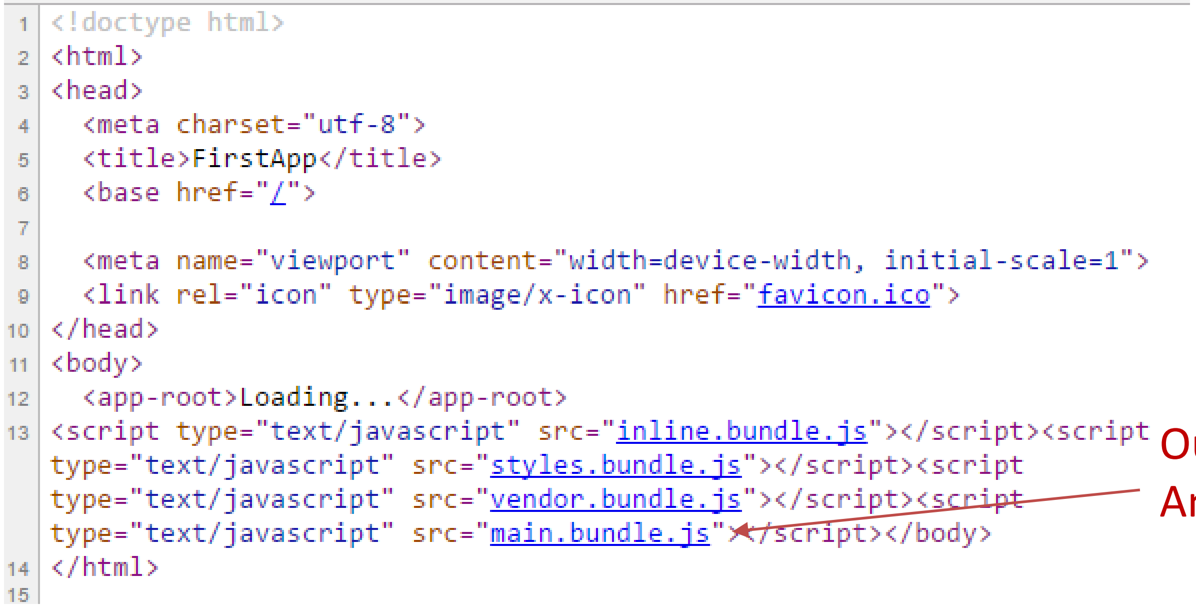
How does Angular 2 App gets started?

- Notice that there are no script tags in the index.html file…



```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>FirstApp</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

- Yet, the CLI of angular adds the scripts dynamically. Try View-Source in Chrome for our app:

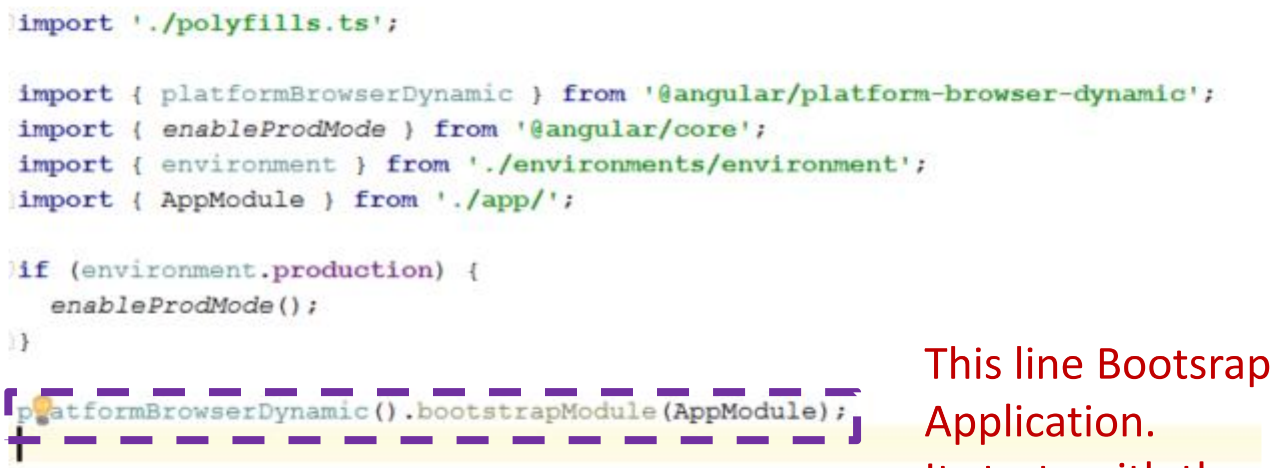


```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>FirstApp</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
<script type="text/javascript" src="inline.bundle.js"></script><script
type="text/javascript" src="styles.bundle.js"></script><script
type="text/javascript" src="vendor.bundle.js"></script><script
type="text/javascript" src="main.bundle.js"></script></body>
</html>
```

Our code including the Angular 2 code

Src\main.ts file

- This is the first file that gets loaded by the CLI

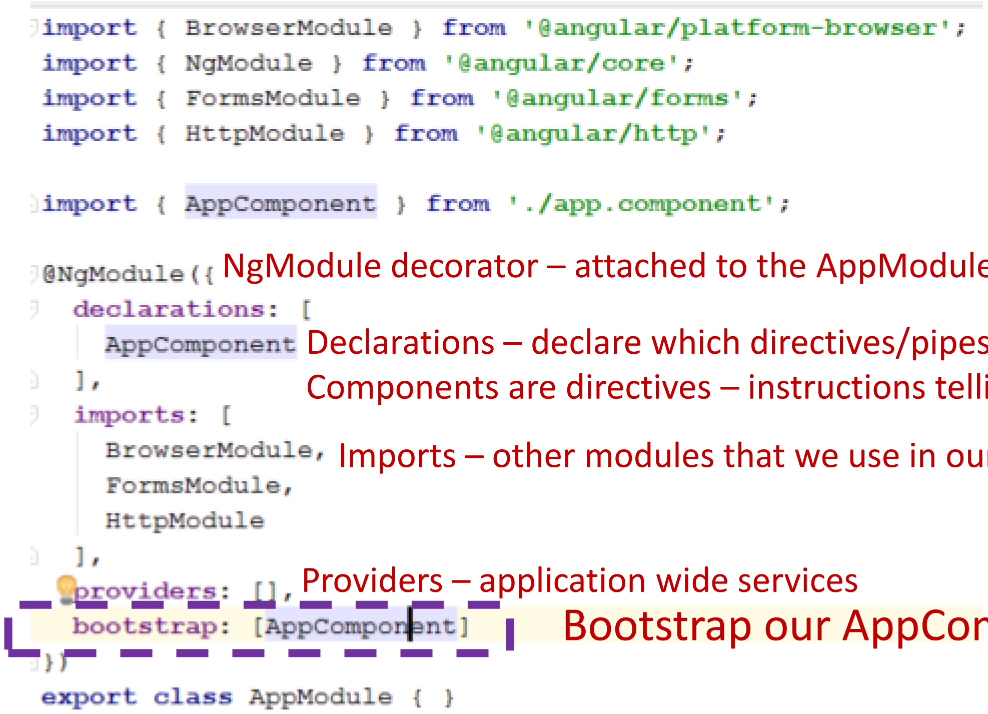


```
import './polyfills.ts';

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { enableProdMode } from '@angular/core';
import { environment } from './environments/environment';
import { AppModule } from './app/';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

This line Bootsraps (starts) our Application.
It starts with the module named **AppModule –** which is actually the **app.module.ts** file

Src\app\app.module.ts file

- This is the first file that gets loaded by the CLI



```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

NgModule decorator – attached to the AppModule class

Declarations – declare which directives/pipes we use in our app.
Components are directives – instructions telling angular 2 what to do.

Imports – other modules that we use in our code

Providers – application wide services

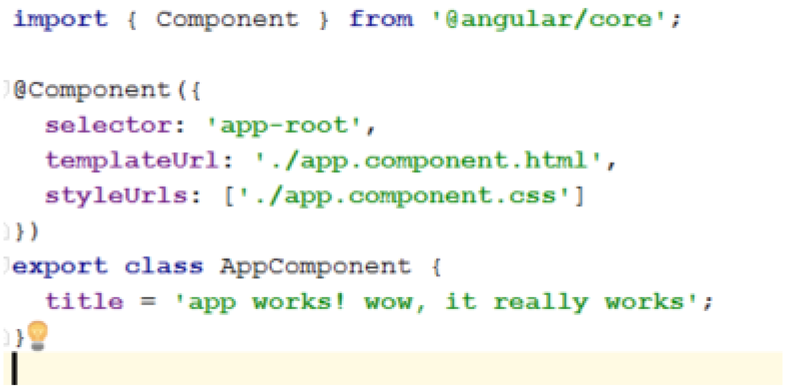
Bootstrap our AppComponent class – this is the root component of our application, which usually has a selector in the main html file

Summary of the call chain in Angular 2

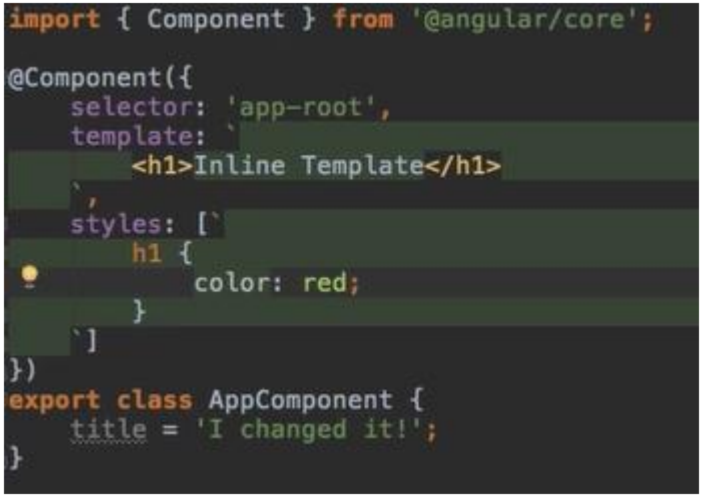
- **Main.TS** runs first
  - Bootstraps a module (AppModule)
- **AppModule** (in app.module.ts)
  - Bootstraps a component (AppComponent)
- **AppComponent** (in app.component.ts)
  - Renders its class (together with its html template) using a selector in the index.html file

Inline HTML instead of Template

in our app.component.ts



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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app works! wow, it really works';
}
```



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

@Component({
    selector: 'app-root',
    template: `
        <h1>Inline Template</h1>
    `,
    styles: [`
        h1 {
            color: red;
        }
    `]
})
export class AppComponent {
    title = 'I changed it!';
}
```

Before

Using external html template and external css

templateUrl:
stylesUrls:

After

Using internal html & css

with backtick (`) for multi-line code

template: ``
styles: ``

Creating our own Component

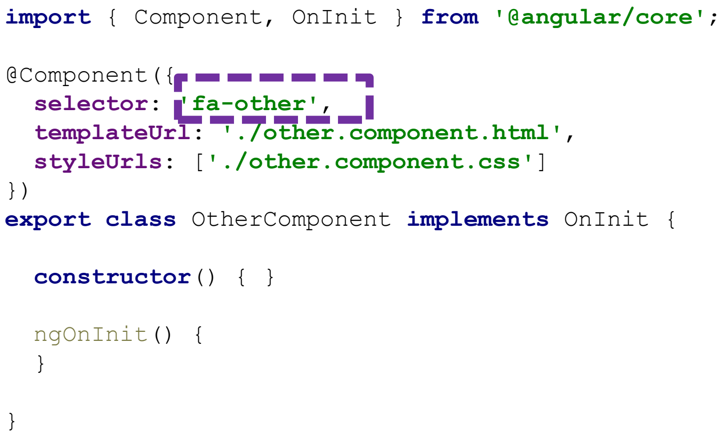
- In WebStorm terminal Alt-F12 (or regular command line) write:
  - **Ng generate component other**

Our new component name

- Under the \app folder, a new folder will be added named **other**
  - Notice that also references to the new component were inserted automatically to **\app\app.module.ts**
- You can delete the **css** and the **spec.ts** files and the reference to the css within other.component.ts

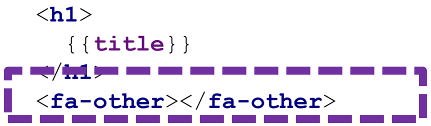
Connecting the component to our app

- In \app\other\other.component.html, modify the selector to be **fa-other**
  - **fa** stands here for **First App**



```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'fa-other',
  templateUrl: './other.component.html',
  styleUrls: ['./other.component.css']
})
export class OtherComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

- In \app\app.component.html, add an fa-other element:



```
<h1>
  {{title}}
</h1>
<fa-other></fa-other>
```

- **Look how the app changed in Chrome**

Adding yet another component…

- In the terminal (alt-F12):
  - Cd src/app/other/
  - Ng g c another --flat --is --it

generate

component

Don’t create a new folder

inline-styles

inline-template

- Notice that this time the CSS and Template files were not created
- Delete the another.component.**spec.ts** test file
- In **another.component.ts** Modify the selector to **fa-another** and add a call to it in \app\app.component.html

Files after update

for those of you that didn’t get it right…

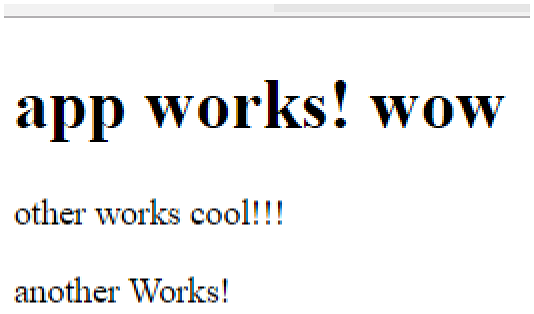
Another.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'fa-another',
  template: `
    <p>
      another Works!
    </p>
  `,
  styles: []
})
export class AnotherComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

App.component.html

```
<h1>
  {{title}}
</h1>
<fa-other></fa-other>
<fa-another></fa-another>
```

In Chrome:



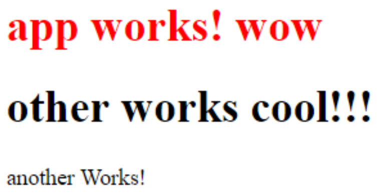
app works! wow

other works cool!!!

another Works!

View Encapsulation

- Angular 2 encapsulates CSS styles per each component in separate.
- Lets' test it!:
  - In \app\app.component.css insert this code:
    - h1{color:red}
  - In \app\other\other.component.html modify the <p> elements to <h1>
  - Now - Check up chrome:

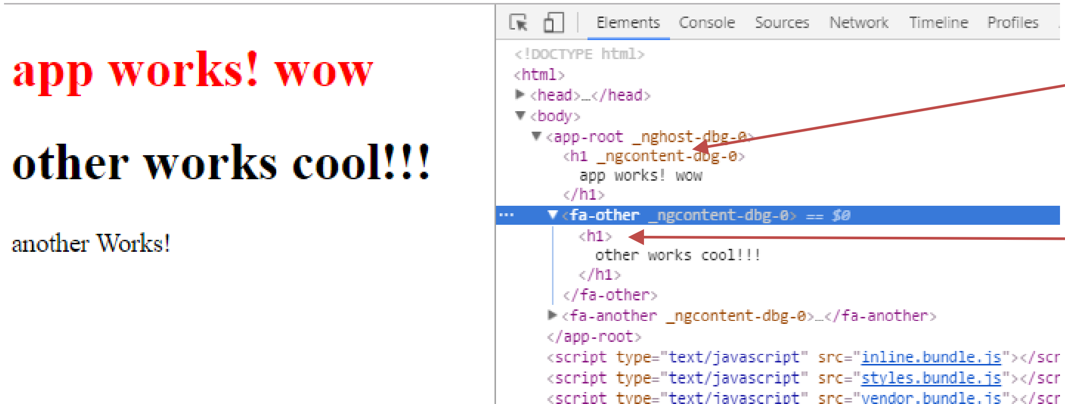


app works! wow

other works cool!!!

another Works!

  - Angular2 inserts a special attribute only to the first component such that only its h1 elements are affected by its css .
  - Go ahead and view source in Chrome:



app works! wow

other works cool!!!

another Works!

```
<!DOCTYPE html>
<html>
▶ <head>…</head>
▼ <body>
  ▼ <app-root _nghost-dbg-0>
      <h1 _ngcontent-dbg-0>
        app works! wow
      </h1>
    ▼ <fa-other _ngcontent-dbg-0> == $0
        <h1>
          other works cool!!!
        </h1>
      </fa-other>
    ▶ <fa-another _ngcontent-dbg-0>…</fa-another>
    </app-root>
    <script type="text/javascript" src="inline.bundle.js"></scr
    <script type="text/javascript" src="styles.bundle.js"></scr
    <script type="text/javascript" src="vendor.bundle.js"></scr
```

Attributed H1 for css

Regular H1 (no css)

Passing Parameters to Components with ng-content

- In app.component.html, duplicate <fa-another> twice:

```
<h1>
  {{title}}
</h1>
<fa-other></fa-other>
<fa-another></fa-another>
<fa-another></fa-another>
```

- What if we want each copy of fa-another present a different content? Modify app.component.html to this:

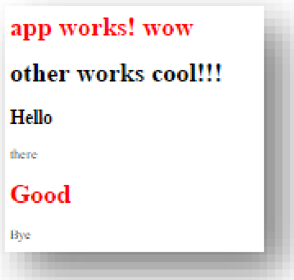
```
<h1>
  {{title}}
</h1>
<fa-other></fa-other>
<fa-another>
  <h2>Hello</h2>
  <p>there</p>
</fa-another>
<fa-another>
  <h1>Good</h1>
  <p>Bye</p>
</fa-another>
```

- But no change in Chrome… we have to let angular know that we pass content within the component.
- Go back to **another.component.ts** and

  add **ng-content**:

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'fa-another',
  template: `
    <ng-content></ng-content>

  `,
  styles: []
})
export class AnotherComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

In chrome:



app works! wow

other works cool!!!

Hello

there

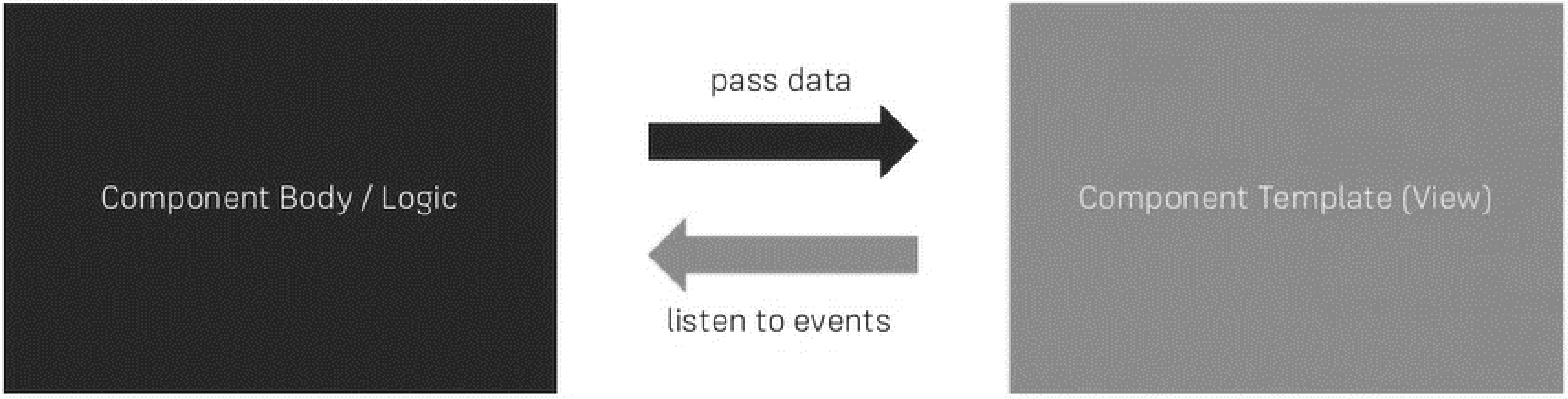
Good

Bye

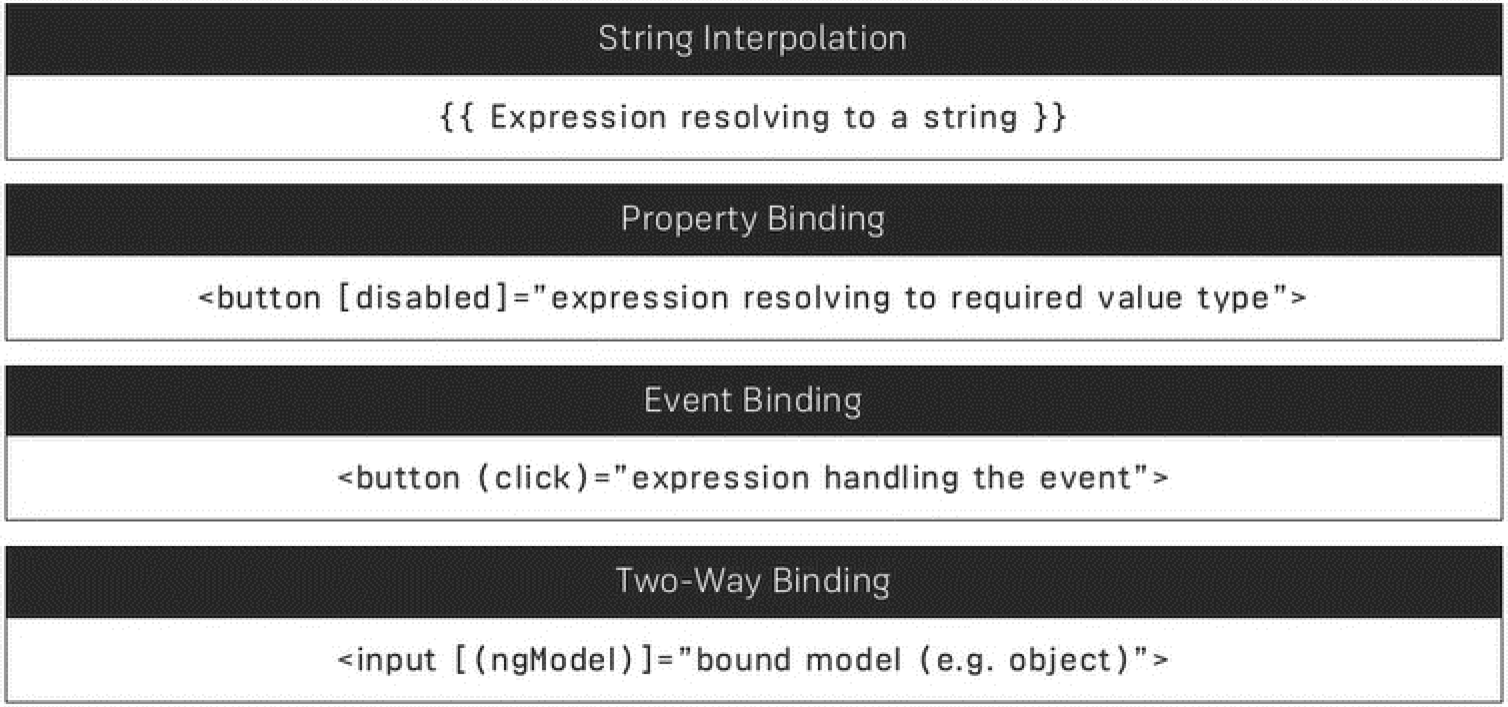
Notice the red H1 (Good)

Data Binding

allows communication between the components logic and the view



Methods of DataBinding



String Interpolation

```
{{ Expression resolving to a string }}
```

Property Binding

```
<button [disabled]="expression resolving to required value type">
```

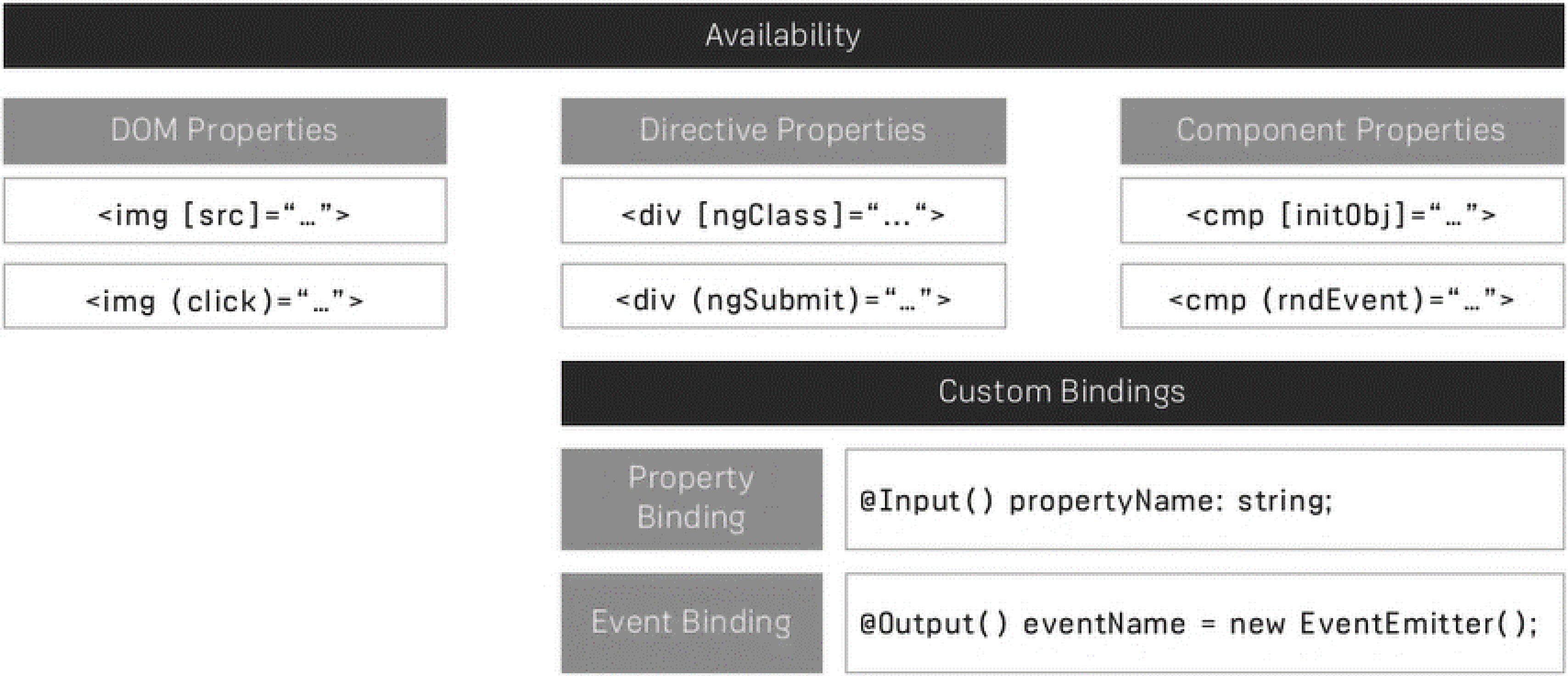
Event Binding

```
<button (click)="expression handling the event">
```

Two-Way Binding

```
<input [(ngModel)]="bound model (e.g. object)">
```

Property & Events Binding



Availability

DOM Properties	Directive Properties	Component Properties
<img [src]=“…”>	<div [ngClass]=“...“>	<cmp [initObj]=“…”>
<img (click)=“…”>	<div (ngSubmit)=“…”>	<cmp (rndEvent)=“…”>

Custom Bindings

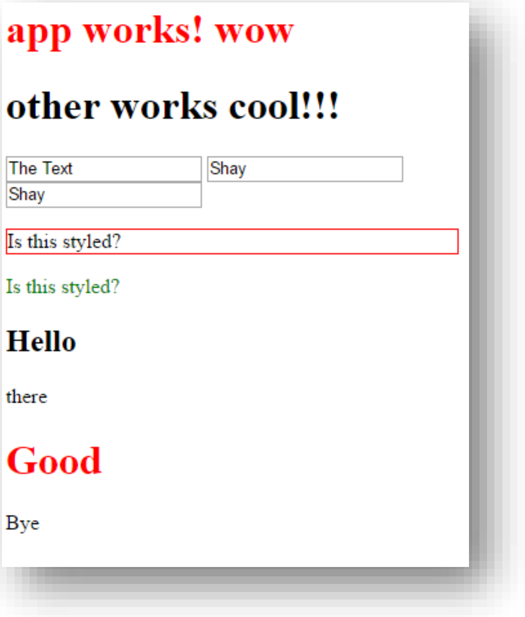
Property Binding	@Input() propertyName: string;
Event Binding	@Output() eventName = new EventEmitter();

Property Binding

Perform the following changes:

other.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'fa-other',
  templateUrl: './other.component.html',
  styles:[`.redBorder{border: 1px solid red;}`]
})
export class OtherComponent implements OnInit {
  ourStringInterpolation = "Shay ";

  OurOnTest()
  {
    return true;
  }
  constructor() { }
  ngOnInit() {
  }
}
```



app works! wow

other works cool!!!

The Text

Shay

Shay

Is this styled?

Is this styled?

Hello

there

Good

Bye

Style effect

other.component.html

```
<h1>
  other works cool!!!
</h1>
```

Regular input

String interpolation

DOM Property [value] binding

DOM Property [class] binding

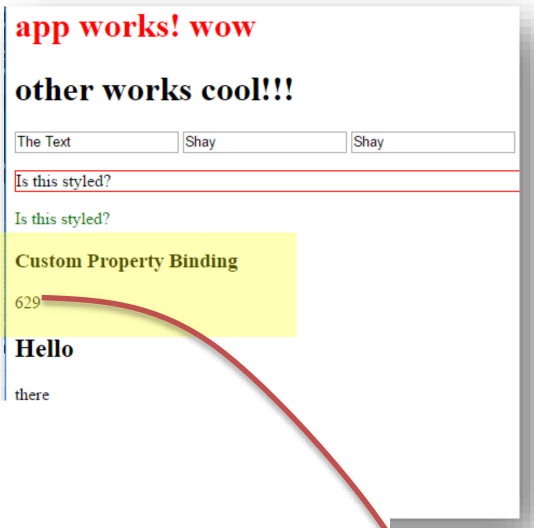
DOM Property [style] binding

```
<input type="text" value="The Text">

<input type="text" value="{{ourStringInterpolation}}">

<input type="text" [value]="ourStringInterpolation">

<p [ngClass]="{redBorder:OurOnTest()}">Is this styled?</p>

<p [ngStyle]="{color:'green'}">Is this styled?</p>
```

OurOnTest is a method in our component that returns **true**

Custom Property Binding

- Let’s create a new component:
  - using the terminal (Alt-F12) go to \src\app\other
  - ng g c property-binding --flat -it -is
  - Get rid of the spec.ts test file
  - Within **property-binding.component.ts** we’d like the template to support a dynamic content (remember ng-content? This time we’ll try something else)
    - First get rid of the **Init** interface
    - Rename the selector to **fa-property-binding**
    - Add **result:number** property as a number, and use **@input** to make it bindable from outside
    - Use **{{result}}** in the template in order to present the result
  - Within **other.component.html** we’d like to present the property-binding value simply by calling **fa-property-binding**



app works! wow

other works cool!!!

The Text Shay Shay

Is this styled?

Is this styled?

Custom Property Binding

629

Hello

there

property-binding.component.ts

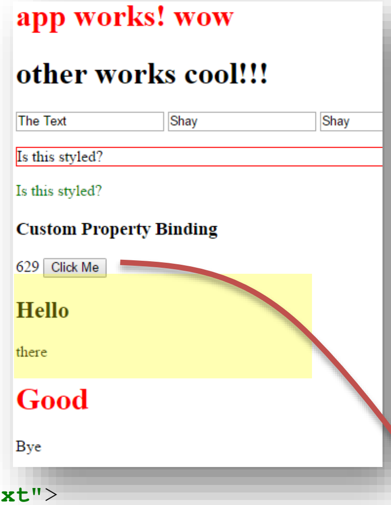
```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'fa-property-binding',
  template: `
    {{result}}
  `,
  styles: []
})
export class PropertyBindingComponent  {
  @Input() result:number = 0;
}
```

other.component.html

```
<h1>
  other works cool!!!
</h1>

<input type="text" value="The Text">
<input type="text" value="{{ourStringInterpolation}}">
<input type="text" [value]="ourStringInterpolation">
<p [ngClass]="{redBorder:OurOnTest()}">Is this styled?</p>
<p [ngStyle]="{color:'green'}">Is this styled?</p>
<h3>Custom Property Binding</h3>
<fa-property-binding [result]="629></fa-property-binding>
```

Event Binding – Click Event

- Let’s create a new component for event binding:
  - using the terminal (Alt-F12) go to \src\app\other
  - ng g c event-binding --flat -it -is
  - Get rid of the spec.ts test file
  - Within **event-binding.component.ts** we’d like the template to support a button event
    - First get rid of the **Init** interface
    - Rename the selector to **fa-event-binding**
    - Add a **button** element to the template with an event (click) calling to a method onClicked()
    - Define the method onClicked() in the event biding component.
  - Within **other.component.html** we’d like to present the event-binding component simply by calling **fa-event-binding**



app works! wow

other works cool!!!

The Text | Shay | Shay

Is this styled?

Is this styled?

Custom Property Binding

629 Click Me

Hello

there

Good

Bye

event-binding.component.ts

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

@Component({
  selector: 'fa-event-binding',
  template: `
    <button (click)="onClicked()">Click Me</button>
  `,
  styles: []
})
export class EventBindingComponent  {
  onClicked(){
    alert('wow!!! it works!');
  }

}
```

other.component.html

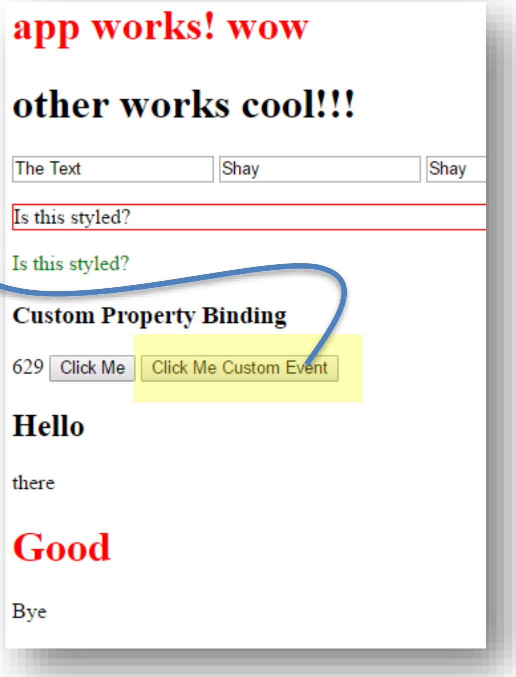
```
<h1>
  other works cool!!!
</h1>

<input type="text" value="The Text">
<input type="text" value="{{ourStringInterpolation}}">
<input type="text" [value]="ourStringInterpolation">
<p [ngClass]="{redBorder:OurOnTest()}">Is this
styled?</p>
<p [ngStyle]="{color:'green'}">Is this styled?</p>
<h3>Custom Property Binding</h3>
<fa-property-binding [result]="629"></fa-property-
binding>
<fa-event-binding></fa-event-binding>
```

Custom Event Binding

Let’s create a new Custom Event:

event-binding.component.ts

```
import { Component, EventEmitter, Output } from '@angular/core';

@Component({
  selector: 'fa-event-binding',
  template: `
    <button (click)="onClicked()">Click Me</button>
    <button (click)="onCustomClicked()">Click Me Custom Event</button>
  `,
  styles: []
})
export class EventBindingComponent  {
  @Output() clicked = new EventEmitter<string>();

  onClicked(){
    alert('wow!!! it works!');
  }
  onCustomClicked(){
    this.clicked.emit('Emitted Event Works!!');
  }
}
```



app works! wow

other works cool!!!

The Text | Shay | Shay

Is this styled?

Is this styled?

Custom Property Binding

629 Click Me Click Me Custom Event

Hello

there

Good

Bye

other.component.ts

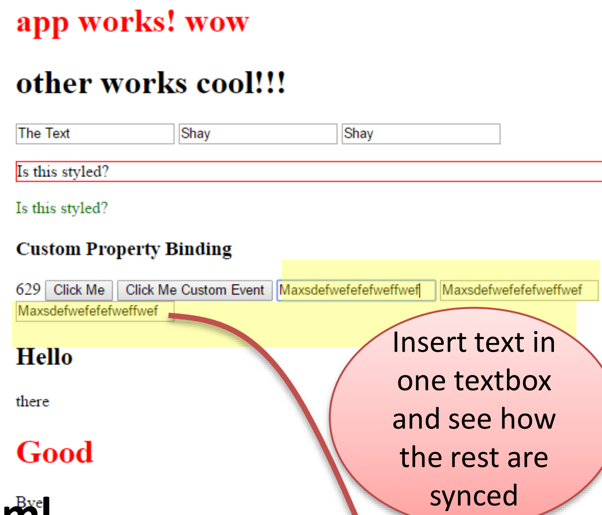
```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'fa-other',
  templateUrl: './other.component.html',
  styles:[`.redBorder{border: 1px solid red;}`]

})

export class OtherComponent implements OnInit {
  ourStringInterpolation = "Shay ";

  OurOnTest()
  {
    return true;
  }

  onClicked(value:string){
    alert("YAY! "+value);
  }
  constructor() { }
  ngOnInit() {
  }
```

other.component.html

```
<h1>
  other works cool!!!
</h1>

<input type="text" value="The Text">
<input type="text" value="{{ourStringInterpolation}}">
<input type="text" [value]="ourStringInterpolation">
<p [ngClass]="{redBorder:OurOnTest()}">Is this styled?</p>
<p [ngStyle]="{color:'green'}">Is this styled?</p>
<h3>Custom Property Binding</h3>
<fa-property-binding [result]="629"></fa-property-binding>
<fa-event-binding (clicked)="onClicked($event)"></fa-
event-binding>
```

Two-Way Data Binding

- Let’s create a new component:
  - using the terminal (Alt-F12) go to \src\app\other
  - ng g c two-way-binding --flat -it -is
  - Get rid of the spec.ts test file
  - Within **two-way-binding.component.ts** First get rid of the **Init** interface
    - Rename the selector to **fa-two-way-binding**
    - add a property json **person**, with fields name, age
    - In the template – insert two/three standard html input fields <input type=“text”> with ngModel binding to person.name
  - Within **other.component.html** we’d like to present the fields simply by calling **fa-two-way-binding**



app works! wow

other works cool!!!

The Text | Shay | Shay

Is this styled?

Is this styled?

Custom Property Binding

629 | Click Me | Click Me Custom Event | Maxsdefwefefefweffwef | Maxsdefwefefefweffwef

Maxsdefwefefefweffwef

Hello

there

Good

Bye

Insert text in one textbox and see how the rest are synced

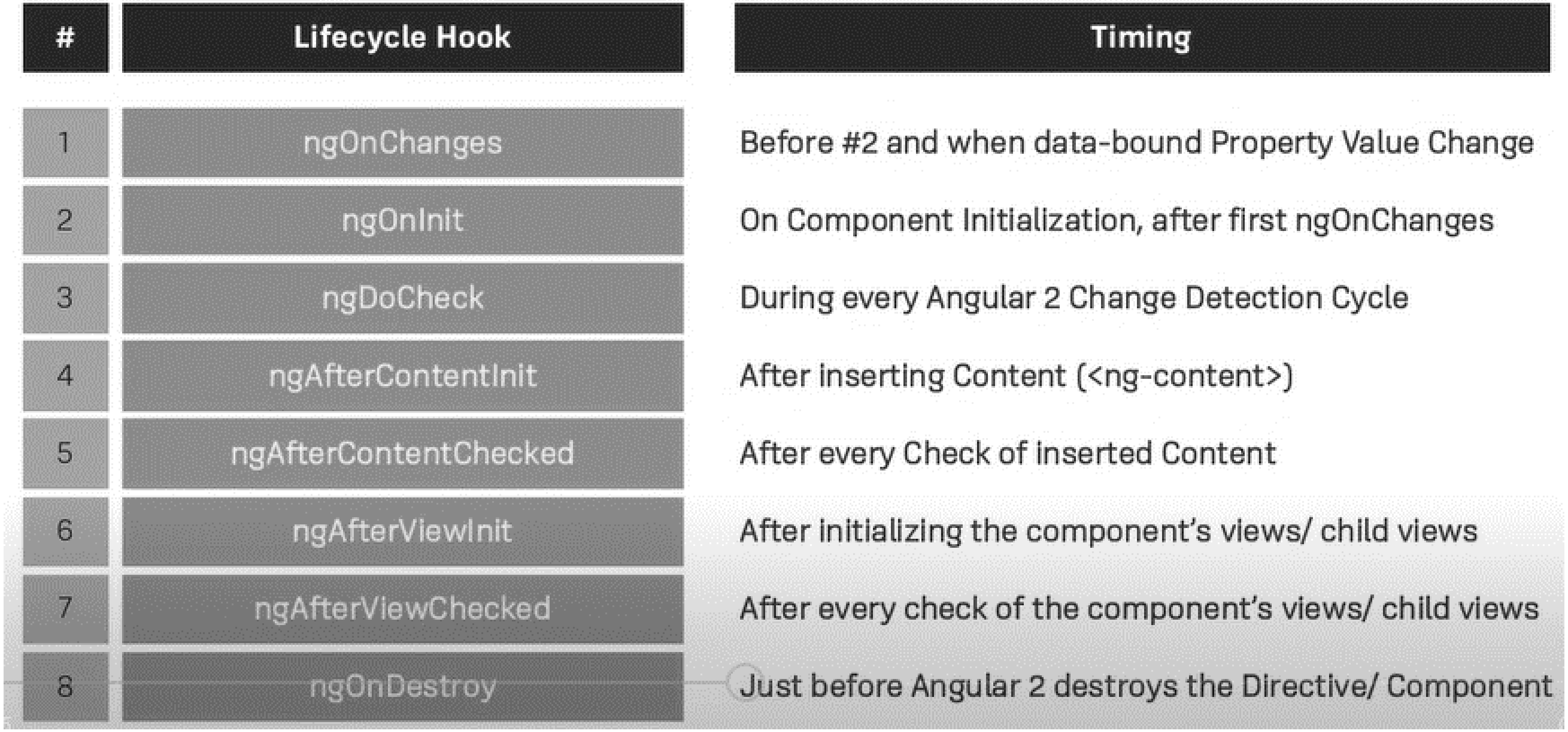
two-way-binding.component.ts

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

@Component({
  selector: 'fa-two-way-binding',
  template: `
    <input type="text" [(ngModel)]="person.name">
    <input type="text" [(ngModel)]="person.name">
    <input type="text" [(ngModel)]="person.name">
  `,
  styles: []
})
export class TwoWayBindingComponent  {
  person = {
    name: 'Max',
    age: 27
  };
}
```

other.component.html

```
<h1>
  other works cool!!!
</h1>

<input type="text" value="The Text">
<input type="text" value="{{ourStringInterpolation}}">
<input type="text" [value]="ourStringInterpolation">
<p [ngClass]="{redBorder:OurOnTest()}">Is this styled?</p>
<p [ngStyle]="{color:'green'}">Is this styled?</p>
<h3>Custom Property Binding</h3>
<fa-property-binding [result]="629"></fa-property-binding>
<fa-event-binding (clicked)="onClicked($event)"></fa-event-
binding>
<fa-two-way-binding></fa-two-way-binding>
```

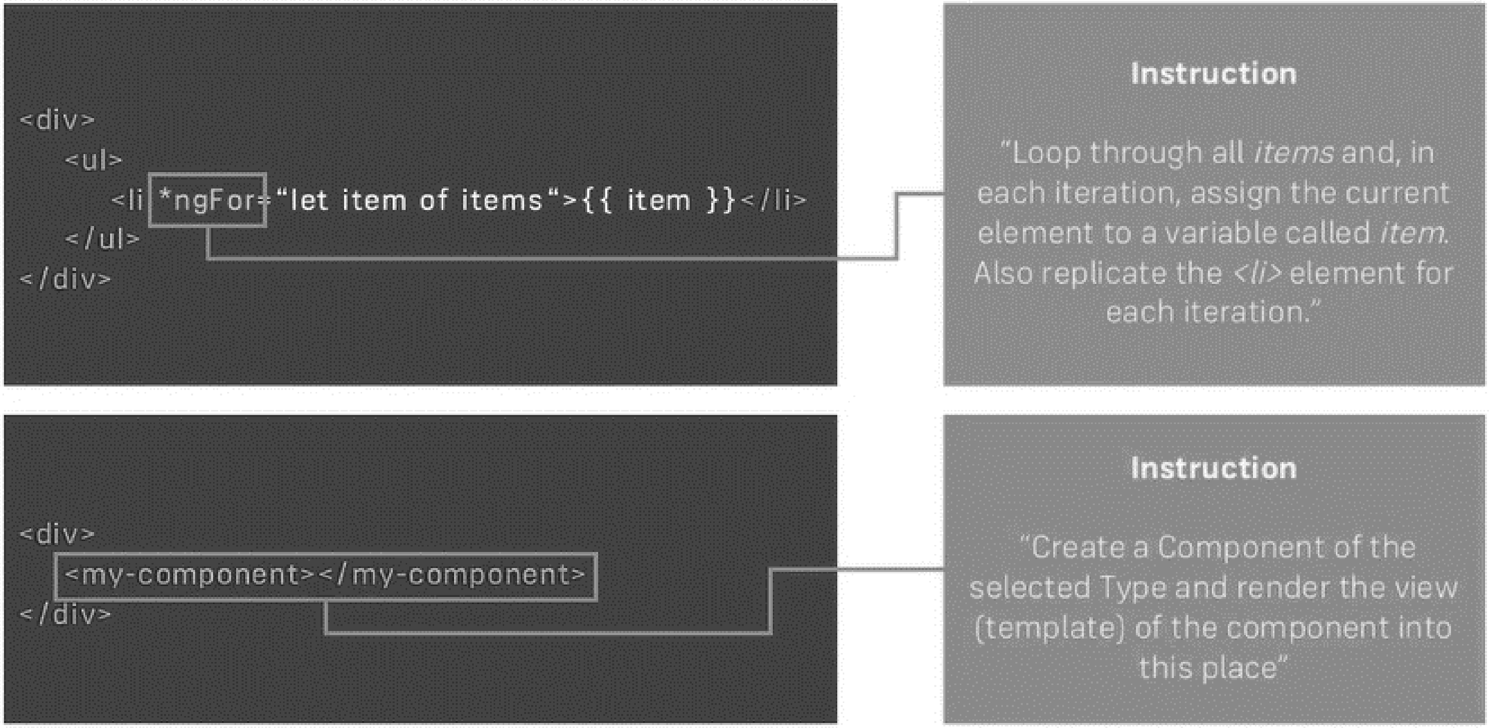
Components Lifecycle Overview



#	Lifecycle Hook	Timing
1	ngOnChanges	Before #2 and when data-bound Property Value Change
2	ngOnInit	On Component Initialization, after first ngOnChanges
3	ngDoCheck	During every Angular 2 Change Detection Cycle
4	ngAfterContentInit	After inserting Content (<ng-content>)
5	ngAfterContentChecked	After every Check of inserted Content
6	ngAfterViewInit	After initializing the component's views/ child views
7	ngAfterViewChecked	After every check of the component's views/ child views
8	ngOnDestroy	Just before Angular 2 destroys the Directive/ Component

Directives

- Components are a type of Directives
- Directives are instructions for DOM elements
- Examples:



```
<div>
    <ul>
        <li *ngFor=“let item of items“>{{ item }}</li>
    </ul>
</div>
```

Instruction

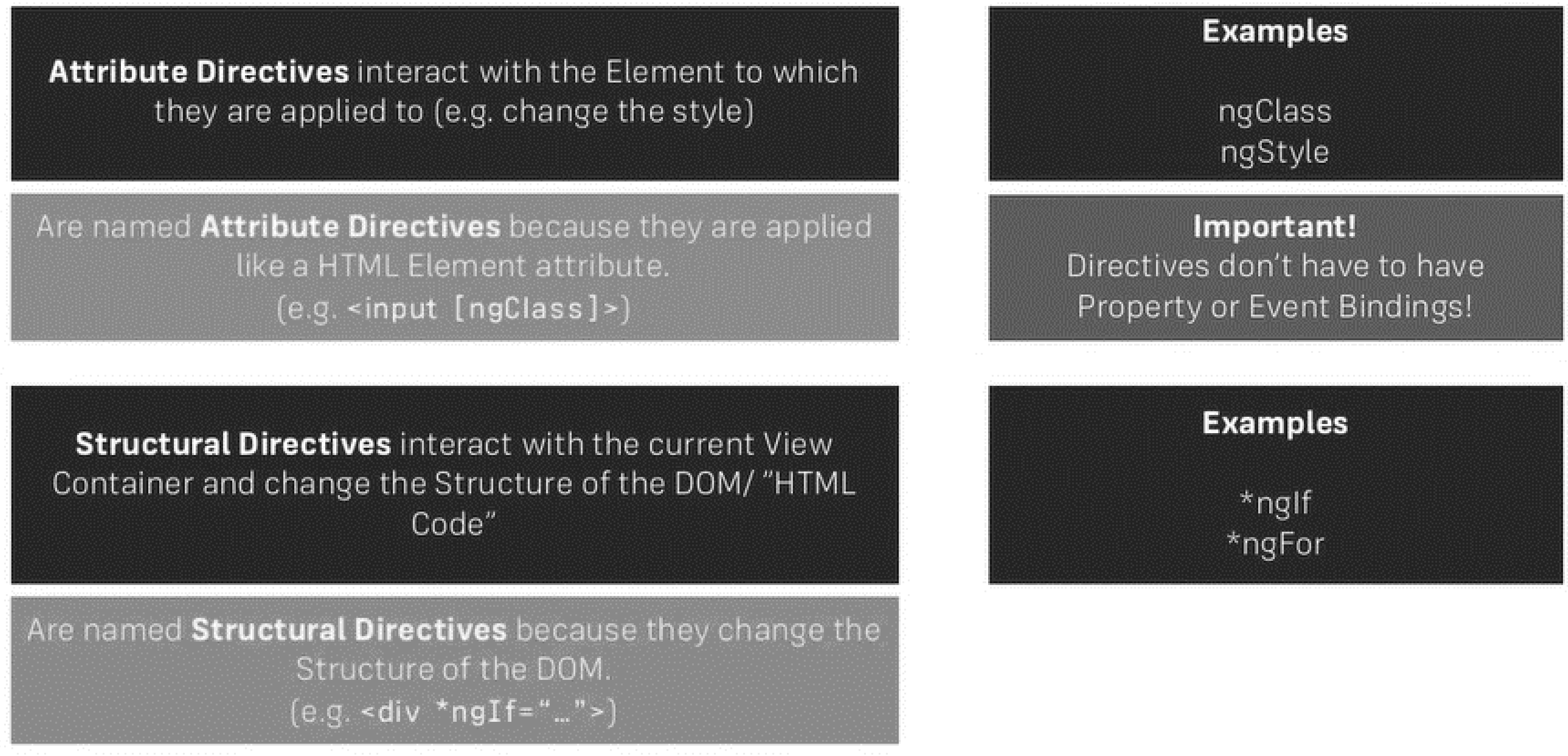
“Loop through all *items* and, in each iteration, assign the current element to a variable called *item*. Also replicate the *<li>* element for each iteration.”

```
<div>
    <my-component></my-component>
</div>
```

Instruction

“Create a Component of the selected Type and render the view (template) of the component into this place”

Attribute & Structural Directives in Angular 2



Attribute Directives interact with the Element to which they are applied to (e.g. change the style)

Are named **Attribute Directives** because they are applied like a HTML Element attribute.
(e.g. `<input [ngClass]>`)

Examples

ngClass
ngStyle

Important!
Directives don't have to have Property or Event Bindings!

Structural Directives interact with the current View Container and change the Structure of the DOM/ "HTML Code"

Are named **Structural Directives** because they change the Structure of the DOM.
(e.g. `<div *ngIf="…">`)

Examples

*ngIf
*ngFor

Attribute Directives

- Create a new project by entering the following in the terminal:
  - ng new directives --directory second-directives
  - Get rid of the test file (.spec.ts)
  - In App.component.ts delete the title
  - In app.component.html delete the content of the file and add a div with ngClass for the border and background
  - In app.component.css insert a definition for class of border, background and class
  - In the terminal window, cd to the project folder and run **ng serve**
  - **Run** (localhost:4200)

app.component.ts

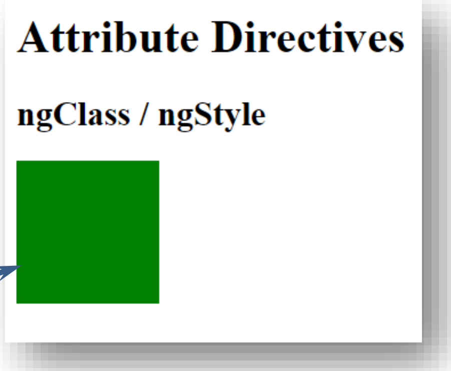
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
}
```

app.component.html

```
<h1>Attribute Directives</h1>
<h2>ngClass / ngStyle</h2>
<div [ngClass]="{border:false, background:true}"></div>
```

app.component.css

```
.border{
  border: 3px solid blue;
}

.background {
  background-color: green;
}

div{
  width: 100px;
  height: 100px;
}
```



Attribute Directives

ngClass / ngStyle

Background is green
Border is not blue (since it’s false), try changing it…

Structural Directives (*ngIf)

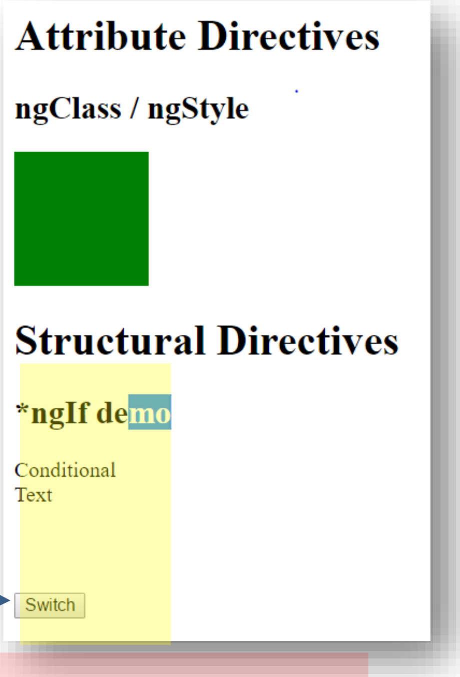
- In app.component.html add a <div> element under ngIf confidition (based on a switch parameter) and add a button to change the switch states upon each click on that button
- In app.component.ts add the switch property

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  private switch = true;

  onSwitch(){
    this.switch = !this.switch;
  }
}
```

app.component.html

```
<h1>Attribute Directives</h1>
<h2>ngClass / ngStyle</h2>
<div [ngClass]="{border:false,
background:true}"></div>

<h1>Structural Directives</h1>
<h2>*ngIf demo</h2>
<div *ngIf="switch">Conditional Text</div>
<button
(click)="onSwitch()">Switch</button>
```



Attribute Directives

ngClass / ngStyle

Structural Directives

*ngIf demo

Conditional Text

Switch

Notice!!!

The star in *ngIf is used in order to beautify the code

The process of converting between the * to regular binding – is called **De-Sugaring**

```
<div *ngIf="switch">Conditional Text</div>
```

==

```
<template [ngIf]="switch">
  <div>
    Conditional Text
  </div>
</template>
```

Structural Directives (*ngFor)

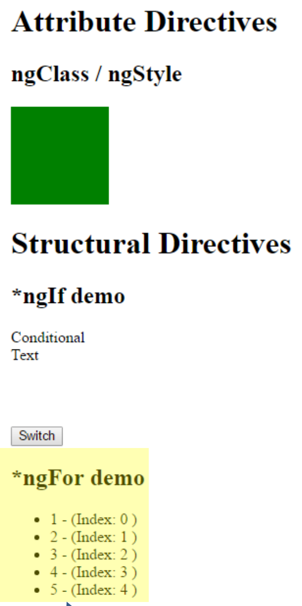
- In app.component.ts add an array of 5 items
- In app.component.html add an unordered list <ul>, with <li> items repeated over the property “items” using *ngFor.
- Present the index as well along with the values

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  private items = [1,2,3,4,5];

  private switch = true;
  onSwitch(){
    this.switch = !this.switch;
  }
}
```

app.component.html

```
<h1>Attribute Directives</h1>
<h2>ngClass / ngStyle</h2>
<div [ngClass]="{border:false, background:true}"></div>

<h1>Structural Directives</h1>
<h2>*ngIf demo</h2>
<div *ngIf="switch">Conditional Text</div>
<button (click)="onSwitch()">Switch</button>

<h2>*ngFor demo</h2>
<ul>
  <li *ngFor="let item of items; let i = index">{{item}} - (Index: {{i}} )</li>
</ul>
```



Attribute Directives

ngClass / ngStyle

Structural Directives

***ngIf demo**

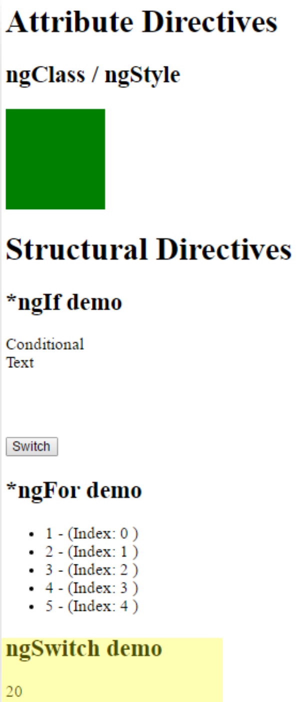
Conditional
Text

***ngFor demo**

- 1 - (Index: 0)
- 2 - (Index: 1)
- 3 - (Index: 2)
- 4 - (Index: 3)
- 5 - (Index: 4)

Structural Directives (*ngSwitch)

- In app.component.ts add an array of 5 items
- In app.component.html add an multiple paragraph (we will switch between them according to a value), and mark them with *ngSwitchCase, under a div element with an attribute [ngSwitch]



Attribute Directives

ngClass / ngStyle

🟩

Structural Directives

*ngIf demo

Conditional
Text

*ngFor demo

- 1 - (Index: 0)
- 2 - (Index: 1)
- 3 - (Index: 2)
- 4 - (Index: 3)
- 5 - (Index: 4)

ngSwitch demo

20

app.component.ts

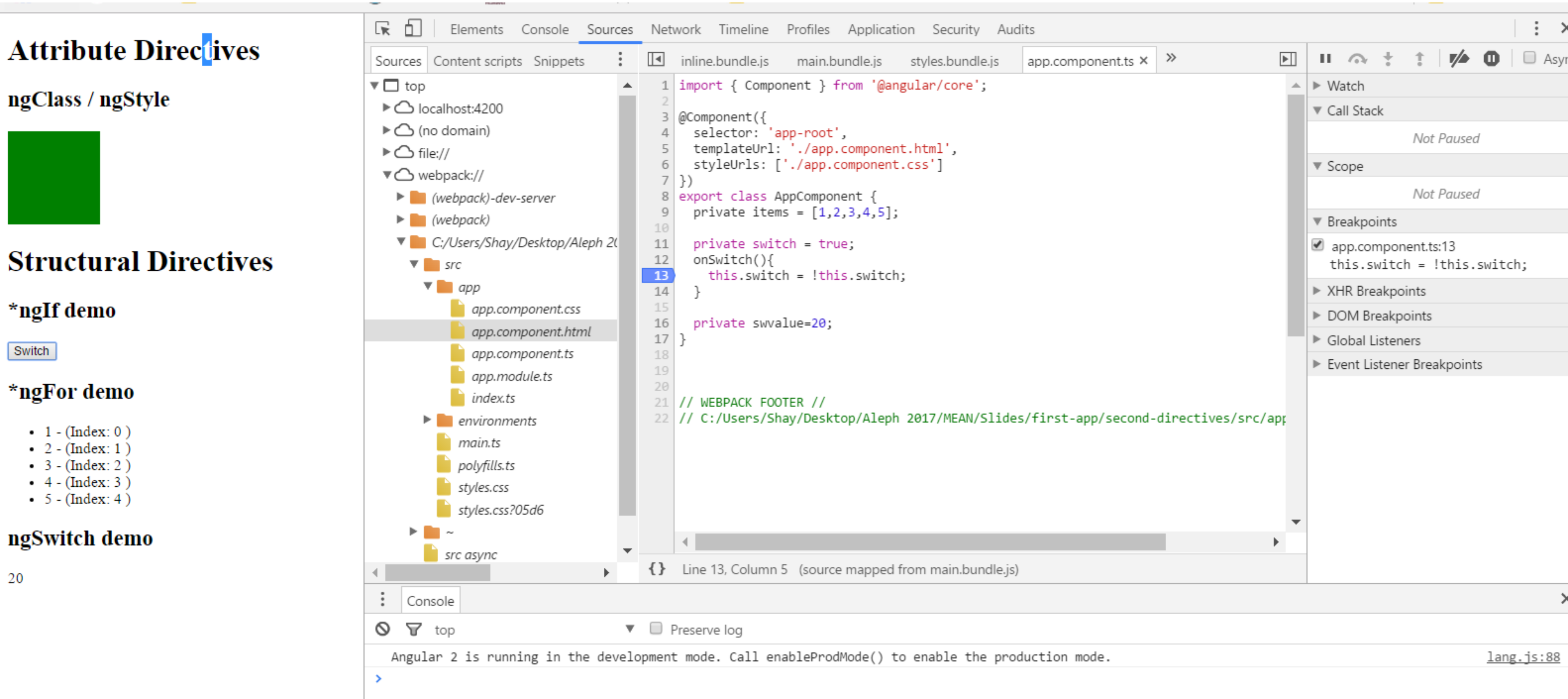
```
import { Component } from
'@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  private items = [1,2,3,4,5];

  private switch = true;
  onSwitch(){
    this.switch = !this.switch;
  }

  private swvalue=20;
}
```

app.component.html

```
<h1>Attribute Directives</h1>
<h2>ngClass / ngStyle</h2>
<div [ngClass]="{border:false, background:true}"></div>

<h1>Structural Directives</h1>
<h2>*ngIf demo</h2>
<div *ngIf="switch">Conditional Text</div>
<button (click)="onSwitch()">Switch</button>

<h2>*ngFor demo</h2>
<ul>
  <li *ngFor="let item of items; let i = index">{{item}} - (Index: {{i}} )</li>
</ul>

<h2>ngSwitch demo</h2>
<div [ngSwitch]="swvalue">
  <p *ngSwitchCase="10">10</p>
  <p *ngSwitchCase="20">20</p>
  <p *ngSwitchCase="30">30</p>
  <p *ngSwitchDefault>Default</p>
</div>
```

Debugging an Angular 2 app

- In Chrome, under F12, go to the sources, and find the orange rectangle – underneath you'll find your TypeScript code and you can breakpoint and debug.



Attribute Directives

ngClass / ngStyle

Structural Directives

*ngIf demo

Switch

*ngFor demo

- 1 - (Index: 0)
- 2 - (Index: 1)
- 3 - (Index: 2)
- 4 - (Index: 3)
- 5 - (Index: 4)

ngSwitch demo

20

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  private items = [1,2,3,4,5];

  private switch = true;
  onSwitch(){
    this.switch = !this.switch;
  }

  private swvalue=20;
}



// WEBPACK FOOTER //
// C:/Users/Shay/Desktop/Aleph 2017/MEAN/Slides/first-app/second-directives/src/app
```

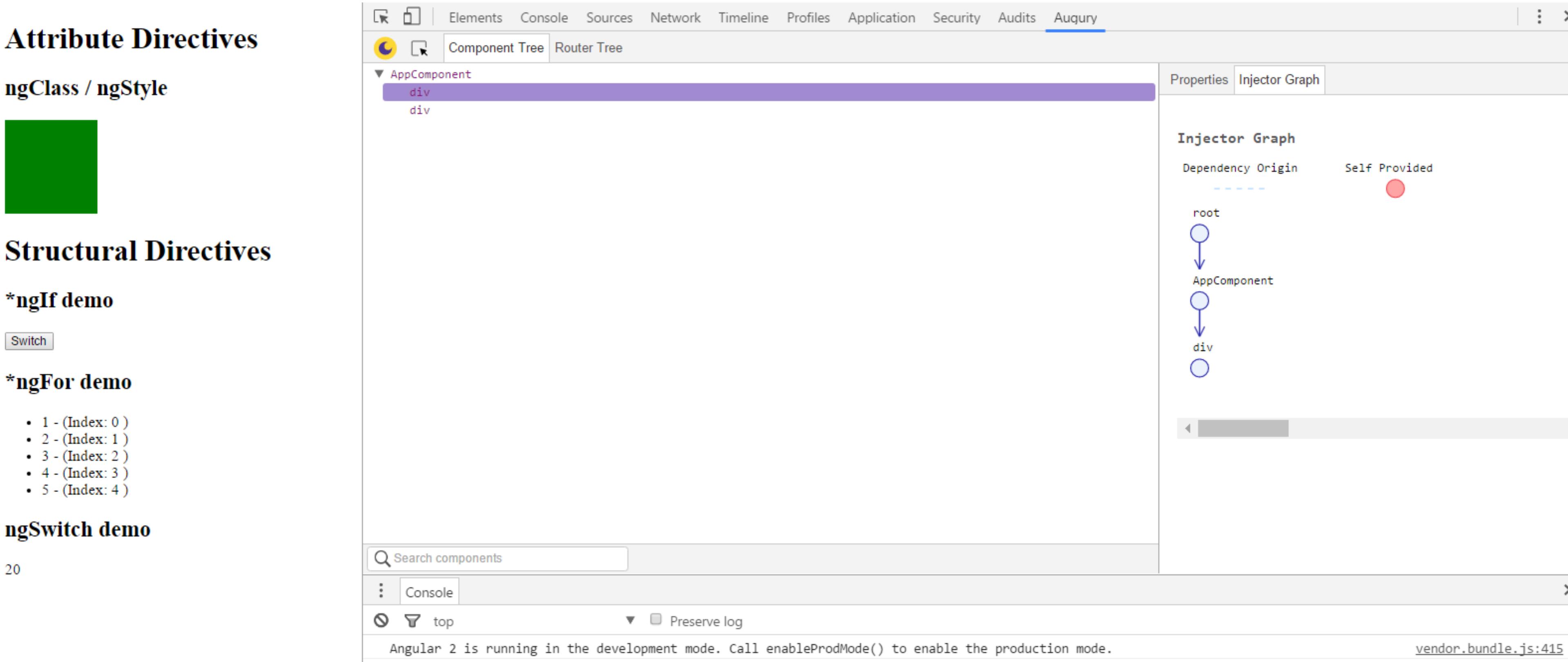
Line 13, Column 5 (source mapped from main.bundle.js)

Breakpoints
☑ app.component.ts:13
this.switch = !this.switch;

```
Angular 2 is running in the development mode. Call enableProdMode() to enable the production mode.    lang.js:88
```

Debugging Angular 2 - Augury

- You can also use Augury
- Get it from https://augury.angular.io/ (press install).
- Then in chrome it will appear as a tool under F12



Attribute Directives

ngClass / ngStyle

Structural Directives

*ngIf demo

Switch

*ngFor demo

- 1 - (Index: 0)
- 2 - (Index: 1)
- 3 - (Index: 2)
- 4 - (Index: 3)
- 5 - (Index: 4)

ngSwitch demo

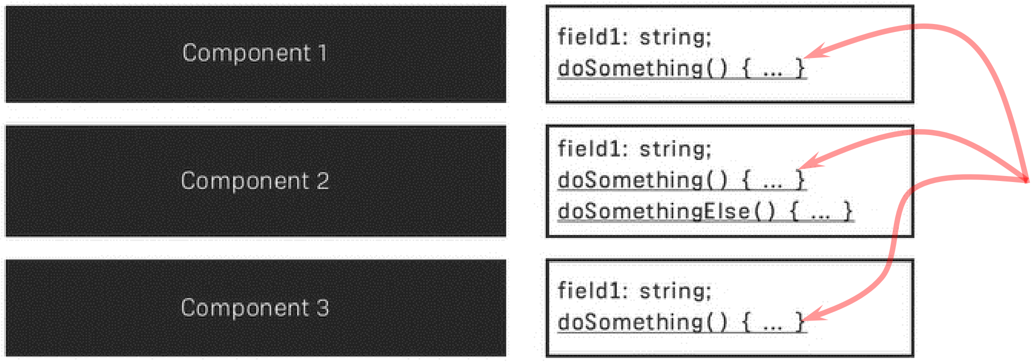
20

```
Angular 2 is running in the development mode. Call enableProdMode() to enable the production mode.    vendor.bundle.js:415
```

Angular 2 Services

- We don’t want to repeat ourselves!

No Services



Component 1

```
field1: string;
doSomething() { … }
```

Component 2

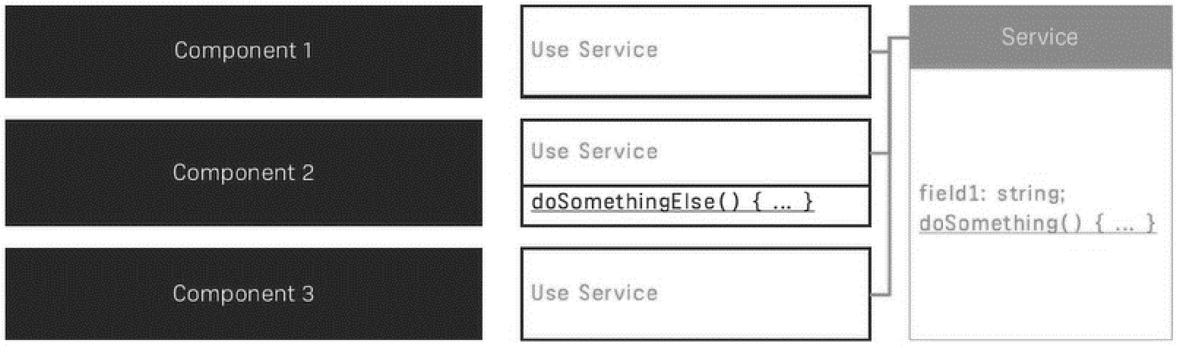
```
field1: string;
doSomething() { … }
doSomethingElse() { … }
```

Component 3

```
field1: string;
doSomething() { … }
```

Code is duplicated!

With Services



Component 1

Use Service

Component 2

Use Service

```
doSomethingElse() { … }
```

Component 3

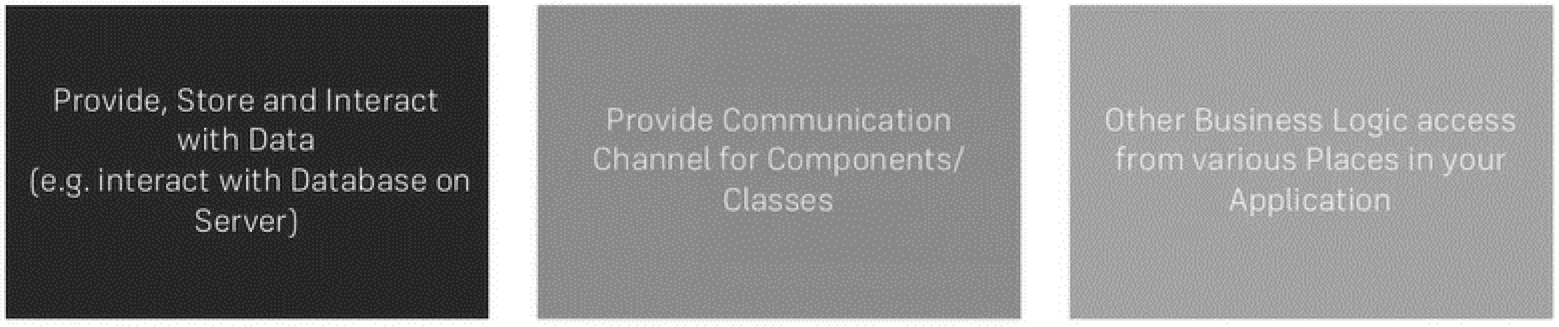
Use Service

Service

```
field1: string;
doSomething() { … }
```

No duplication

Common Service Tasks



Provide, Store and Interact with Data
(e.g. interact with Database on Server)

Provide Communication Channel for Components/ Classes

Other Business Logic access from various Places in your Application

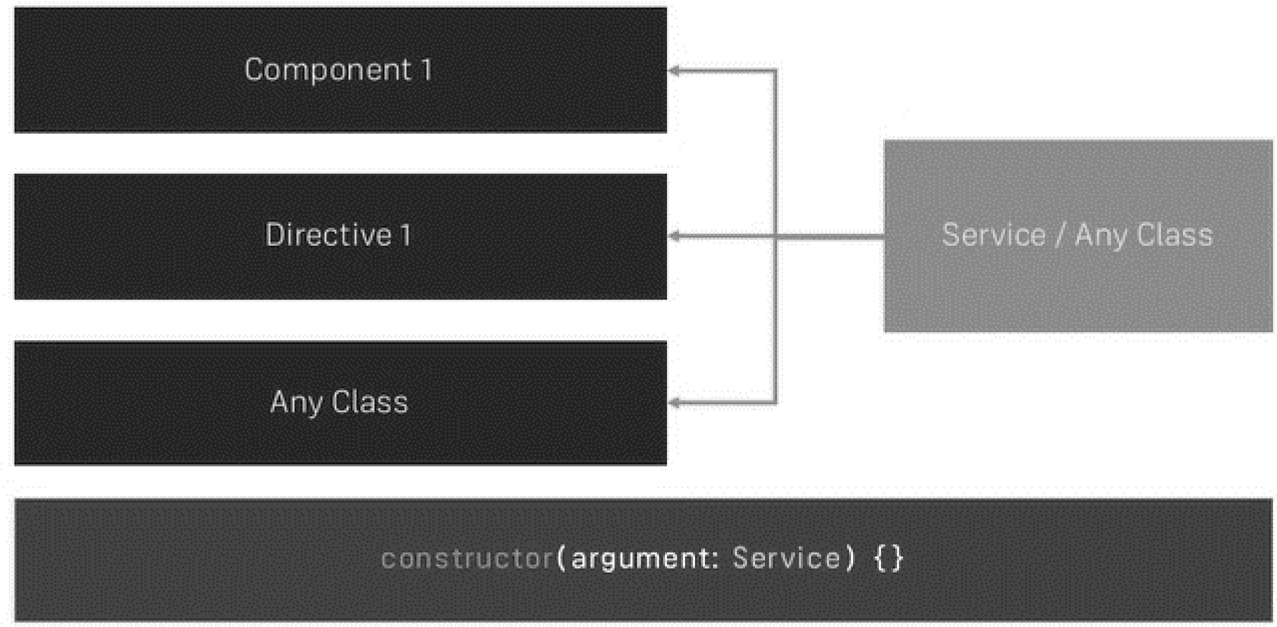
Creating a new Service

- In WebStorm terminal panel:
  - generate service
  - **ng g s log**
- Delete the .spec.ts test file
- Remote the @Injectable metadata and its import
- Remove the constructor
- Code should look like this:

```
export class LogService {
  writeToLog(logMessage: string){
    console.log(logMessage);
  }

}
```

Now… how can we use that service in our code?

Dependency Injection



```
constructor(argument: Service) {}
```

Service

- In app.component.ts add
  - a constructor with a call to our new LogService
  - A function onLog that uses the logSerice
- In app.component.html add a textbox with a variable #myLogInput and a button that calls the onLog function with that #myLogInput variable
- In app.module.ts add the new LogService to the providers list

app.component.ts

```
import { Component } from '@angular/core';
import {LogService} from "./log.service";

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  private items = [1,2,3,4,5];

  private switch = true;
  onSwitch(){
    this.switch = !this.switch;
  }

  private swvalue=20;

  constructor (private logService: LogService) {}

  onLog(value: string){
    this.logService.writeToLog(value);
  }


}
```

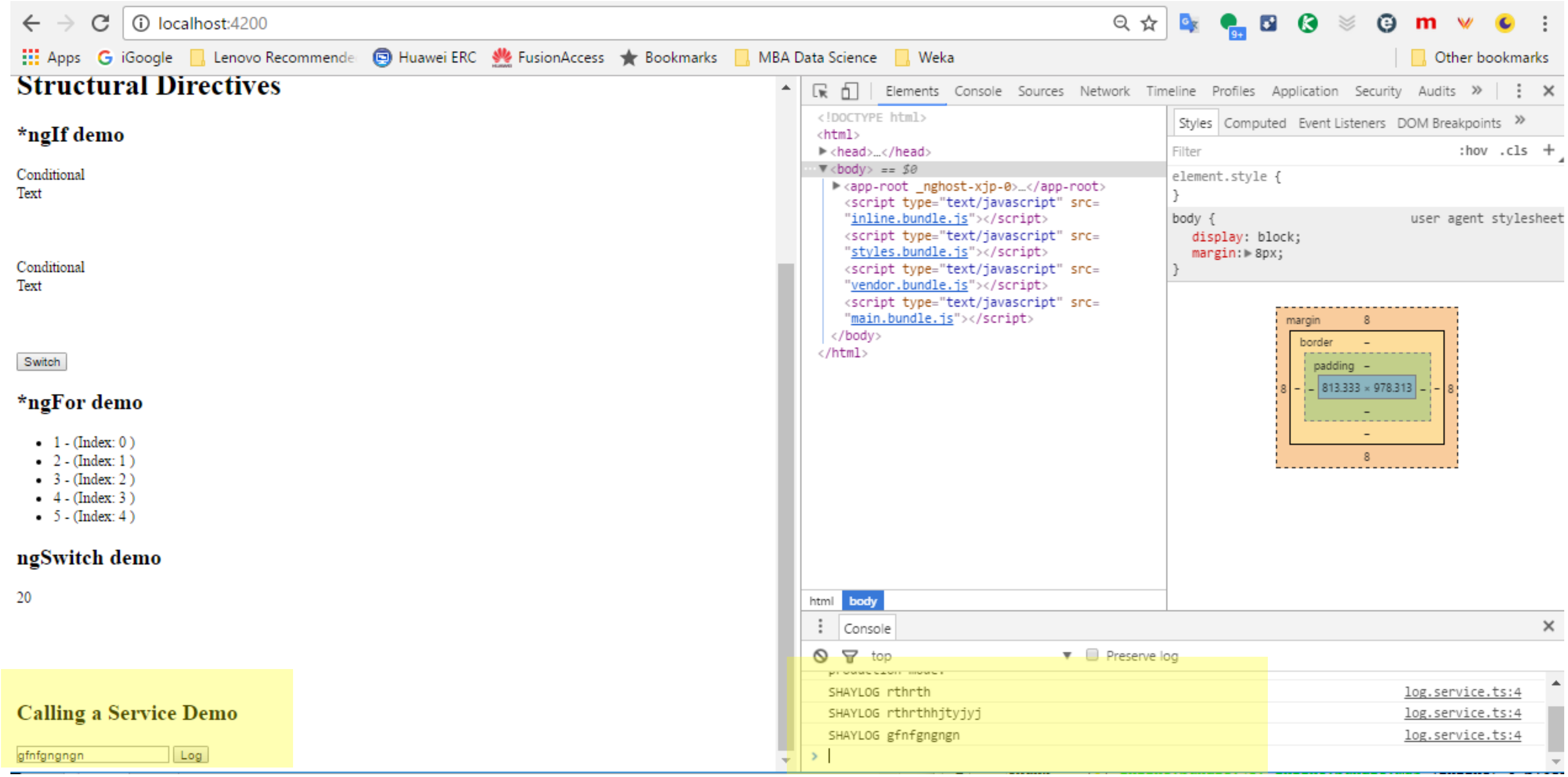
app.component.html

```
<h1>Attribute Directives</h1>
<h2>ngClass / ngStyle</h2>
<div [ngClass]="{border:false, background:true}"></div>

<h1>Structural Directives</h1>
<h2>*ngIf demo</h2>
<div *ngIf="switch">Conditional Text</div>

<template [ngIf]="switch">
  <div>
    Conditional Text
  </div>
</template>


<button (click)="onSwitch()">Switch</button>

<h2>*ngFor demo</h2>
<ul>
  <li *ngFor="let item of items; let i = index">{{item}} -
)</li>
</ul>

<h2>ngSwitch demo</h2>
<div [ngSwitch]="swvalue">
  <p *ngSwitchCase="10">10</p>
  <p *ngSwitchCase="20">20</p>
  <p *ngSwitchCase="30">30</p>
  <p *ngSwitchDefault>Default</p>
</div>

<h2>Calling a Service Demo</h2>
<input type="text" #myLogInput>
<button (click)="onLog(myLogInput.value)">Log</button>
```

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';
import {LogService} from "./log.service";

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [LogService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Test it in Chrome



localhost:4200

Structural Directives

*ngIf demo

Conditional
Text

Conditional
Text

*ngFor demo

- 1 - (Index: 0)
- 2 - (Index: 1)
- 3 - (Index: 2)
- 4 - (Index: 3)
- 5 - (Index: 4)

ngSwitch demo

20

Calling a Service Demo

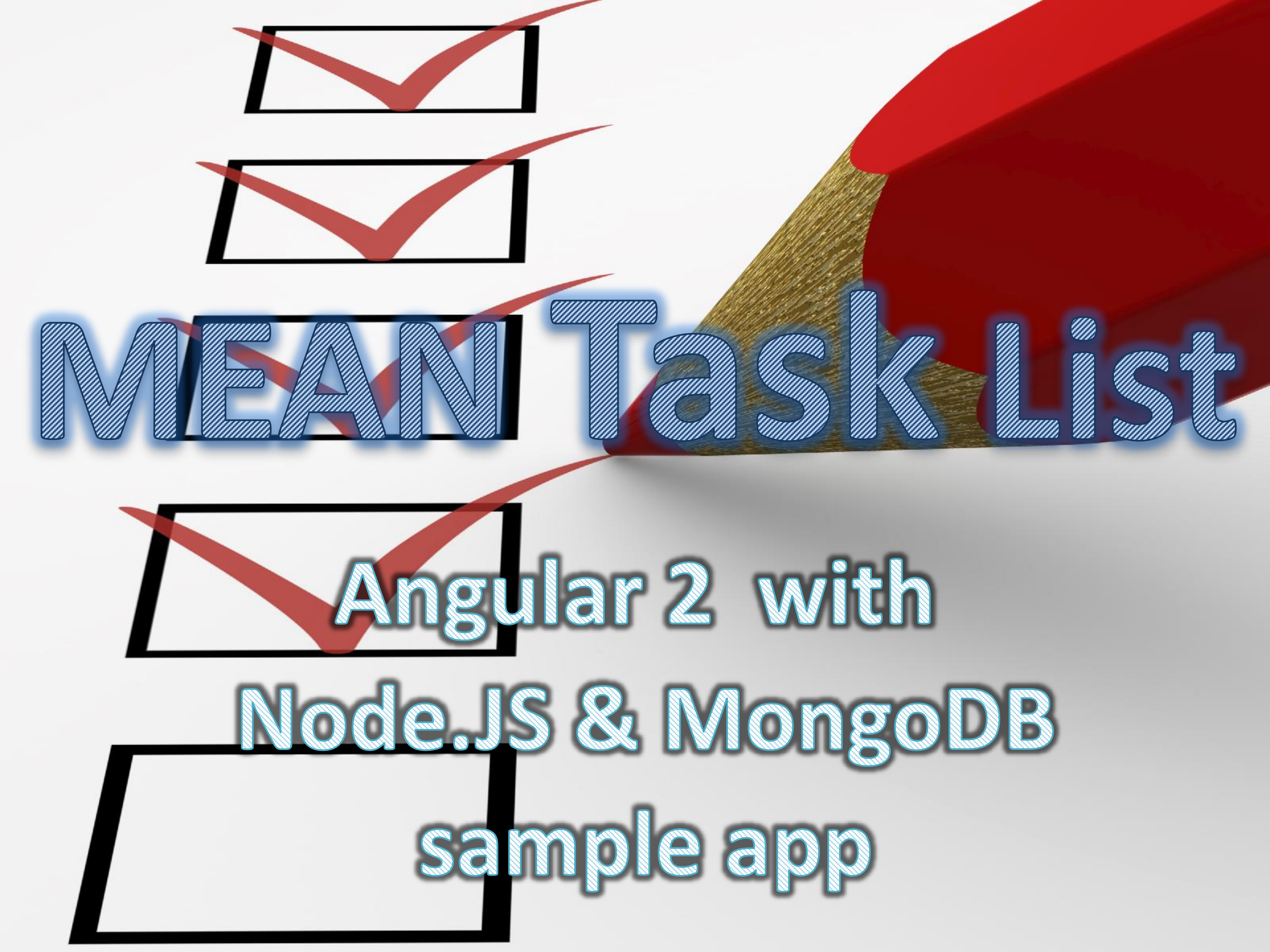
gfnfgngngn

```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body> == $0
  ▶<app-root _nghost-xjp-0>…</app-root>
   <script type="text/javascript" src=
   "inline.bundle.js"></script>
   <script type="text/javascript" src=
   "styles.bundle.js"></script>
   <script type="text/javascript" src=
   "vendor.bundle.js"></script>
   <script type="text/javascript" src=
   "main.bundle.js"></script>
  </body>
</html>
```

```
element.style {
}
body {                                user agent stylesheet
  display: block;
  margin:▶ 8px;
}
```

```
SHAYLOG rthrth                                log.service.ts:4
SHAYLOG rthrthhjtyjyj                         log.service.ts:4
SHAYLOG gfnfgngngn                            log.service.ts:4
>
```

- Add some text to the textbox, press on the newly added button and see the log printed in Chrome’s console (F12)



MEAN Task List

Angular 2 with Node.JS & MongoDB sample app

Environment

- Install Node.js
- Install MongoDB and run it
- Create a folder \MyTaskList
- Create a packet.json file and install libraries by:
  - **npm init**
    - Description: bla bla bla
    - Entry point: server.js
  - **npm install express body-parser ejs mongojs --save**

Folder Structure

- / (root)
  - server.js node.js file
  - Package.json required node.js packages
  - /views
  - /routes
  - /node_modules
  - /client angular 2 code

\server.js

```
var express = require('express');
var path = require('path');
var bodyParser = require('body-parser');

var index = require('./routes/index');
var tasks = require('./routes/tasks');
```

The files index.js & tasks.js will sit in \routes

```
var port = 3000;

var app = express();

//View Engine
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
app.engine('html', require('ejs').renderFile);
```

Path of our views (\views)

Views are rendered by EJS

```
// Set Static Folder
app.use(express.static(path.join(__dirname, 'client')));
```

share path of our angular stuff (\client)

```
// Body Parser MW
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: false}));
```

bodyParser parses incoming requests through *req.body*

```
app.use('/', index);
app.use('/api', tasks);
```

When user calls for / (root) – get ./routes/index.js

When user calls for /api – get ./routes/tasks.js

```
app.listen(port, function(){
  console.log('Server started on port '+port);
});
```

\routes\index.js

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res, next){
   res.render('index.html');
});

module.exports = router;
```

\views\index.html

```
<html>
 <head>
  <title>MyTaskList</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="bower_components/bootstrap/dist/css/bootstrap.css">
  <link rel="stylesheet" href="styles.css">
  <!-- 1. Load libraries -->
   <!-- Polyfill(s) for older browsers -->
  <script src="node_modules/core-js/client/shim.min.js"></script>
  <script src="node_modules/zone.js/dist/zone.js"></script>
  <script src="node_modules/reflect-metadata/Reflect.js"></script>
  <script src="node_modules/systemjs/dist/system.src.js"></script>
  <!-- 2. Configure SystemJS -->
  <script src="systemjs.config.js"></script>
  <script>
    System.import('app').catch(function(err){ console.error(err); });
  </script>
 </head>
 <!-- 3. Display the application -->
 <body>
  <my-app>Loading...</my-app>
 </body>
</html>
```

Here the app will be loaded with its Root angular component

\routes**tasks.js** router code

```
var express = require('express');
var router = express.Router();
var mongojs = require('mongojs');
var db = mongojs('mydb', ['tasks']); //local mongo installation, DB is mydb

// Get All Tasks
router.get('/tasks', function(req, res, next){
  db.tasks.find(function(err, tasks){
    if(err){
      res.send(err);
    }
    res.json(tasks);
  });
});
```

all tasks

http://localhost:3000/api/tasks

```
// Get Single Task
router.get('/task/:id', function(req, res, next){
  db.tasks.findOne({_id: mongojs.ObjectId(req.params.id)}, function(err, task){
    if(err){
      res.send(err);
    }
    res.json(task);
  });
});
```

Single

```
//Save Task
router.post('/task', function(req, res, next){
  var task = req.body;
  if(!task.title || !(task.isDone + '')){
    res.status(400);
    res.json({
      "error": "Bad Data"
    });
  } else {
    db.tasks.save(task, function(err, task){
      if(err){
        res.send(err);
      }
      res.json(task);
    });
  }
});
```

Create / Add

```
// Delete Task
router.delete('/task/:id', function(req, res, next){
  db.tasks.remove({_id: mongojs.ObjectId(req.params.id)}, function(err, task){
    if(err){
      res.send(err);
    }
    res.json(task);
  });
});
```

Delete

```
// Update Task
router.put('/task/:id', function(req, res, next){
  var task = req.body;
  var updTask = {};

  if(task.isDone){
    updTask.isDone = task.isDone;
  }

  if(task.title){
    updTask.title = task.title;
  }

  if(!updTask){
    res.status(400);
    res.json({
      "error":"Bad Data"
    });
  } else {
    db.tasks.update({_id: mongojs.ObjectId(req.params.id)},updTask, {}, function(err, task){
    if(err){
      res.send(err);
    }
    res.json(task);
  });
  }
});

module.exports = router;
```

Update

\client

- Within this folder, there are some config files:
  - Package.json
    - (not modified from angular.io except for the app name)
  - Tsconfig.json
    - Telling that the target is ES5
  - Typings.json
  - System.config.js

- Now we can run NPM install from the client folder
  - (that will install the required angular files)

\client\app\app.module.ts

Contains our code imports, and the bootstrapping component

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import {HttpModule} from '@angular/http';
import {FormsModule} from '@angular/forms';
import {AppComponent} from './app.component';
import {TasksComponent} from './components/tasks/tasks.component';

@NgModule({
  imports:      [ BrowserModule, HttpModule, FormsModule ],
  declarations: [AppComponent, TasksComponent],   Our angular components
  bootstrap: [AppComponent]   Our root component is AppComponent
})
export class AppModule { }
```

\client\app\app.component.ts

This is our core component

```
import { Component } from '@angular/core';
import {TaskService} from './services/task.service';Our task service to speak with the server

@Component({
  moduleId: module.id,
  selector: 'my-app',     Where to push the component template HTML code in the main html
  templateUrl: 'app.component.html',Core component HTML template of the code component
  providers:[TaskService] Our angular service for handling tasks for the task list
})

export class AppComponent { }
```

\client\app\app.component.ts

This is our core component

```
import { Component } from '@angular/core';
import {TaskService} from './services/task.service';

@Component({
  moduleId: module.id,
  selector: 'my-app',     Where to push the component template HTML code in the main html
  templateUrl: 'app.component.html',Core component HTML template of the code component
  providers:[TaskService] Our angular service for handling tasks for the task list
})

export class AppComponent { }
```

\client\app\app.component.html

```
<div class="container">  Bootstrap container
   <h1>MyTaskList</h1>
   <hr>
   <tasks></tasks>      Here we should show our tasks component
</div>
```

\client\app\main.ts

Bootstrapping our core Module (app.module.ts)

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app.module';

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

Installing Bower to get Bootstrap *(not a must)*

- In the root folder run:
  - **Npm install –g bower**
  - Create a file: **.bowerrc** with content:

```
{   "directory":"client/bower_components"}
```

(this tells bower where to install the components)

- **Bower install bootstrap –save**

(will also install Jquery automatically)

Creating our Tasks service

- Under **\client\app\** create a folder \services\

Dependency Injection in Angular

- Before we continue, we should understand Dependency Injection in Angular

```
export class Car {
 public engine: Engine;
 public tires: Tires;
 public description = 'No DI';
 constructor() {
  this.engine = new Engine();
  this.tires = new Tires();
 }
 // Method using the engine and tires
 drive() {
  return `${this.description} car with ` +
   `${this.engine.cylinders} cylinders and ${this.tires.make} tires.`;
 }
}
```

In this code, we create a Car class,
Our Car needs an **engine** and **tires**.

Instead of asking for them, the Car constructor instantiates **its own** copies from the very specific classes Engine and Tires

What if the **Engine** class evolves and its constructor requires a parameter?
…. We would need to modify the constructor with:

this.engine = new Engine(theNewParameter)

What if we want to put a different brand of tires on our Car?

Dependency Injection continued

- How to fix this?

```
constructor(public engine: Engine, public tires: Tires) { }
```

See what happened?
We moved the definition of the dependencies to the constructor.
Our Car class no longer creates an engine or tires. It just consumes them.

Now we can create a simple car with 4 cylinders and Flintstone tires.

```
let car = new Car(new Engine(), new Tires());
```

If someone extends the Engine class, that is not Car's problem:.

```
class Engine2 {
  constructor(public cylinders: number) { }
}
// Super car with 12 cylinders and Flintstone tires.
let bigCylinders = 12;
let car = new Car(new Engine2(bigCylinders), new Tires());
```

In Angular 2 – an injectable service must be defined as : **@Injectable()**

\client\app\components\services\task.service.ts

Send and Receive the tasks from the server

```
import {Injectable} from '@angular/core';
import {Http, Headers} from '@angular/http';
import 'rxjs/add/operator/map';
```

*Get a request and send a data as an observable using the **map** operator in async form*
(taken from ReactiveX project - http://reactivex.io/documentation/operators.html https://github.com/Reactive-Extensions/RxJS)

```
@Injectable()
```

Injectable service

```
export class TaskService{
  constructor(private http:Http){
```

Injecting Http service to our TaskService

```
    console.log('Task Service Initialized...');
  }

  getTasks(){
    return this.http.get('/api/tasks')
      .map(res => res.json());
```

Return as Observable with the map operator

```
  }

  addTask(newTask){
    var headers = new Headers();
    headers.append('Content-Type', 'application/json');
    return this.http.post('/api/task', JSON.stringify(newTask), {headers: headers})
      .map(res => res.json());
  }

  deleteTask(id){
    return this.http.delete('/api/task/'+id)
      .map(res => res.json());
  }

  updateStatus(task){
    var headers = new Headers();
    headers.append('Content-Type', 'application/json');
    return this.http.put('/api/task/'+task._id, JSON.stringify(task), {headers: headers})
      .map(res => res.json());
  }
}
```

\client**Task.ts**

Our Task class

```
export class Task{
    title: string;
    isDone: boolean;
}
```

Creating our Tasks component

- Under **\client\app\** create a folder \components\tasks\

- **This component will contain the Task objects**

\client\app\components\tasks\tasks.components.ts

Our tasks component

```
import { Component } from '@angular/core';
import {TaskService} from '../../services/task.service';
import {Task} from '../../../Task';

@Component({
  moduleId: module.id,
  selector: 'tasks',
  templateUrl: 'tasks.component.html'
})

export class TasksComponent {
  tasks: Task[];
```

Task objects

```
  title: string;
```

Injecting taskService

```
  constructor(private taskService:TaskService){
    this.taskService.getTasks()
      .subscribe(tasks => {
```

Subscribe to our observable

```
        this.tasks = tasks;
      });
  }

  addTask(event){
    event.preventDefault();
```

Prevent doing submit

```
    var newTask = {
      title: this.title,
```

this.title is what we type in the text box

```
      isDone: false
    }

    this.taskService.addTask(newTask)
      .subscribe(task => {
        this.tasks.push(task);
        this.title = '';
      });
  }

  deleteTask(id){
    var tasks = this.tasks;

    this.taskService.deleteTask(id).subscribe(data => {
      if(data.n == 1){
        for(var i = 0;i < tasks.length;i++){
          if(tasks[i]._id == id){
            tasks.splice(i, 1);
          }
        }
      }
    });
  }

  updateStatus(task){
    var _task = {
      _id:task._id,
      title: task.title,
      isDone: !task.isDone
    };

    this.taskService.updateStatus(_task).subscribe(data => {
      task.isDone = !task.isDone;
    });
  }
}
```

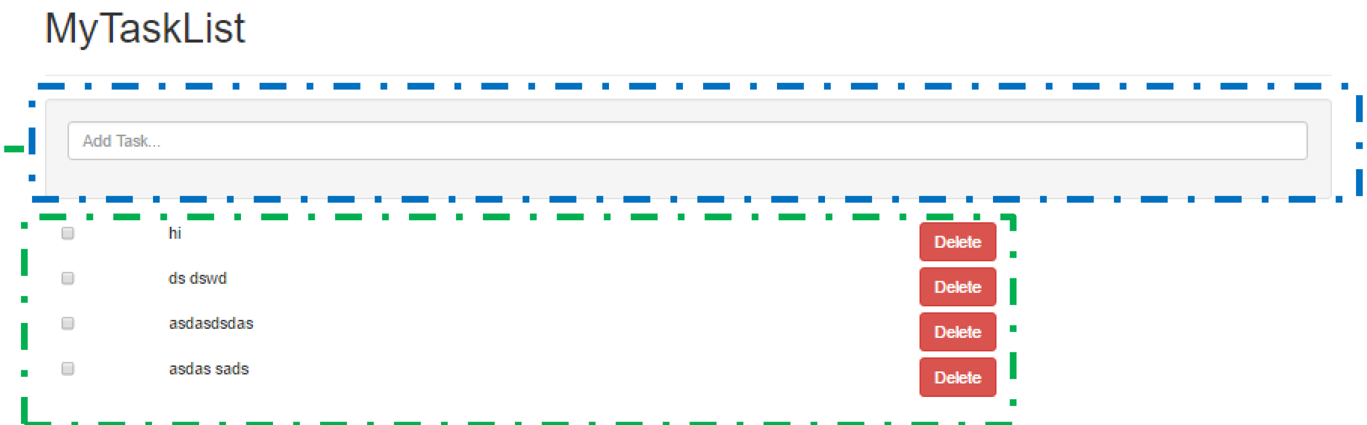
\client\app\components\tasks\tasks.components.html

Our tasks component template HTML

```
<form class="well" (submit)="addTask($event)">
    <div class="form-group">
     <input type="text" [(ngModel)]="title" name="title" class="form-control" placeholder="Add Task...">
    </div>
</form>
```

```
<div class="task-list">
  <div *ngFor="let task of tasks">
    <div class="col-md-1">
      <input type="checkbox" [checked]="task.isDone" (click)="updateStatus(task)">
    </div>
    <div class="col-md-7">
      {{task.title}}
    </div>
    <div class="col-md-4">
      <input type="button" (click)="deleteTask(task._id)" value="Delete" class="btn btn-danger">
    </div>
    <br><br>
  </div>
</div>
```

Loop over the tasks array



MyTaskList

Add Task...

□ hi Delete

□ ds dswd Delete

□ asdasdsdas Delete

□ asdas sads Delete

Last Remarks

- If you'd like to prevent stopping and re-running node.js on each code change, you can install NodeMon
  - **npm install –g nodemon**
  - **nodemon** *(in the project's folder)*
- Before launching the server, don't forget to run **MongoD**