

INTRODUCCIÓN A PRUEBAS UNITARIAS






TABLA DE CONTENIDOS

01

TESTING

¿Para qué sirve?

02

TEST UNITARIOS

¿Qué es un test unitario?

03

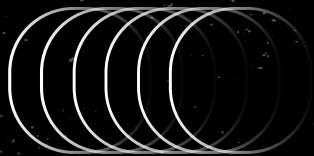
JEST

¿Para qué sirve esta herramienta?

04

EJEMPLO

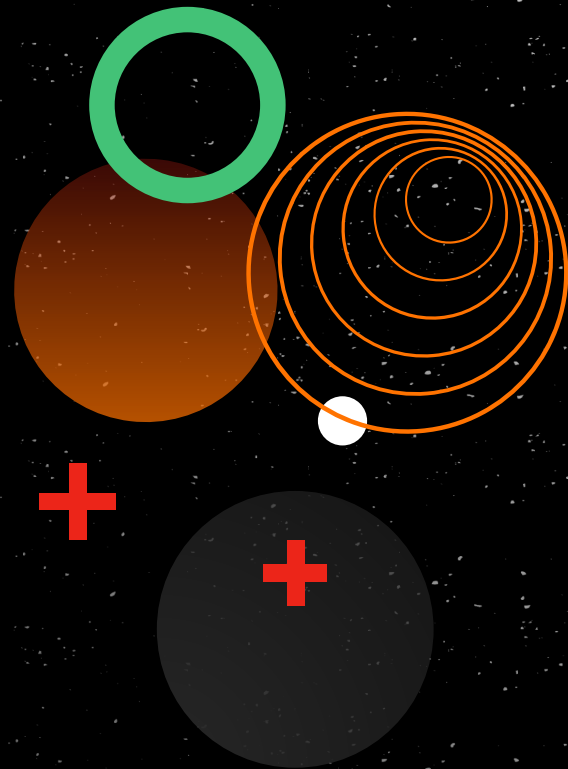




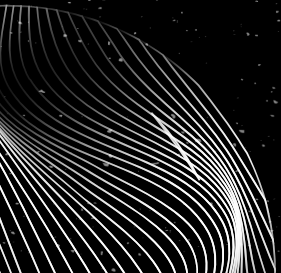
01

TESTING

¿Para qué sirve?



En el proceso de desarrollo de software es normal encontrar errores. El testing de software juega un papel fundamental y supone una garantía de calidad de suma importancia para cualquier empresa.



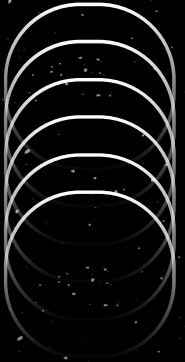
Las pruebas de software son importantes porque permiten identificar de manera temprana si hay algún problema en el software, facilitando su resolución antes de la entrega del producto. Un producto de software debidamente probado garantiza calidad, seguridad, confiabilidad y alto rendimiento, además de otros beneficios como ahorro de tiempo, seguridad y satisfacción del cliente.



TIPOS DE PRUEBAS

Las pruebas de software se clasifican generalmente en dos categorías principales:

- Pruebas funcionales: Verifican cada función de una aplicación o software, su funcionalidad con un conjunto específico de requisitos.
- Pruebas no funcionales o pruebas de rendimiento: Consideran parámetros como la confiabilidad, la usabilidad y el rendimiento. Mantenimiento (regresión y mantenimiento).



TIPOS DE PRUEBAS

Pruebas funcionales

Pruebas unitarias

Pruebas de integración

Pruebas de sistema

Pruebas de sanidad

Pruebas de humo

Pruebas de interfaz

Pruebas de regresión

Pruebas de aceptación

Pruebas no funcionales

Pruebas de rendimiento

Prueba de carga

Pruebas de estrés

Pruebas de volumen

Pruebas de seguridad

Pruebas de compatibilidad

Pruebas de instalación

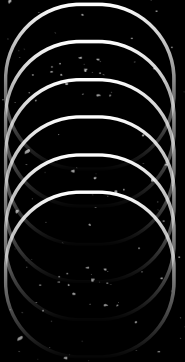
Pruebas de recuperación

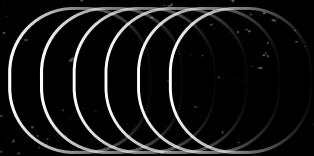
Pruebas de confiabilidad

Pruebas de usabilidad

Pruebas de conformidad

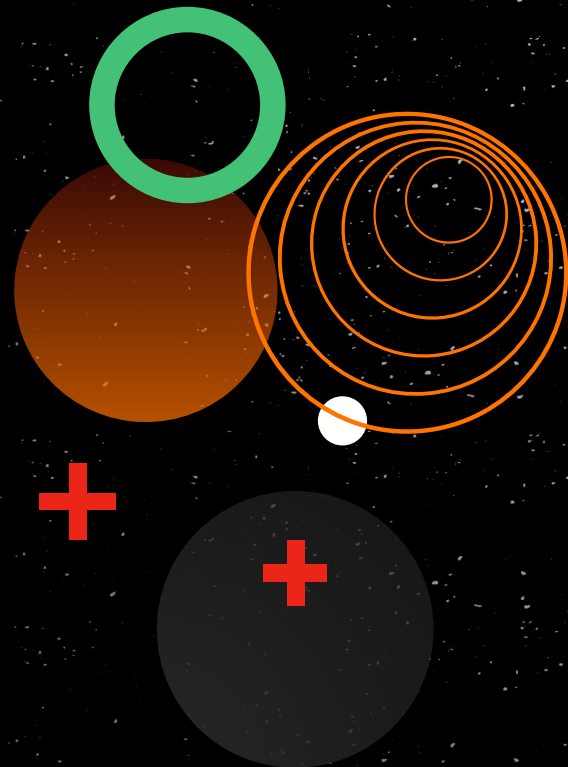
Pruebas de localización





02 TEST UNITARIOS

¿Qué es un test unitario?



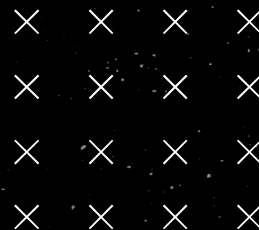
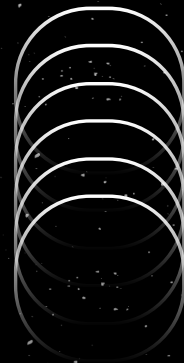


UNIT TESTING

Las pruebas unitarias consisten en aislar una parte del código y comprobar que funciona a la perfección. Son pequeños tests que validan el comportamiento de un objeto y la lógica.

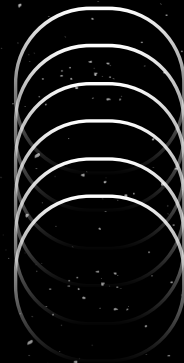
El unit testing suele realizarse durante la fase de desarrollo de aplicaciones de software o móviles. Normalmente las llevan a cabo los desarrolladores, aunque en la práctica, también pueden realizarlas los responsables de QA.

Con ellas se detectan antes errores que, sin las pruebas unitarias, no se podrían detectar hasta fases más avanzadas.



¿POR QUÉ REALIZAR UN TEST UNITARIO?

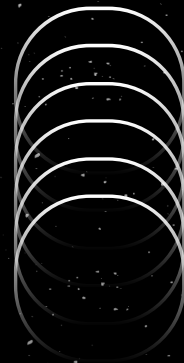
- Las pruebas unitarias demuestran que la lógica del código está en buen estado y que funcionará en todos los casos.
- Aumentan la legibilidad del código y ayudan a los desarrolladores a entender el código base, lo que facilita hacer cambios más rápidamente.
- Las unit testing permiten al desarrollador refactorizar el código más adelante y tener la garantía de que el módulo sigue funcionando correctamente. Para ello se escriben casos de prueba para todas las funciones y métodos, para que cada vez que un cambio provoque un error, sea posible identificarlo y repararlo rápidamente.
- Como las pruebas unitarias dividen el código en pequeños fragmentos, es posible probar distintas partes del proyecto sin tener que esperar a que otras estén completadas.



3A'S DEL UNIT TESTING

Para llevar a cabo buenas pruebas unitarias, deben estar estructuradas siguiendo las tres A's del Unit Testing.

- Arrange (organizar). Es el primer paso de las pruebas unitarias. En esta parte se definen los requisitos que debe cumplir el código.
- Act (actuar). Es el paso intermedio de las pruebas, el momento de ejecutar el test que dará lugar a los resultados que deberás analizar.
- Assert (afirmar). En el último paso, es el momento de comprobar si los resultados obtenidos son los que se esperaban. Si es así, se valida y se sigue adelante. Si no, se corrige el error hasta que desaparezca.



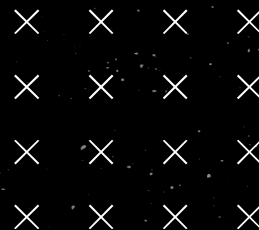
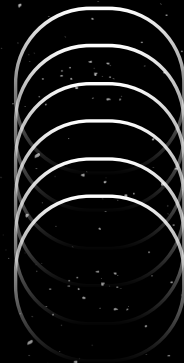
TDD

El test driven development (TDD) o desarrollo guiado por pruebas implica desarrollar las pruebas unitarias a las que se va a someter el software antes de escribirlo. De esta manera, el desarrollo se realiza atendiendo a los requisitos que se han establecido en la prueba que deberá pasar.

Con esta metodología, se escoge un requisito de la lista y se plantea una prueba que se ejecuta para comprobar que falla. Si no falla puede ser porque no se ha planteado correctamente o porque la función ya estaba implementada.

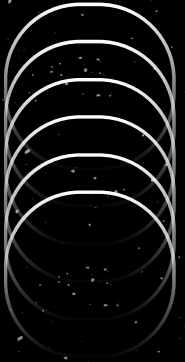
A continuación, se escribe el código que haga posible pasar la prueba de la manera más simple posible, se ejecutan las pruebas y, si todo es correcto, se refactoriza el código para eliminar las partes duplicadas.

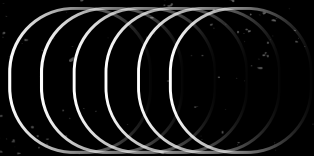
Así se puede tachar ese requisito de la lista y seguir avanzando con el desarrollo.



BUENAS PRÁCTICAS

- Prueba sólo un código a la vez.
- Cualquier cambio necesita pasar el test. En el caso de producirse un cambio en el código de cualquier módulo.
- Corrige los bugs identificados durante las pruebas antes de continuar.
- Acostumbrarse a realizar pruebas regularmente mientras se programa .
Cuanto más código escribas sin testar, más caminos tendrás que revisar para encontrar errores.

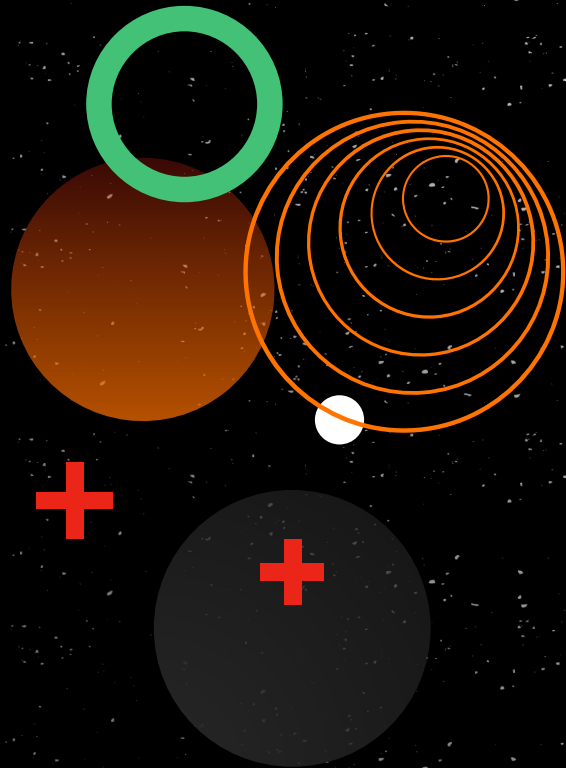




03

JEST

¿Para qué sirve?

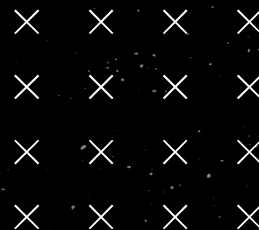
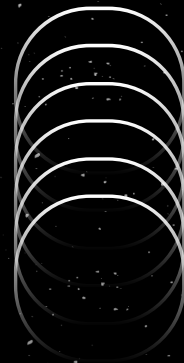


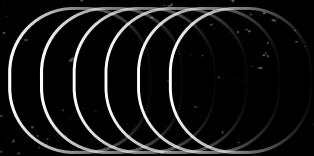


JEST

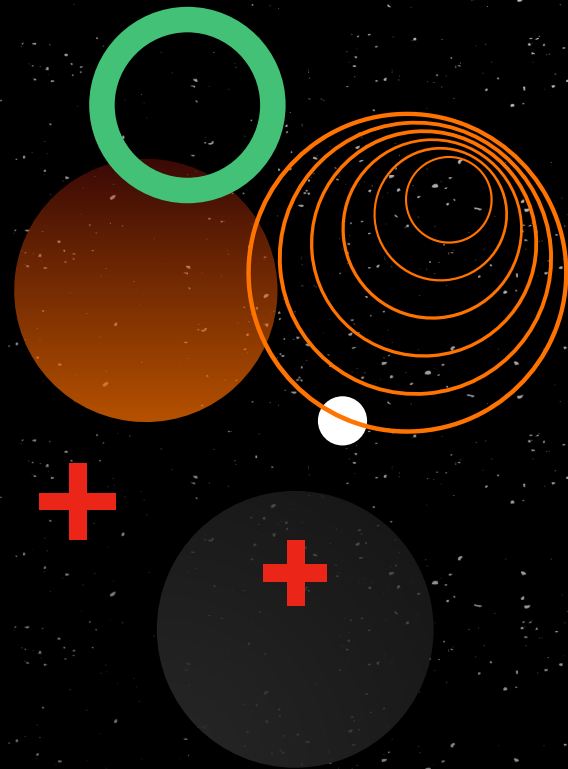
Jest es una librería abierta para pruebas en JavaScript desarrollada por Facebook. Si bien Jest se puede usar para probar cualquier librería de JavaScript, es usada comúnmente para React y React Native.

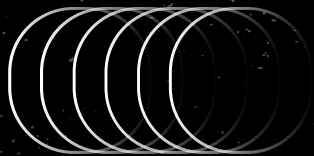
Una librería es uno o varios archivos escritos en un lenguaje de programación determinado, que proporcionan diversas funcionalidades.





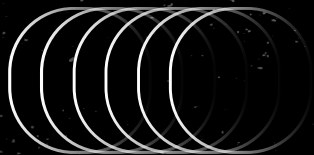
04 EJEMPLO





Supongamos que estás construyendo una función **sum** que suma dos números:

```
function sum(number1, number2){  
    return number1 + number2;  
}
```



El framework de pruebas Jest de Javascript introduce una función especial llamada **expect** para permitirnos realizar las pruebas unitarias, aquí hay un ejemplo de cómo usar **expect** para crear nuestras pruebas unitarias.

```
test('12 and 5 should return 17', () => {  
  let result = sum(12,5);  
  expect(result).toBe(17);  
})
```

Nota: Las pruebas unitarias no se preocupan por el contenido de la función sum, sólo se preocupan por la SALIDA de la función con una ENTRADA determinada.

```
function isUpperCase(sentence){  
  return (sentence == sentence.toUpperCase());  
}
```

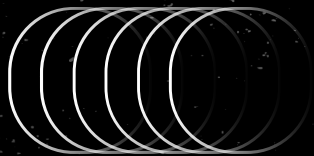
```
test('The string HELLO should return true', () => {  
  const result = isUpperCase('HELLO');  
  expect(result).toBe(true);  
})
```

```
// Segunda prueba posible
test('Testing for Hello (mixed)', () => {
  const result = isUpperCase('Hello');
  expect(result).toBe(false);
})

// Tercera prueba posible
test('Testing for hello (lower)', () => {
  const result = isUpperCase('Hello');
  expect(result).toBe(false);
})

// Cuarta prueba posible
test('Boolean should return false', () => {
  const result = isUpperCase(true);
  expect(result).toBe(false);
})

// Quinta prueba posible
test('Number should return false', () => {
  const result = isUpperCase(12341234);
  expect(result).toBe(false);
})
```



GRACIAS

