Curso Angular 6
Intermedio dia 2

NEORIS

Index

- Unit Test
- Binidings
- Estilos
- UI

Por cada componente encontraremos un archivo con extension .spec

Jasmine es un behavior-driven development framework para probar código JavaScript. No dep ende de ningún otro framework de JavaScript. No requiere un DOM. Y tiene una sintaxis clara y obvia para que pueda escribir fácilmente pruebas.

karma js es un *test runner*, desarrollado por el equipo de angular, que nos permite automatizar algunas tareas de los frames de tests, como jasmine.

Jasmine primeros pasos

```
describe("A suite is just a function", function() {
  var a;

  it("and so is a spec", function() {
    a = true;

    expect(a).toBe(true);
  });
});

```

https://jasmine.github.io/api/edge/global

Ejemplo de testeo

TestBed es la primera y más importante de las utilidades para p oder hacer pruebas en Angular. Crea un módulo de prueba angu lar -una clase

Ejemplo:

```
const fixture = TestBed.createComponent(AppComponent);
const app = fixture.debugElement.componentInstance;
```

Expect → Metodo que evalua condiciones

Ejemplo

```
import { TestBed, async } from '@angular/core/testing';
import { RouterTestingModule } from '@angular/router/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
beforeEach(async(() => {
TestBed.configureTestingModule({
imports: [
RouterTestingModule
],
declarations: [
AppComponent
],
}).compileComponents();
}));

it('should create the app', () => {
const fixture = TestBed.createComponent(AppComponent);
const app = fixture.debugElement.componentInstance;
expect(app).toBeTruthy();
});

it(`should have as title 'aaa'`, () => {
const fixture = TestBed.createComponent(AppComponent);
const app = fixture.debugElement.componentInstance;
expect(app.title).toEqual('aaa');
});

it('should render title in a h1 tag', () => {
const fixture = TestBed.createComponent(AppComponent);
fixture.detectChanges();
const compiled = fixture.debugElement.nativeElement;
expect(compiled.querySelector('h1').textContent).toContain('Welcome to aaa!');
});

it('tag <span> con texto pepito', () => {
const fixture = TestBed.createComponent(AppComponent);
fixture.detectChanges();
const compiled = fixture.debugElement.nativeElement;
expect(compiled.querySelector('#sp').textContent).toContain('texto');
})

});
```

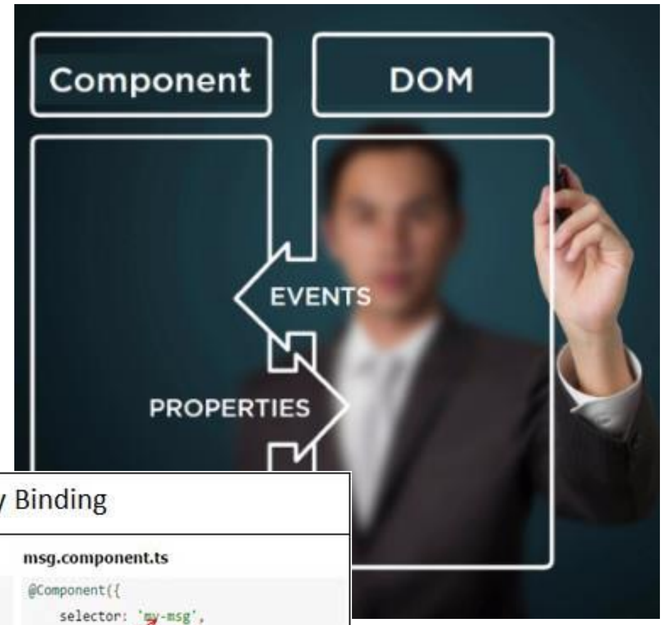
Binding

Property Binding

```
<img [src]='product.imageUrl'>
```

[ ] Binding Target

' ' Binding Source



Event Binding

Template

```
<h1>{{pageTitle}}</h1>
<img [src]='product.imageUrl'>
<button (click)='toggleImage()'>
```

Class

```
export class ListComponent {
 pageTitle: string = 'Product List';
 products: any[] = […];
 toggleImage(): void {…}
}
```

Angular 2 Property Binding

site.component.ts

```
@Component({
    selector: 'site-app',
    templateUrl: 'app/site.component.html',
    styleUrls: ['app/site.component.css']
})
export class SiteComponent {
    flag = true;
    website = {
        name : 'ConcretePage',
        url : 'http://www.concretepage.com',
        logo : 'images/logo.jpg',
        description: 'Learn angular 2 property binding.'
    }
}
```

msg.component.ts

```
@Component({
    selector: 'my-msg',
    templateUrl: 'app/msg.component.html'
})
export class MsgComponent {
    @Input() prefixMsg: string;
    @Input() siteName: string;
}
```

msg.component.html

```
{{prefixMsg}} {{siteName}}
```

concretepage.com

site.component.html

```
<br/><my-msg [prefixMsg]= "Website name is " [siteName] = "website.name"> </my-msg>
Logo: <img [src]="website.logo"/>
<p [ngClass]="'one two'"> Angular 2 Property Binding Example </p>
```

Component Property Binding

Element Property Binding

Directive Property Binding

Simplifiquemos con un ejemplo

En el HTML

```
<img [src]="path" >
```

En el ts

```
public path: string = '../assets/homero.jpg';
```

En el html

```
<button (click)="funcion()">click!</button>
```

En el ts

```
public funcion(){
console.log('hola mundo');
}
```

Otros ejemplos
https://angular.io/api/router/RouterLink

Estilos de componentes

Las aplicaciones angulares están diseñadas con CSS estándar. Eso significa que puede aplicar todo lo que sabe sobre hojas de estilo CSS, selectores, reglas y consultas de medios directamente a aplicaciones Angulares.
Además, Angular puede agrupar *estilos* de componentes con componentes, lo que permite un diseño más modular que las hojas de estilo regulares.

Para cada componente angular que escriba, puede definir no solo una plantilla HTML, sino también los estilos CSS que acompañan a esa plantilla, especificando los selectores, las reglas y las consultas de medios que necesite.
Una forma de hacerlo es establecer la propiedad de styles en los metadatos del componente. La propiedad de styles toma una matriz de cadenas que contienen código CSS. Usualmente le das una cadena, como en el siguiente ejemplo:

Utilice el selector de pseudoclases del :host para orientar los estilos en el elemento que *aloja* el componente (a diferencia de los elementos de segmentación *dentro de* la plantilla del componente).
Ejemplo.
:host { display: block; border: 1px solid black; }

https://code.i-harness.com/es/docs/angular/guide/component-styles

```
@Component({
  selector: 'app-root',
  template: `
    <h1>Tour of Heroes</h1>
    <app-hero-main [hero]="hero"></app-hero-main>
  `,
  styles: ['h1 { font-weight: normal; }']
})
export class HeroAppComponent {
/* . . . */
}
```

UI – Framework - Clarity

https://vmware.github.io/clarity/documentation/v0.11/get-started

Es un framework desarrollado por Vmware, facilitando la creación de sitios Robustos con componentes ya creados y disponibles

Estructura

```
<div class="main-container">
    <div class="alert alert-app-level">
        ...
    </div>
    <header class="header header-6">
        ...
    </header>
    <nav class="subnav">
        ...
    </nav>
    <div class="content-container">
        <div class="content-area">
            ...
        </div>
        <nav class="sidenav">
            ...
        </nav>
    </div>
</div>
```

www.neoris.com

Dante Panella
Master
Dante.panella@neoris.com

NEORIS