Angular2

GitHub: <https://github.com/lightforsin/angular-seed>

Commands:

1. **npm install** 🡪 installs dependencies
2. **npm start** 🡪 starts the server

All dependencies and possible npm commands can be found in the *packages.json* file.

# The index.html file

<!-- 1. Load libraries -->

<!-- IE required polyfills, in this exact order -->

<script src="node\_modules/es6-shim/es6-shim.min.js"></script>

<script src="node\_modules/systemjs/dist/system-polyfills.js"></script>

<script src="node\_modules/angular2/bundles/angular2-polyfills.js"></script>

<script src="node\_modules/systemjs/dist/system.src.js"></script>

<script src="node\_modules/rxjs/bundles/Rx.js"></script>

<script src="node\_modules/angular2/bundles/angular2.dev.js"></script>

<!-- 2. Configure SystemJS -->

<script>

System.config({

packages: {

app: {

format: 'register',

defaultExtension: 'js'

}

}

});

System.import('app/boot')

.then(null, console.error.bind(console));

</script>

<!-- 3. Display the application -->

<body>

<my-app>Loading...</my-app>

</body>

# boot.ts

import {bootstrap} from 'angular2/platform/browser'

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

bootstrap(AppComponent);

# course.service.ts

export class CourseService {

getCourses(): string[] {

return ["Course 1", "Course 2", "Course 4"];

}

}

# auto-grow.directive.ts

import {Directive, ElementRef, Renderer} from 'angular2/core';

// **ElementRef (service)** is used for getting access to the **host** element.

// **Renderer (service)**, used for modifying elements

@Directive({ // directive decorator

selector: '[autoGrow]', // the [ ] means it will be applied as an HTML attribute (e.g. <input autoGrow />)

host: { // subscribe to events raised by this element

'(focus)': 'onFocus()', // when the **focus** event is fired, the onFocus() method is called

'(blur)': 'onBlur()' //\* events are placed between parentheses

}

})

export class AutoGrowDirective {

constructor(private el: ElementRef, private renderer: Renderer){

// when using “private” before the argument, the TS compiler automatically creates the **el** and **renderer** fields within the class

};

onFocus() {

this.renderer.setElementStyle(this.el.nativeElement, 'width', '200');

}

onBlur() {

this.renderer.setElementStyle(this.el.nativeElement, 'width', '120');

}

}

# courses.component.ts

import {Component} from 'angular2/core';

import {CourseService} from './course.service';

@Component({ // attribute, metadata, annotation 🡪 Component decorator

selector: 'courses', // the actual HTML tag 🡪 <courses>

template: `

<h2>Courses</h2>

{{ title }} // interpolation (one-way binding)

<ul>

<li \*ngFor="#course of courses">{{course}}</li> // iterating through a collection

</ul>`,

providers: [CourseService] // declaring dependencies

})

export class CoursesComponent { // exposing the component through “export”

title: string = "Course title";

courses: string[];

constructor(courseService: CourseService) { // dependency injection 🡪 the courseService is injected in the constructor

this.courses = courseService.getCourses();

};

}

# app.component.ts

import {Component} from 'angular2/core';

import {CoursesComponent} from './courses.component'; // importing component

@Component({

selector: 'my-app', // usage: <my-app>

template: '<h1>AJS 2 app</h1><courses></courses>',

directives: [CoursesComponent] // declaring directive dependencies

})

export class AppComponent { }

**Decorators**

They are prefixed with @.

They are functions and must be called as one, passing it a JSON object:

**Conventions**

A component should be named like: **<name>.component**

A service should be named like: **<name>.service**

A directive should be named like: **<name>.directive**

**Other aspects**

Each class is considered a module. That’s why we need to “export” classes to be used in other modules.

A component (@Component) has a template.

A directive (@Directive) does NOT have a template.

*Import {CoursesComponent} from ‘./courses.component’;* 🡪 “**./**” means “current folder”

The HTML template can be written on multiple lines if used between `` (so NOT ‘’)

*template: `*

*<h2>Courses</h2>*

*{{ title }}`*

**Binding**

The following *img* tags do the same thing:

template: `

<img src="{{ imageUrl }}" /> // recommended when displaying text

<img [src]="imageUrl" /> // this is the preferred way for DOM properties

<img bind-src="imageUrl" />`

**Class binding**

<button class="btn btn-primary" [class.active]="isActive">Submit</button>

// *isActive* is a bool property of the components’ class

**Style binding**

<button class="btn btn-primary" [style.backgroundColor]="isActive ? 'blue' : 'gray'">Submit</button>

**Event binding**

We use () for event binding and [] for property binding.

<button (click)="onClick()">Submit 1</button>

<button on-click="onClick()">Submit 2</button>

onClick($event) {

$event.stopPropagation();

console.log("Button clicked", $event);

}

**Two-way binding**

<input type="text" [value]="title" /> 🡪 one-way binding

<input type="text" [(ngModel)]="title" /> 🡪 two-way binding

<input type="text" bindon-ngModel="title" /> 🡪 two-way binding

**The component API**

Input properties

1st method:

import {Input} from 'angular2/core';

export class FavoriteComponent {

@Input() title: string; 🡪 the “title” property is exposed by declaring it as @Input()

}

This property can be assigned to in the HTML using property binding:

template: `<favorite [title]="the title"></favorite>`

We can also use an alias for the property:

@Input(‘alias-title’) title: string;

This way, we bind the property via the alias: <favorite [alias-title]="the title"></favorite>

2nd method:

@Component({

selector: '…',

template: ‘…’

inputs: ['isFavorite'] 🡪 simply specify the input properties inside the *inputs* array

})

export class FavoriteComponent {

*isFavorite*: boolean = false;

}

This method also allows aliases:

inputs: ['isFavorite: is-favorite'] 🡪 the alias is “is-favorite”

The downside with method 2 is the ‘magic strings’: inputs are strings, not actual entities

Output properties