

# 0 - Introducción al testing de aplicaciones Angular

Pruebas de Angular *sin Angular*

## 0 - Karma y Jasmine

- ✓ preconfigurado
- ✓ autogenerado
- ✓ scriptable

# Mejoras

karma.conf.js

```
plugins: [  
  require('karma-jasmine'),  
  require('karma-chrome-launcher'),  
  require('karma-mocha-reporter'),  
  require('karma-jasmine-diff-reporter'),  
  require('karma-coverage-istanbul-reporter'),  
  require('@angular-devkit/build-angular/plugins/karma'),  
],  
browsers: ['ChromeHeadless'],  
reporters: ['jasmine-diff', 'mocha'],
```

- 1 - Trigger
- 2 - Build
- 3 - Run
- 4 - Report
- 5 - Coverage

# 1 - Probando un servicio como una clase

## Issue: Testing minimalista del LogicService

Probar un servicio con métodos de lógica de negocio.

Sin dependencias.

Métodos *puros*.

## S.U.T: LogicService

```
export class LogicService {  
  public slugify(text: string): string {  
    return text  
      .toLowerCase()  
      .trim()  
      .replace(/[\s\W-]+/g, '-');  
  }  
}
```

## Guías:

- **GWT:** Given When Then
- **AAA:** Arrange Act Assert
- **DBI:** describe before it

## Test: LogicService - Test

```
describe('GIVEN: the LogicService', () => {  
  let sut: LogicService;  
  beforeEach(() => {  
    // Arrange  
    sut = new LogicService();  
  });  
  it('WHEN slugifies Angular 10.1 THEN returns angular-10-1', () => {  
    // Act  
    const actual = sut.slugify('Angular 10.1');  
    // Assert  
    const expected = 'angular-10-1';  
    expect(actual).toEqual(expected);  
  });  
});
```

# snippets

```
{
  "Jasmine Given When Then": {
    "prefix": "ab-jsm-gwt",
    "body": [
      "describe('GIVEN: $1', () => {",
      "  let sut;",
      "  beforeEach(() => {",
      "    // Arrange",
      "    sut = null;",
      "  });",
      "  it('WHEN $2 THEN $3', () => {",
      "    // Act",
      "    const actual = null;",
      "    // Assert",
      "    const expected = null;",
      "    expect(actual).toEqual(expected);",
      "  });",
      "});",
    ],
    "description": "Esqueleto GWT con Jasmine"
  },
}
```



## Ejercicio

**GIVEN:** the composeTaskView method  
**WHEN:** we have two tasks both undone  
**THEN:** returns { **total:** 2, **pending :** 2 }

## 2 - Probando un componente como una clase

### Issue: Testing minimalista de un componente

Probar un componente con propiedades de datos.

Sin dependencias.

Sin importar *la presentación*.

## S.U.T: AboutComponent

```
export class AboutComponent implements OnInit {  
  title = 'Angular Budget';  
  constructor() {}  
  
  ngOnInit(): void {}  
}
```

## Test: AboutComponent - Test

```
describe('GIVEN: the AboutComponent', () => {  
  let sut: AboutComponent;  
  beforeEach(() => {  
    // Arrange  
    sut = new AboutComponent();  
  });  
  it('WHEN ask for title THEN equals Angular Budget', () => {  
    // Act  
    const actual = sut.title;  
    // Assert  
    const expected = 'Angular Budget';  
    expect(actual).toEqual(expected);  
  });  
});
```

## Ejercicio

**GIVEN:** the DateTimeComponent component

**WHEN:** we ask the userFormat

**THEN:** returns default `dd/MM/yyyy`

## 3 - Probando unidades y espiando dependencias

### Issue: Pruebas de un servicio con dependencias usando espías

Probar un servicio con dependencias (Title).

Queremos hacer tests unitarios.

Usamos un **dobble** en lugar de la dependencia original.

Con *Jasmine* lo aconsejable es usar un **spy**

Probamos el buen **comportamiento** con los colaboradores

## S.U.T: UtilService

```
export class UtilService {  
  private siteTitle = 'Angular.Budget';  
  
  constructor(private titleService: Title) {}  
  
  public setDocumentTitle(title: string): void {  
    const documentTitle = title ? `${title} | ${this.siteTitle}` : this.siteTitle;  
    this.titleService.setTitle(documentTitle);  
  }  
}
```

## Test: UtilService - Test

```
describe('GIVEN the UtilsService', () => {  
  // UtilService es el Subject Under Test  
  let utilServiceSUT: UtilService;  
  // TitleService es un colaborador (una dependencia)  
  let titleServiceSpy: jasmine.SpyObj<Title>;  
  beforeEach(() => {  
    // Arrange  
    // El colaborador es un doble  
    titleServiceSpy = jasmine.createSpyObj('TitleService', {  
      // Metodo y respuesta predefinida para void  
      setTitle: undefined,  
    });  
    utilServiceSUT = new UtilService(titleServiceSpy);  
  });  
  it('WHEN setting a title SHOULD send that title to Angular Service', () => {  
    // Act  
    utilServiceSUT.setDocumentTitle('Pruebas unitarias');  
    // Assert  
    // Prueba de comportamiento testeando el envío a un colaborador  
    // Espíamos para saber el uso que se hace del colaborador  
    const actual = titleServiceSpy.setTitle.calls.mostRecent().args[0];  
    const expected = 'Pruebas unitarias | Angular.Budget';  
    expect(actual).toEqual(expected);  
  });  
});
```



## Ejercicio

**GIVEN:** the UtilService

**WHEN:** we call setDocumentTitle without arguments

**THEN:** sets the default `Angular.Budget`

## 4 - Probando código asíncrono

### Issue: Prueba de un servicio asíncrono

Probar un servicio con dependencias asíncrona (*HttpClient*).

Los tests tienen que ser **asíncronos**.

Podemos probar:

- la llamada (*url*)
- la respuesta (*subscripción*)

Jasmine Asynchronous Work

## S.U.T: DataService

```
export class DataService {  
  private rootUrl = `https://api-base.herokuapp.com/api/pub`;   
  
  constructor(private httpClient: HttpClient) {}  
  
  getProjects$(): Observable<Project[]> {  
    return this.httpClient.get<Project[]>(`${this.rootUrl}/projects`);  
  }  
}
```

## Test: DataService - Test

```
describe('GIVEN: A DataService', () => {  
  let sut: DataService;  
  let httpClientSpy: jasmine.SpyObj<HttpClient>;  
  
  beforeEach(() => {  
    // Arrange  
    httpClientSpy = jasmine.createSpyObj('HttpClient', {  
      // Respuesta predefinida asíncrona  
      get: of([{ id: 'ok', title: 'ok' }]),  
    });  
    sut = new DataService(httpClientSpy);  
  });  
});
```

```
it('WHEN call the getProjects THEN the url is the expected', () => {  
  // Act  
  sut.getProjects$.subscribe();  
  // Assert  
  // Prueba de comportamiento testeando una llamada al colaborador  
  // Comprobación de argumentos  
  const actual = httpClientSpy.get.calls.mostRecent().args[0];  
  const expected = 'https://api-base.herokuapp.com/api/pub/projects';  
  expect(actual).toEqual(expected);  
});
```

```
it('WHEN call the getProjects$ THEN returns an observable of a project list', () => {  
  // Act  
  let actual: Object[];  
  // La suscripción a observables funciona  
  sut.getProjects$.subscribe({  
    next: data => (actual = data),  
  });  
  // Assert  
  // Prueba de estado directo testeando la respuesta obtenida  
  const expected = [{ id: 'ok', title: 'ok' }];  
  expect(actual).toEqual(expected);  
});  
});
```

## Ejercicio

**GIVEN:** the DataService

**WHEN:** we ask the project `learning-to-test`

**THEN:** returns "{ id: 'learning-to-test' , title : 'Learning to Test'}"

Repositorio: [angularbuilders/angular-budget/test\\_0\\_no-angular](#)

By [Alberto Basalo](#)