

Testing

¿Qué es?

Comprobación automática del código

- Organizada por casos
- Cada caso comprueba un aspecto
- Comparando el resultado obtenido con el esperado

¿Para qué sirve?

¡Para garantizar que todo funciona!

- Que el nuevo código es correcto
- Que no se ha roto nada de lo anterior
- Que una refactorización no ha introducido bugs

¿En JavaScript?

Una práctica que va penetrando poco a poco

- Aunque sigue sin estar muy extendida
- Necesaria para aplicaciones complejas
- En general, una garantía de calidad

Testing

Hay muchos tipos de tests:

- Unitarios: comprueban un componente o una parte específica del código
- Integración: comprueban la interacción de componentes
- Aceptación: comprueban los requisitos del proyecto
- Regresión: comprueban la corrección de cambios
- etc...

Tests Unitarios

La idea de test unitario es muy simple:

- Dado un componente del sistema
- Para cada caso posible
- Comprobar que se comporta de la manera adecuada

Test Unitarios

```
var Contador = ProJS.Class.extend({  
  init: function() {  
    this.i = 0;  
  },  
  get: function() {  
    return this.i;  
  },  
  inc: function() {  
    this.i++;  
  },  
  dec: function() {  
    this.i--;  
  },  
  reset: function() {  
    this.i = 0;  
  }  
});
```

Jasmine


Estupenda librería de testing

- Al estilo rspec
- Sencilla
- Potente
- <http://pivotal.github.com/jasmine/>

Jasmine

¿Qué pinta tiene?

- <tema5/jasmine-1/index.html>



```
describe("Conjunto de tests", function() {  
  it("debería ser un caso válido", function() {  
    expect(true).toBe(true);  
  });  
  it("debería ser un caso con error", function() {  
    expect(true).toBe(false);  
  });  
});
```

Jasmine

Test del contador con Jasmine

- [tema5/jasmine-2/index.html](#)

Jasmine

Test asíncronos

- ¿Cómo testearías que esta función llama al **cb** con **true**?

```
function asyncFn(cb) {  
  setTimeout(function() { cb(true); }, 250);  
}
```

- <tema5/jasmine-3/index.html>

Jasmine

```
describe("Test asíncrono", function() {  
  it("debería llamar al callback con true", function() {  
    var result,  
        callback = function(response) { result = response; };  
    ➡ runs(function() {  
        asyncFn(callback);  
    });  
    ➡ waitsFor(function() {  
        return result == true;  
    }, 300);  
    ➡ runs(function() {  
        expect(result).toBe(true);  
    });  
  });  
});
```

Intermedio: Jasmine

¡Testea alguna de las funciones que hemos visto!

- La que te parezca más confusa
- Documentación y “matchers” de Jasmine en
➡ <http://pivotal.github.com/jasmine/>

Spam Mode: ON

Al escribir test JS acaba surgiendo un problema:

- ¡Los datos!
- ¿De dónde saco datos válidos para testear?
- ¿Del servidor?
 - No es fácil de conseguir modificar/resetear un set de datos cada vez que ejecuto un test
 - Dependencia del backend
- Lo ideal sería:
 - Factorías de datos (estilo FactoryGirl)
 - Simular la interacción con el servidor de forma inocua

Solipsist.js

Solipsist.js es una librería auxiliar para testear

➡ <https://github.com/WeRelax/solipsist-js>

- Tests JS aislados
- Factorías
- Mocking de peticiones AJAX
- Otro uso: programar el frontend independiente del backend