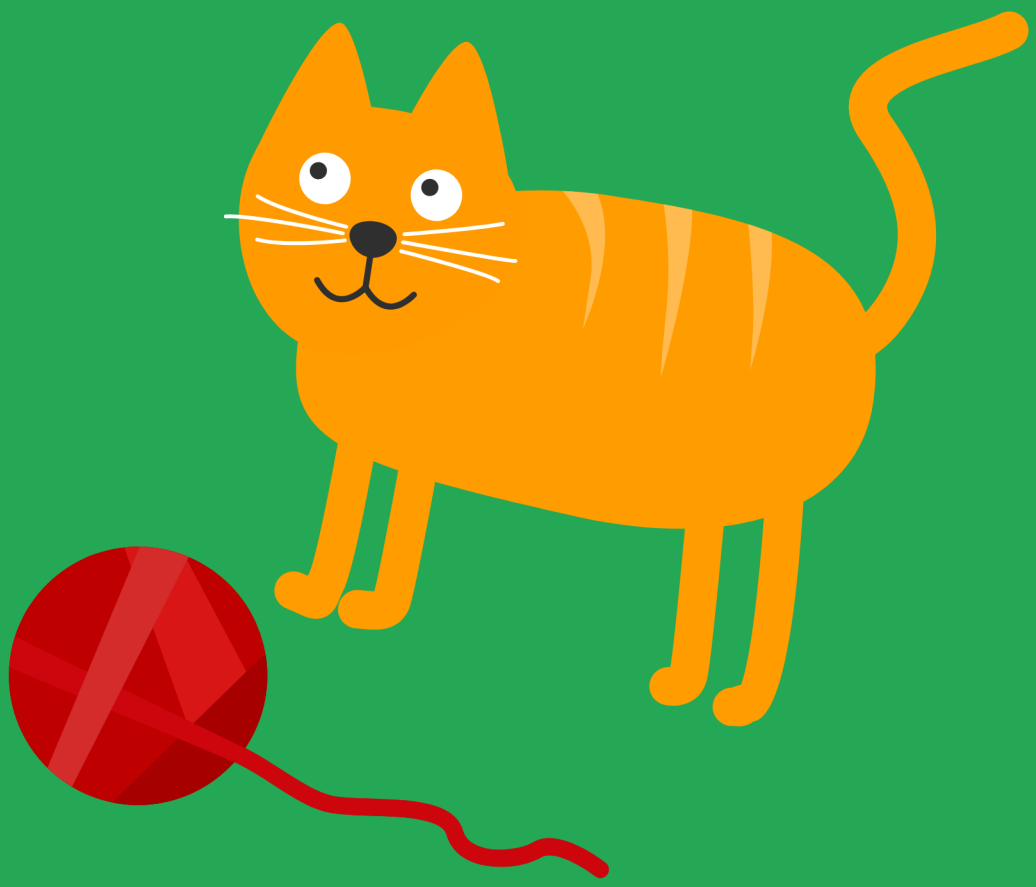
Frontend Testing



Una introducción al mundo del Testing orientado a Front-End

Iago Lastra Rodríguez

Frontend Testing

Iago Lastra Rodríguez

Este libro está a la venta en http://leanpub.com/frontend-testing

Esta versión se publicó en 2020-04-25

Índice general

Introducción

El típico flujo de trabajo

Para empezar a hablar del mundo de los tests me gustaría contar la historia de Bob. Bob no existe, pero en mi experiencia representa perfectamente la forma de trabajar de muchos de los programadores que he conocido.

Bob trabaja como ingeniero de software en una de las tienda online más importantes de Europa y le acaban de encargar escribir el módulo que se encarga de calcular los impuestos que se aplican a las compras en la nueva versión de la web.

Bob tiene dos pestañas abiertas en su navegador, una con la descripción de la issue, donde extrae requisitos a partir de los comentarios y otra con localhost:3000[1] donde tiene ejecutandose una versión de desarrollo de la app.

Bob empieza por escribir un componente para mostrar los impuestos que se aplican al carrito de la compra, en menos de media hora escribe el código lo guarda y abre la pestaña de su navegador, inicia sesión con un usuario de prueba, añade un producto a la lista y comprueba que su recien estrenado componente está presente y muestra el IVA correctamente.

Bob repite este proceso con un usuario español y con un usuario alemán y se da cuenta que le falta calcular los impuestos alemanes y vuelve a su editor de código para arreglarlo.

Bob añade una condición a su código

```
if(user.countryCode == "GER") {
// ...
}
```

donde incluye todo lo necesario para calcular los impuestos en clientes alemanes. Al terminar vuelve a abrir el navegador simulando una nueva compra, comprueba que se muestra el precio correcto y sube su código al servidor.

¿Te ves reflejado en Bob? ¿ Se parece tu flujo de trabajo al suyo?

Este proceso de escribir código y abrir el navegador, abrir una consola, mandar una petición HTTP y examinar el resultado o incluso poner `console.log` por todas partes es en mi experiencia el flujo de trabajo más extendido: Programar algo y comprobar manualmente el resultado.

¿Se puede mejorar este proceso? ¿Qué pasa con todo el conocimiento si Bob se va de la empresa? ¿Cómo va a estar seguro el siguiente desarrollador de que no ha roto nada al hacer cambios?

¿Qué es un test?

Podríamos definir un test como:

> Un **proceso** que permite obtener **información** acerca de un sistema.

Según esa definición ese proceso en el que Bob abre su navegador, elige un producto y comprueba el valor del IVA es un test! Todo el mundo hace tests sin darse cuenta, cambiar una función y poner un breakpoint para ver que los parámetros se procesan en el orden correcto es un test. Usar postman para hacer una petición es un test, ejecutar el código poniendo un console.log() es un test…

El principal objetivo de todo este texto es entender que estos tests manuales que la mayoría de la gente hace pueden ser automatizados! Es posible llegar a un punto en el que no hace falta abrir el navegador, un punto en el que podemos dejar una documetación formal que indique a los nuevos desarrolladores qué es lo que debe de hacer el sistema, un punto en el que ni siquiera hace falta lanzar la aplicación porque tenemos mejores herramientas para verificar que nuestro trabajo esta bien hecho: **Los tests automáticos**.

Tests automáticos vs no automáticos

Para sacar el mayor provecho posible a los tests nos interesa que sean automáticos, fiables y rápidos de forma que se integren perfectamente en nuestro proceso de desarrollo y puedan ser ejecutados en cualquier momento.

La tienda online sin tests

Volviendo al ejemplo de la tienda online, imaginemos que Bob acaba de programar la issue y ejecuta los siguientes tests manuales:

- Abrir web
- Elegir un producto cualquiera, una tostadora.
- Comprobar al precio final es de 40€ con un IVA de un 21%

Meses más tarde a una compañera de Bob, Alice, le encargan programar un nuevo sistema de tickets regalo con descuentos para el Black Friday.

Debido a una terrible planificación Alice tiene poco menos de una tarde para terminar su trabajo así que se pone a programar inmediatamente: abre su editor, escribe unas cuantas lineas y varias horas más tarde abre localhost:3000[2] y sigue los siguientes pasos:

- Abrir la tienda online
- Elegir un producto cualquiera, una maquinilla de afeitar.
- Añade un código de descuento de 5€
- Comprueba que el precio final es de 15€ con un descuento de 5€ .

"Perfecto", Alice sube el código a producción a las 11:10 de la noche, le han sobrado 50 minutos, respira aliviada.

A la mañana siguiente todos los programadores leen el siguiente email:

> Para: developers@empresa.com
>
> Buenos dias,
>
> Escribo para informaros de que durante esta noche hemos perdido unos 2.000.000€ en pedidos debido a un BUG que hemos introducido en la actualización de ayer.
>
> Por algún motivo se estaba calculando mal el precio final para los clientes alemanes con un ticket regalo. Hemos quitado el cambio y vamos a investigar (...)

Alice estaba aterrorizada, aunque el código se lo reviso Carol a nadie se le ocurrió que el módulo nuevo de impuestos pudiese interaccionar de esta manera con el código de los tickets regalo. Esa parte la escribió Bob y ese día no estaba en la oficina.

La tienda online con tests

Ahora imaginemos la misma situación en una empresa en la que todos los programadores escriben tests automáticos por cada funcionalidad que desarrollan:

Bob escribe su test automático, escribe el código de la funcionalidad y cuando todos los checks estan en verde y se va a su casa.

Por desgracia los tests no evitan una mala gestión por lo que el marrón de escribir la funcionalidad de los descuentos del Black Friday a contrareloj aparece de nuevo.

Esta vez el Alice ejecuta los tests de la aplicación antes de subir sus cambios y en menos de 2 segundos ve por su pantalla:

> Resultados de los tests: 5432 Tests OK 1 Test FAIL
> [X] Cálculo de precios para Alemania: Se esperaba un precio de 40€ pero se tiene 15€

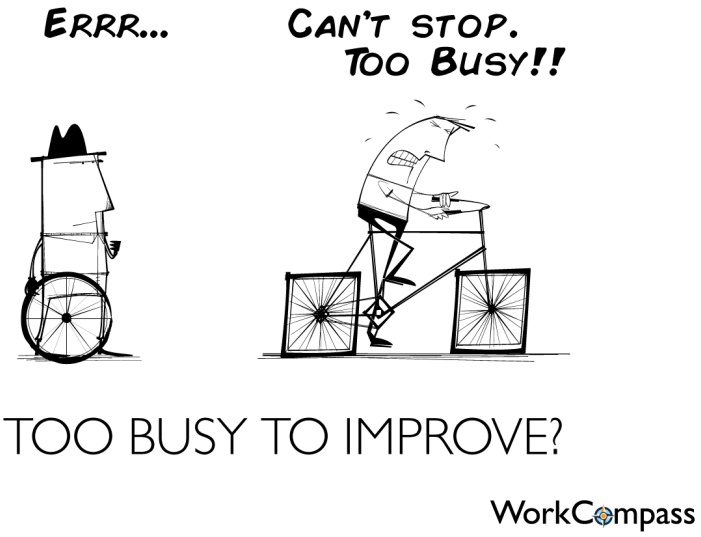
Trás revisar el código se da cuenta que efectivamente ha añadido una condición erronea que hace que no se calcule el IVA correctamente en algunos casos. Lo soluciona en 2 minutos y vuelve a ejecutar los tests:

> Resultados de los tests: 5433 Tests OK 0 Test FAIL

Sube los cambios y se va para casa. El único email destacable que recibió al día siguiente era uno que informaba del nuevo record de ventas.

La importancia de los tests

Últimamente es bastante raro encontrarse desarrolladores que abiertamente digan que los tests no son útiles sin embargo todo el mundo tiene su excusa para no hacer tests.



"No tengo tiempo"

La excusa por excelencia... ¿Te imaginas a un piloto diciendo a la tripulación que cómo van con retraso se van a saltar las comprobaciones de despegue?

Imagínate que empiezas a trabajar como repartidor de periódicos, es un pequeño proyecto y lo puedes hacer mientras das un paseo por las mañanas. Con el tiempo el negocio crece y alguien te pregunta ¿por qué no repartes en bicicleta? y contestas *"No tengo tiempo! Mi jefe no quiere saber nada de bicicletas."*

Es cierto que en un día concreto es mucho más rápido repartir el periódico andando que aprender a montar en bicicleta y repartirlos, pero una vez sepas andar en bicicleta no hay vuelta atrás serás más rápido y tendrás una ventaja sobre los repartidores que no saben andar en bicicleta.

Con los tests sucede lo mismo, no es que no tengas tiempo para hacer tests, lo que probablemente no tengas es tiempo para aprender a hacer tests y ponerlos en práctica en tu proyecto!

"Muchos bugs no se podrían detectar incluso haciendo tests"

Tampoco los cinturones de seguridad evitan la muerte en el 100% de los accidentes pero nadie discute si es mejor llevarlo puesto ¿verdad? Al igual que un cinturón los tests no te garantizan que no se produzcan errores pero al menos aportan un extra para evitar que se produzcan.

“Mi cliente no quiere tests”

Puedo entender que el cliente no quiera asumir el coste de un equipo aprendiendo a escribir tests pero una vez formado la productividad aumenta considerablemente y ningún cliente se va a oponer a un equipo más productivo.

“Mi código cambia constantemente, no tiene sentido hacer tests”

Aunque implicitamente me estas diciendo que los tests dan más trabajo, quiero destacar dos cosas:

1. Si tus tests se rompen con cualquier mínimo cambio, quizá es que están muy acoplados al código y deberías revisarlos.
2. Si los tests estan bien lo que cambia son los requisitos y es un problema de producto, no de ingeniería.

“Mi código no cambian nunca, no tiene sentido hacer tests”

¿Puede ser que no cambie porque no tiene tests y nadie se atreve a tocarlo? En mi experiencia la única constante en el mundo del software es el cambio, la propia palabra software lo indica.

“Mi código es muy difícil de probar”

Si estás en esta situación no te preocupes, hay libros dedicados integramente a cómo trabajar con código legacy que explican cómo hacer tests poco a poco.

En mi experiencia en cuanto un programador aprende a hacer tests bien no hay vuelta atrás: Los tests ahorran tiempo y dinero.

Cuando una base de código esta bien testeada se trabaja más rápido y con más confianza, el número de bugs se reduce drásticamente y los refactors de código son más frecuentes ayudando a tener un código más limpio. Hay estudios[3] llevados a cabo por Microsoft e IBM que indican que cuando se aplica TDD se reduce el número de bugs entre un 40 y un 90% y compañías de referencia como Google o Facebook incluyen el testing en sus entrevistas de trabajo.

Si con todo esto aún no estas convencido de darle una oportunidad al testing te pido que leas la siguiente sección y si sigues sin estar seguro me gustaría saber tus motivos para intentar convencerte el futuro.

Los tres beneficios principales de los tests

Podemos decir que los tests tienen 3 funciones principales: Documentación, Diseño y Verificación.

Documentación.

Uno de los problemas más comunes en los equipos de desarrollo es que no existe documentación formal del código. Es habitual que unas pocas personas tengan en su cabeza todo el conocimiento de una aplicación convirtiéndolas en un cuello de botella o en los conocidos como *silos de conocimiento*.

Esta situación es tan común que incluso se ha definido el término "bus factor"[4] para hacer referencia a cuantos desarrolladores tendrían que ser atropellados por un autobús para que el proyecto se vuelva inmantenible.

Para evitar los silos de conocimiento en muchas empresas intentan escribir documentos que indican qué hace el código y cómo lo hace y suelen parecerse a esto:

> El archivo Taxes.js tiene utilidades para hacer calcular
> los impuestos que se aplican a una compra.
>
> Por defecto se usa el IVA general de España en el año 2010 (18%).
>
> Se ha migrado todo a Taxes.new.js.
>
> Los impuestos de Alemania siguen usando Taxes.js cuando el
> producto pertenece a la categoría "Electrónica"...

El problema es que actualizar estos documentos con cada cámbio da trabajo y con el tiempo la mayoría de la documentación acabará desactualizada.

Veremos que unos tests bien escritos se utilizan como "documentación viva". Documentación porque ayudan a entender qué debe de hacer el código y viva porque en cuando cambia el código los tests tienen que cambiar obligatoriamente con cada los nuevo requisto.

Diseño.

Aunque poca gente se da cuenta de esta ventaja los tests ayudan mucho a mejorar la calidad del código.

Para que un código sea fácil de probar tiene que ser una pieza aislada de funcionalidad bien definida, generalmente si un código es fácil de probar es que cumple los principios SOLID[5].

El hecho de escribir el test obliga a reflexionar sobre la funcionalidad, sobre como interactuarán los compenentes, sobre las APIS y parámetros que se utilizarán.

Todo este proceso hace que el resultado final sea más maduro y completo que si directamente se programa la funcionalidad. Metodologías como TDD previenen la aparición de código innecesario porque solamente se escribe el código necesario para pasar el tests.

Por último los tests facilitan los cambios y los refactors. En un entorno sin test es raro que se mejoren estos trozos porque se aplica la ley de *"si funciona no lo toques"* siendo frecuentes los parches por encima que hacen que el código se degrade progresivamente. Por contra en un entorno con tests son habituales los refactors donde el código se limpia cada poco tiempo.

Verificación.

Esta es la ventaja que historicamente más se asocia a los tests: Permiten verificar que se cumplen una serie de requisitos de forma rápida y automática.

Tener un código con una batería de pruebas sólida reduce notablemente el coste de mantenimiento y de extensibilidad del sistema por lo el tiempo necesario para escribir los tests se amortiza rápidamente.

Durante las primeras etapas de desarrollo pueden no verse claras las ventajas del testing pero a medida que el equipo y el proyecto crece se hacen evidentes. Incluso cuando solamente hay un programador es complicado tener en la cabeza el funcionamiento de todas las características del sistema y por lo tanto es muy difícil asegurar que los cambios que va a introducir no rompen nada de lo hecho anteriormente.

Con el tiempo siempre se acaba en una de estas situaciones:

1. El desarrollador no prueba todo el código y tarde o temprano aparecerán bugs que hacen que se pierda tiempo y dinero.
2. El desarrollador prueba todo el código a mano perdiendo tiempo y dinero.
3. El desarrollador prueba el código utilizando tests automáticos.

¿Cuál parece la mejor?

¿Debería escribir el test primero?

Mi recomendación es que si, no conozco a ningún programador que empezase a hacer TDD correctamente y se arrepienta, una vez te acostumbras no hay vuelta atrás. Dicho esto, lo realmente importante es que el código tenga tests, da igual cómo!

Entre las ventajas de escribir el test primero detacamos:

Muchas veces da pereza escribir el test después

Mucha gente que escribe el test después se da cuenta que el código que ha escrito no es fácil de probar y lo deja para luego y todos sabemos lo que siginifica “dejar algo para luego” en software.

Obliga a la gente a entender y definir claramente el problema.

Si te esfuerzas en escribir los tests primero tendrás que razonar sobre la API que va a tener el código, tendrás que ver que parametros reciben y que valores devuelve en cada caso y sin querer realizarás un análisis mayor del problema en sí.

Por desgracia es bastante común encontrarse issues mal definidas, donde los casos de uso no estan claros, o no se sabe el alcance de una tarea. Si escribes el test primero tendrás una definicion formal del criterio de aceptación de la issue que ayudará a negocio y desarrollo a analizar en profundidad los posibles casos que puedan darse.

Permite escribir menos código

Si el test está bien definido, solamente hace falta escribir el código necesario para pasar el test.

Escribir el test primero permite retomar el trabajo rápidamente

Si por lo que sea te interrumpen mientras estas programando y has escrito un test, hace que puedas retomar el trabajo en el punto en donde lo dejaste previamente.

Historias reales: “Deberías hacer tests”

Odín del Río

Primero de todo, preséntate: ¿Quién eres? ¿A qué te dedicas?

Pues, mi nombre es Odín del Río y llevo programando unos 10 años. Ahora mismo tengo el rol de Tech Lead en N26.

¿Puedes contarme un caso en el que la falta de tests supusiese un desastre/una pérdida de tiempo enorme para la empresa?

Casi todos los grandes incidentes que he vivido han sido por falta de tests en algún sitio. De los más típicos y castrastróficos que he visto son los relacionados con breaking changes en APIs. La falta de tests de contrato puede reventar todos los clientes de un servicio sin que sea obvio en los logs del servicio tras el despliegue.

Sobre pérdidas de tiempo, aquí podemos hablar de todo ese código sin tests que las empresas tienen en producción. Hay quién define legacy como código sin tests, y ¿cuántas empresas conocemos con un producto que da dinero pero tiene un gran legacy? En mi experiencia laboral, todas.

La inversión de tiempo y dinero para simplemente mantener con vida el código que ya está en producción es brutal. Además, ese legacy será un foco de frustración tanto para producto (que no entenderá como esa feature tan sencilla lleva tanto tiempo) como para el equipo de desarrollo (que tendrá miedo de realizar cualquier cambio).

Esa frustración causará que la gente deje la empresa, haciendo a esta invertir un extra de tiempo en reclutamiento y on-boarding… Todo eso consumirá mucho tiempo y dinero.

¿Puedes contarme un caso en el que los tests te salvasen la vida?

Pues constantemente! Cuando estoy implementando algo, recurrentemente tengo ese momento WTF en el que un test falla y no sabes porqué… Luego te das cuenta de que tenías un error en tu código. Por un momento te imaginas el desastre del que ese test te ha salvado y continúas feliz con tu vida

¿Qué le dirías a alguien que está leyendo un libro sobre cómo hacer tests en frontend?

Que no se desanime, al principio da pereza, es cierto. Pero una vez se aplica, la tranquilidad que supone el desplegar código testeado no tiene precio. Si alguien está leyendo un libro sobre testing es que ya está en el buen camino.

Para acabar de convencer a la gente sobre hacer testing yo siempre digo que es cuestión de respeto a tus compañeros. Si alguien dice que le da pereza escribir un test es que no es un buen compañero de equipo porque alguien en el futuro tendrá que modificar ese código y si ese código no tiene tests el extra de esfuerzo que se tendrá que hacer es mucho más grande que el de haber escrito el test en primer lugar.

¿Tienes algún truco o consejo a la hora de escribir tests que te gustaría compartir?

Pues suelo practicar TDD en casi todos los escenarios, tanto para unit test como para test de más alto nivel.

Para los unit test suelo empezar por el final, escribiendo primero la aserción que quiero hacer, y de ahí sigo “hacia arriba”.

Para los tests de más alto nivel, aquí la pereza es más fácil que gane, aún así mi experiencia me ha demostrado varias veces que esa inversión de tiempo en el setup de escenarios de tests complejos merecen la pena, aquí hablo de esos tests de BE donde necesitas un servidor, una bbdd, un sistema de mensajería… Todo ese setup de docker, y del estado inicial para tu test puede ser tedioso, pero siempre vale la pena.

Y por último, los nombres de los tests están hechos para que otras personas los entiendan cuando fallen, así que dadle un extra de cariño!

Victor Ribero

Primero de todo, preséntate: ¿Quién eres?¿A qué te dedicas?

Muy buenas! Pues mi nombre es Victor Ribero (@devictoribero[6]) y soy desarrollador frontend al que le encanta compartir conocimiento y conectar con la gente.Como podrás deducir, trabajo como desarrollador frontend y en mi tiempo libre desarollo chooseyourplant.com[7] que es una comunidad internacional para los amantes de las plantas de interior.

¿Puedes contarme un caso en el que la falta de tests supusiese un desastre/una pérdida de tiempo enorme para la empresa?

Excepto en Schisbted (que tenemos test de varios tipos integrados en distintas pipelines), en las demás empresas en las que he trabajado no había test y aunque no lo calculaé en su debido momento, supongo que la pérdida de tiempo era muy muy grande. ¿Cuanto no hemos desarrollado una funcionalidad alguna vez que no estaba testeada de manera automática y hemos perdido tranquilamente 30 minutos o 1hora para testearla? ¿Y nadie se pregunta cuantas horas al año pierde uno testeando manualmente?

¿Puedes contarme un caso en el que los tests te salvasen la vida?

Muchas son las veces que repasando el código visualmente lo veo perfecto y que incluso hago algo de testing manual y también está bien, pero que cuando ejecuto los test automatizados, algo revienta y por suerte, eso estaba ahí para avisarme.

Más de una vez hubiera subido un código a producción que hubiera reventado la home page fácilmente al no gestionar bien los errores.

¿Qué le dirías a alguien que está leyendo un libro sobre cómo hacer tests en frontend?

Cada vez tengo una opinion más ferrea de que los test han de ir orientados al usuario y sobretodo en frontend, pues nuestro código dependerá de como interactúa esta persona con nuestra aplicación.

¿Tienes algún truco o consejo a la hora de escribir tests que te gustaría compartir?

A mi me gusta testear la accesibilidad del código. Si por ejemplo tengo que seleccionar una imagen, lo haré por su atributo `alt` y si es una imagen que no aporta semantica por `alt=''` o `role='presentation'`. Hay una librería muy chula que se llama Axe (la descubrí hace poco) que ayuda a testear la accesibilidad. **Si añades accesibilidad a tu web, no estás añadiendo una funcionalidad, estas cumpliendo con tu obligación**.

Aunque no uso TDD, cuando testeo la lógica de negocio, me gusta escribir el test sin tener en la cabeza como es la implementación o la API existente. Eso, me ayuda muchas veces a darme cuenta de que a lo mejor ciertos métodos no son suficientemente claros, o que realmente no se comportan como deberían (hacen cosas de más). **Hagamos una web inclusiva para todos**, esa es mi idea.

Sergio de Candelario

Primero de todo, preséntate

Me llamo Sergio, llevo unos 7-8 años dedicados al mundo de la programación y los últimos 5-6 enfocado en la programación web (principalmente PHP y JS), pero me tira más el backend que el frontend.

¿Puedes contarme un caso en el que la falta de tests supusiese un desastre/una pérdida de tiempo enorme para la empresa?

En mi anterior empresa el no tener test complicaba los despliegues a producción, que además eran cada dos semanas, se tenía que probar todo manualmente sin tener en cuenta todas los posibles casos, eso hacía que muchas veces en vez de hacer el despliegue en lunes acabase siendo martes o miércoles y eso no qutiaba que luego hubiese que hacer algún fix de algún error

¿Puedes contarme un caso en el que los tests te salvasen la vida?

Donde estoy ahora tenemos tanto unitarios como funcionales y sobretodo veo muy útil tener tests sobretodo a la hora de hacer refactors, ya no tanto los unitarios sino los funcionales, que no deberían tocarse a pesar de hacer refactor de la lógica interna. Si estás cambiando lógica interna y el test funcional te falla es que te estás dejando algo y eso ya es señal de alarma de que has de revisar bien el flujo.

¿Qué le dirías a alguien que está leyendo un libro sobre cómo hacer tests en frontend?

Tests de frontend no he hecho nada, la verdad. Pero mi consejo sería que ponga los test en primer lugar, que si puede trate de entender y enfocar TDD u otra metodología similar. Que los tests forman parte del desarrollo y que nunca se debería subir código a producción sin su suite de test correspondiente.

¿Tienes algún truco o consejo a la hora de escribir tests que te gustaría compartir?

Un truco que suelo seguir es hacer los tests según va sucediendo el flujo, prefiero ir de menos a mas, de esta forma me aseguro de que compruebo todos los casos. Otra cosa que me va muy bien es la cobertura, sobretodo para saber que el tests que estoy haciendo hace el flujo que espero, ya no tanto para saber la cobertura del código que no es un dato fiable. Por otra parte a veces trato de aplicar TDD, pero reconozco que es algo que me cuesta adoptar por mi mismo

Gonzalo Serrano

Primero de todo, preséntate: ¿Quién eres? ¿A qué te dedicas?
Hola, me llamo Gonzalo Serrano[8] y soy un mallorquín afincado desde hace años en Barcelona e ingeniero informático por la FIB. He desarrollado mi carrera profesional en varias startups de Barcelona, como Trovit (clasificados), Social Point (juegos sociales), Lernin Games (juegos edicativos) y ahora trabajo en Paack (logística) como ingeniero de software. Además soy co-foundador de la comunidad online BcnEng[9] con más de 3000 inscritos y co-organizador de la meetup GolangBCN meetup[10] Aunque me he especializado en arquitectura de backends escalables con Go en cloud, también he trabajado en frontend web y en temas de infraestructura y automatización. Tengo pendiente aprender más a fondo tecnologías mobile.

¿Puedes contarme un caso en el que la falta de tests supusiese un desastre/una pérdida de tiempo enorme para la empresa?

En una de las empresas donde estuve un bug sutil en la gestión de la persistencia del estado de los usuarios en una funcionalidad provocó pérdidas de decenas de miles de euros hasta que fue encontrado, y se tardó una semana en restaurar parte de los datos. Había testing pero no suficientemente bueno.

¿Puedes contarme un caso en el que los tests te salvasen la vida?

Recuerdo estar programando con un colega una funcionalidad bastante compleja en uno de los productos más importantes de una de las empresas. No hicimos tests (por falta de experiencia) y tardábamos muchísimo en avanzar porque había que recrear a mano los estados para probar todas las casuísticas, y no llegábamos al deadline y eso suponía dejar de ganar mucho dinero. Por suerte el backend lead era experto en testing y nos enseñó que cualquier código puede ser testeable y en un par de horas refactorizamos el código e hicimos tests básicos que nos ayudaron a iterar mucho más rápido y sacar la funcionalidad a tiempo.

¿Qué le dirías a alguien que está leyendo un libro sobre cómo hacer tests en frontend?
Yo no soy un experto, pero me han recomendado echadle un ojo a snapshot testing. Básicamente se trata se serializar el markup de la UI y comprobar diferencias. Tiene sus pros y sus contras pero me pareció interesante.

¿Tienes algún truco o consejo a la hora de escribir tests que te gustaría compartir?

"Learn from the masters", busca y aprende cómo hacen testing los referentes de la indústria y en sus repositorios. Lee sobre la pirámide del testing y los principios SOLID. Dale una oportunidad a TDD como herramienta para diseñar mejor tu código y declarar la especificación de las funcionalidades a implementar. ¡Además es divertido!

Entonces ¿Qué es un test?

Ya hemos visto la definición formal pero ¿Qué forma tiene un test? ¿Cómo lo escribo? En javascript existen diferentes librerías que permiten escribir tests con facilidad siendo Jest[11] una de las más populares, para instalar Jest basta con escribir:

```
npm install -D jest
```

Es habitual añadir jest como comando de test por defecto en el `package.json` de forma que al ejecutar `npm run test` Jest buscará automáticamente cuyo nombre termina en `.test.js` y ejecutará los tests que se definan dentro.

Vamos a crear un test de ejemplo llamado `hello.test.js` donde se comprueba que la función replace[12] de javascript funciona correctamente:

```
// Description
test('this is the test description', () => {
  // Arrange
  const template = 'Hello <name>!';
  // Act
  const actual = template.replace('<name>', 'world');
  // Assert
  expect(actual).toEqual('Hello world!');
});
```

En este ejemplo podemos diferenciar varias partes que estarán presentes en cualquier test:

- **Nombre o Descripción**: Indica qué código estamos y qué resultado se espera.
- **Arrange**: Una primera parte en la que preparamos el test.
- **Act**: Una segunda parte en la que se ejecuta el código que se quiere probar.
- **Assert**: Una parte final en la que se compara el resultado obtenido con el resultado esperado.

Finalmente podemos ejecutar el test

```
npm run test
```

Y tendremos el siguiente output por consola, la descripción del test no es la mejor del mundo pero ya llegaremos a eso más adelante, lo importante es que hemos escrito nuestro primer test.

```
✓ this is the test description (13ms)
```

Características de un test

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Sensibilidad: Sensible vs Insensible

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Especificidad: Específico vs Inespecífico

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Precisión: Preciso vs Impreciso

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fragilidad: Frágil vs Sólido

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Estabilidad: Estable vs Inestable

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Flexibilidad: Flexible vs Inflexible

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Mantenibilidad: Mantenible vs Inmantenible

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Velocidad: Velóz vs Lento

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Profundidad: Profundo vs Superficial

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Test superficial (shallow)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Test profundo (deep)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Tipos de Tests

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Small | Unidad

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Medium | Integración

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Medium | Funcional

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Large | End-to-End | System

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Cómo distribuir los tests

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Pirámide del testing.

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Trofeo del testing.

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

El barco del testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Clean Testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Ejemplo I: Los lados de un triángulo

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Jest

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fase 0: Razonamiento

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fase 1: Nombre y descripción

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fase 2: Arrange

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fase 3: Act

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Paso 4: Assert

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Paso 5: Ejecutar los tests

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Resumen

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Ejemplo II: La aplicación de venta de entradas.

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fase 1: Tests básicos

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fase 2: Añadiendo contexto

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fase 3: Añadiendo más contexto.

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fase 4: Compartiendo contexto

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fase 5: Simulando dependencias.

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Pruebas de integración

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Pruebas unitarias

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Conclusiones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Otros tipos de testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Component Testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Paso 0: Definir los tests

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Paso 1: Renderizar el componente

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Paso 2: Comprobar el precio mostrado por defecto

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Paso 3: Hacer click en el checkbox

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Paso 4: Comprobar el precio de un usuario VIP

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

E2E Testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Cypress.io

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Instalar Cypress

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Configurar Cypress

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Escribir un test

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Ejecutar Cypress

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Selenium

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Visual regression testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

BackstopJS

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

API Testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Postman

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Contract Testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Pact

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Snapshot testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Property based Testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

JSVerify

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Exploratory testing

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Testing doubles

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Dummie

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Stubs

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Spies

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Fake

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Mock

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Buenas prácticas

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Nombres y Descripciones

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

SWA

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

¿Qué unidad se prueba?

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

¿Cuál es el estado inicial y cuál es el resultado esperado?

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Contexto

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Partes de un test (AAA)

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Arrange

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Act

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Assert

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Un test solamente debería fallar por una razón

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

No incluir lógica en el test

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

No probar la implementación

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

No probar métodos privados.

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Probar comportamiento en lugar de estado

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

Exportar Servicios

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

No confiar ciegamente en la cobertura

Este contenido no está disponible en el libro de muestra. El libro se puede comprar en Leanpub en http://leanpub.com/frontend-testing.

*

Notas

1 localhost:3000

2 localhost:3000

3 https://link.springer.com/article/10.1007/s10664-008-9062-z

4 https://es.wikipedia.org/wiki/Bus_factor

5 https://es.wikipedia.org/wiki/SOLID

6 https://bcneng.slack.com/team/UDB4WEC03

7 http://chooseyourplant.com/

8 https://gon.cat

9 https://bcneng.org

10 https://www.meetup.com/Golang-Barcelona

11 https://jestjs.io/

12 developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String/replace