Introducción a Testing

DigitalHouse >
Coding School

Índice

1 ¿Por qué testeamos?

Como ya vieron en playground, testing es una metodología que nos permite a los desarrolladores tener certeza de que nuestro código funciona de la manera que esperamos. Pero, ¿por qué lo hacemos? Veamos las principales razones:

Escribir buen código

Nos permite construir aplicaciones más robustas y menos propensas a errores y resultan de utilidad a la hora de refactorizar el código para cumplir con nuevos requerimientos.

Ahorrar tiempo

¿Por qué? Porque cuanto antes encontremos los errores, más rápido y más barato será corregirlos. Existen estudios que indican que cuesta 100 veces más dinero arreglar un error en producción que repararlo en la etapa de diseño y codificación.

Escribir tests tiene grandes beneficios pero también puede presentar algunos inconvenientes, veamos cuales son:

Inconvenientes del testing

- Escribir tests consume tiempo y es relativamente difícil.
- En algunos escenarios, la ejecución de tests implica el consumo dinero.
- Si los testeos no se realizan correctamente, podemos obtener falsos positivos. Nuestros tests pasarán, pero nuestra app no funcionará como se espera. Incluso, podemos obtener falsos negativos: los tests fallan, pero la app funciona correctamente.

2 ¿Qué testeamos?

“Nuestros test deben testear las funcionalidades de nuestra app con las que interactúan las personas. No debemos preocuparnos por testear nombres de funciones, variables ni bibliotecas externas porque esto es trabajo del equipo detrás de la misma.”

3 Principios del Testing

No testear detalles de implementación

Si nuestro test hace algo que las personas usuarias no harán, existe una gran probabilidad de que estemos testeando detalles de implementación.

Testea tus componentes como lo haría la persona usuaria

Porque mientras tus tests se parezcan más a la forma en que se usa tu software, más confiables serán. Esto es el fundamento en el que se basa React Testing Library (librería con la que trabajaremos más adelante).

No te obsesiones con la cobertura del código

La cobertura de código es el porcentaje de código que ha sido testeado o cubierto por tus test. Lo correcto es centrarse en cubrir casos de uso. Este enfoque producirá automáticamente una alta cobertura de código.

Escribe la lógica de negocio en funciones puras fuera de la UI

Si tenemos componentes que necesitan hacer cálculos o validar datos, debemos mover esa lógica fuera del componente a funciones puras para que el testeo sea más sencillo. Incluso hasta puede moverse esta lógica al back-end.

4 Tipos de Tests

Veamos de qué tratan los tres tipos de tests más conocidos: unitarios, integración, end to end (E2E). Durante la cursada trabajaremos con tests unitarios pero es importante saber que hay varias maneras de testear y de qué trata cada una.

Pruebas Unitarias

Están destinadas a verificar si una parte pequeña y aislada del código (unidad) se comporta como se pretendía. Consiste en un fragmento de código que prueba el código de producción de alguna manera, y luego verifica si el resultado coincide con lo esperado. Es decir, una prueba unitaria adecuada es una herramienta fantástica para obtener una retroalimentación súper precisa.

Pruebas Unitarias

Una prueba no es una prueba unitaria si:

- Habla con la base de datos ya que depender de estructuras como la base de datos o el sistema de archivos las hace lentas.
- Se comunica a través de la red. Una prueba no debería empezar a fallar si antes no fallaba por ende no debería depender de dependencias externas.
- Toca el sistema de archivos porque si este tiene algún error podría trasladarse a nuestros tests.
- No puede ejecutarse al mismo tiempo que cualquiera de tus otras pruebas unitarias,
- Tenemos que hacer cosas especiales en nuestro entorno para ejecutarlas.

Pruebas de Integración

A diferencia de las pruebas unitarias, en las que se prueba una pequeña unidad aislada, las pruebas de integración suelen implicar la comprobación de una funcionalidad concreta, normalmente denominada módulo, que tiene dependencias de otra funcionalidad. El objetivo de estas pruebas es comprobar la conectividad y la comunicación entre los distintos componentes de la aplicación. También ayudan a validar las interfaces de la aplicación para que los datos que fluyen de un módulo a otro sean los adecuados.

Pruebas de Integración

Para implementarlas podemos elegir entre 3 enfoques según el tamaño de nuestro proyecto, equipo, presupuesto y otros factores:

1. Enfoque ascendente: los tester se centran en la integración de los módulos más pequeños. La principal ventaja de este enfoque es la posibilidad de fallar rápidamente, ya que los errores en los módulos más pequeños son más fáciles de detectar y más rápidos de solucionar.
2. Enfoque descendente: aquí se da prioridad a los módulos más grandes y complejos, y se simulan los de menor nivel.
3. Enfoque Big Bang: con este enfoque, todos los módulos se prueban juntos al mismo tiempo. La detección de errores puede ser un poco más difícil en comparación con los otros enfoques, pero en una aplicación pequeña puede ser una gran opción.

Pruebas End to End (E2E)

Cuando se hacen pruebas E2E, se busca probar el producto de la misma manera que lo experimenta una persona usuaria real. Su objetivo es simular la experiencia paso a paso de una persona usuaria y permitir validar diferentes subsistemas y capas de la aplicación. Hay dos maneras de llevarlas a cabo:

1. Pruebas Horizontales: replican los casos de uso típico de las aplicaciones y los recorren de principio a fin. Por ejemplo el proceso de registro de un nuevo usuario. Es la modalidad más común.
2. Pruebas Verticales: tienen como objetivo probar los componentes del sistema siguiendo los datos a través de las capas de la interfaz de usuario, la API y la base de datos. Por ejemplo el proceso de compra en un ecommerce.

Pruebas End to End (E2E)

Los beneficios de este tipo de tests van desde asegurar que todo funciona como se espera, yendo desde la interfaz de usuario hasta la capa de la base de datos, aumentar la cobertura general de las pruebas y ayudar a detectar errores antes de que lo hagan las personas usuarias finales.

Son grandes beneficios pero sepamos también que las pruebas E2E son, claramente, las más costosas y complejas de realizar y que, además, llevan más tiempo.


DigitalHouse>
Coding School