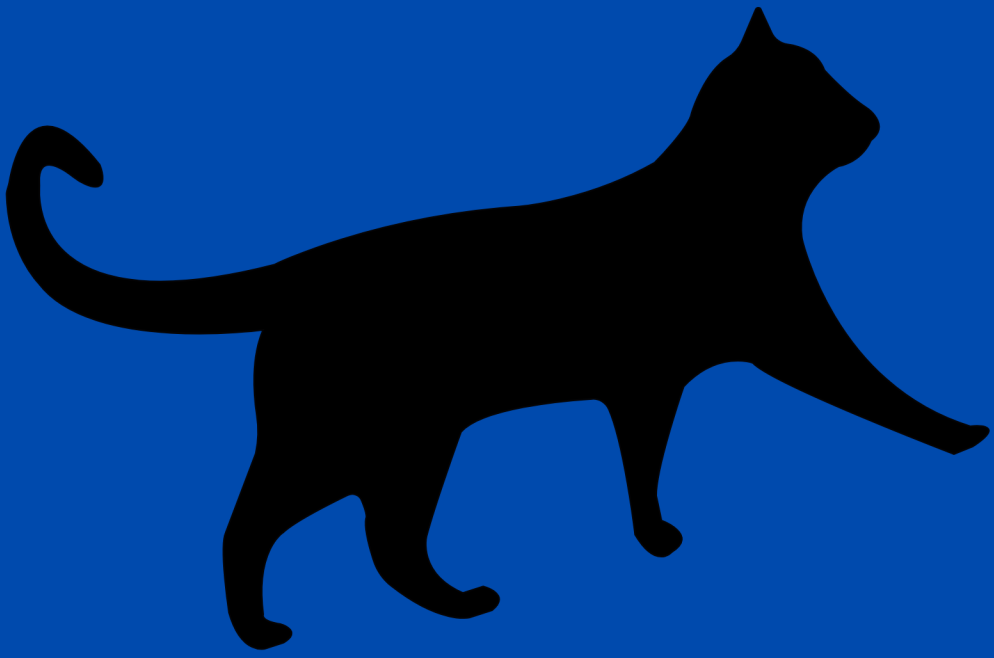
FRONTEND TESTING

Una introducción al mundo del testing orientado al frontend.



2025
@iagolast

Frontend Testing

Una introducción al mundo del testing orientado al frontend.

Iago Lastra Rodríguez

Este libro está disponible en https://leanpub.com/frontend-testing

Esta versión fue publicada el 2025-04-23

Índice general

1: Introducción

1.1: El típico flujo de trabajo

Antes de empezar, quisiera compartir la historia de Bob. Aunque es un personaje ficticio, representa a muchos programadores con los que me he cruzado a lo largo de mi carrera.

Bob es ingeniero de software en una de las tiendas en línea más destacadas de Europa. Esta mañana, le asignaron la tarea de escribir el módulo responsable de calcular los impuestos de las compras para la próxima versión del sitio web, que se lanzará en unos meses.

En su navegador, Bob mantiene abiertas dos pestañas: una con la descripción de la tarea, de la cual extrae los requisitos, y otra en localhost:3000, donde ejecuta una versión de desarrollo de la aplicación.

Su primer paso es crear un componente que muestre los impuestos aplicables al carrito de compras. En menos de media hora, escribe el código, lo guarda, abre la pestaña de su navegador, inicia sesión con un usuario de prueba, añade un producto al carrito y verifica que su recién creado componente muestra el IVA correctamente.

Posteriormente, Bob realiza este proceso con un usuario español y uno alemán, momento en el cual descubre un pequeño error en el cálculo de los impuestos alemanes. Regresa a su editor de código y soluciona el problema añadiendo una nueva línea de código, aproximadamente de la siguiente manera:

```
if (user.countryCode == 'GER') {
  // TODO: Review this code
  return price + computeVAT(price) * GER_VAT;
}
```

Es tarde y Bob no tiene ganas de revisar todo el código, así que decide reutilizar parte de él y deja un comentario para solucionarlo en el futuro.

Al concluir, vuelve a abrir el navegador. Esta vez, inicia sesión con un usuario de prueba alemán, añade un producto al carrito y dirige su mirada hacia la sección “precio total”, donde verifica que se muestra el precio correcto. Todo está perfecto. Bob guarda y sube su código al servidor…

¿Te identificas con Bob? ¿Se asemeja tu proceso de trabajo al de él?

Este método de escribir código, abrir el navegador, usar una consola, enviar una petición HTTP y revisar manualmente el resultado, o incluso

el uso excesivo de `console.log` para entender qué está ocurriendo en el software, es, según mi experiencia, el flujo de trabajo más común: **Programar algo y verificar manualmente el resultado**.

Reflexionemos sobre este proceso: ¿Es posible mejorarlo? ¿Es rápido? ¿Es cómodo? ¿Es fiable? ¿Qué sucede si necesitamos un método distinto para calcular los impuestos en cada país? ¿Cómo podemos estar seguros de no haber roto algo cada vez que realizamos cambios? ¿Es necesario probar manualmente una compra en cada país? ¿No existirá una manera más eficiente de hacer todo esto?

1.2: ¿Qué es un test?

Podríamos definir un test como:

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

De acuerdo con esa definición, el proceso mediante el cual Bob abre su navegador, elige un producto y verifica el valor del IVA, constituye un test. Sin darse cuenta, todo el mundo realiza tests: cambiar una función y establecer un punto de interrupción (breakpoint) para observar los valores de los parámetros es un test. Usar la pestaña de red (network) para comprobar el payload de una petición POST es un test; ejecutar el código e insertar un `console.log()` es un test… en resumen, todo son tests, pero son **tests manuales**.

El propósito principal de este texto es comprender que **esos tests manuales que realiza la mayoría de las personas pueden automatizarse**. Es factible alcanzar un punto en el que no sea necesario abrir el navegador; un punto en el que podamos dejar una documentación formal que indique a los nuevos desarrolladores qué es lo que debe hacer el sistema; un punto en el que ni siquiera sea necesario lanzar la aplicación, porque disponemos de herramientas más eficaces para verificar que nuestro trabajo está bien hecho: **tests automáticos**.

1.3: 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.

1.3.1: La tienda online sin tests automáticos.

Regresando al ejemplo de la tienda en línea, imaginemos que Bob acaba de programar el módulo para calcular impuestos, y como parte de un test manual sigue estos pasos:

1. Abrir la web.
2. Elegir un producto al azar, por ejemplo, una tostadora.
3. Verificar que el precio final es de 40€ con un IVA del 21%.

Meses después, a una colega de Bob, Alice, le asignan la tarea de programar un nuevo sistema de tickets de regalo con descuentos para el Black Friday.

Debido a una planificación deficiente, Alice dispone de poco menos de una tarde para completar su trabajo, así que se pone a programar de inmediato… Más tarde, abre localhost:3000 y sigue estos pasos:

- Abrir la tienda en línea.
- Elegir un producto al azar, por ejemplo, una maquinilla de afeitar.
- Aplicar un código de descuento de 5€.
- Comprobar que el precio final es de 15€ con un descuento aplicado de 5€.

Son las 11:30 de la noche y el despliegue tarda unos 20 minutos. ¡Alice necesita subir el código ahora! Prueba con un par de productos más y, al no observar nada inusual, sube el código y publica la nueva versión del sitio web.

A la mañana siguiente, todos los programadores recibieron el siguiente correo electrónico:

> Para: developers@empresa.com
>
> Buenos días,
>
> Les escribo para informarles que durante la noche hemos perdido unos 2.000.000€ en pedidos debido a un BUG introducido en la actualización de ayer.
>
> Por algún motivo, se estaba calculando incorrectamente el precio final para los clientes alemanes que utilizaban un ticket de regalo. Hemos revertido el cambio y vamos a investigar…

Alice estaba horrorizada. Aunque Carol revisó el código superficialmente, a nadie se le ocurrió que el nuevo módulo de impuestos pudiera interactuar de esta manera con el código de los tickets de regalo. Esa sección la escribió Bob, quien no estaba en la oficina ese día.

1.3.2: La tienda online con tests automáticos.

Ahora imaginemos la misma situación, pero en una empresa donde todos los programadores escriben tests automáticos para cada funcionalidad que desarrollan:

Bob escribe su test automático y el código de la funcionalidad. Una vez que todos los indicadores están en verde, se va a casa.

Desafortunadamente, los tests no pueden prevenir una mala gestión, así que la urgencia de desarrollar la funcionalidad de descuentos del Black Friday en el último minuto surge nuevamente.

Esta vez, Alice ejecuta los tests de la aplicación antes de subir sus cambios y en menos de dos segundos ve en 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 obtuvo 15€.

Tras revisar el código, Alice se da cuenta de que ha añadido una condición errónea que afecta el cálculo del IVA en ciertas situaciones. Corrige el error en dos minutos y vuelve a ejecutar los tests:

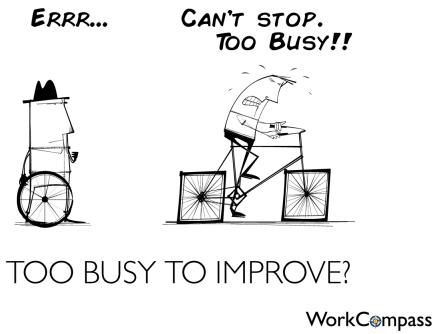
> Resultados de los tests: 5433 Tests OK, 0 Test FAIL.

Sube los cambios y se va a casa, tranquila. Al día siguiente, el único correo electrónico notable que recibe es uno que informa sobre un nuevo récord de ventas.

1.4: La importancia de los tests

1.4.1: Las excusas

Últimamente, es bastante inusual encontrar desarrolladores que abiertamente afirmen que los tests no son útiles. Sin embargo, casi todos tienen su excusa para no realizar tests.



"No tengo tiempo"

Esta es la excusa clásica… Imagínate a un piloto diciéndole a la tripulación que, como van con retraso, van a omitir las verificaciones antes del despegue.

Ahora piensa que comienzas a trabajar como repartidor de periódicos, un proyecto pequeño que puedes manejar durante un paseo matutino. Con el tiempo, el negocio crece y alguien te sugiere repartir en bicicleta. Respondes: "¡No tengo tiempo! A mi jefe no le interesan las bicicletas."

Es verdad que, en un día específico, puede resultar mucho más rápido repartir los periódicos a pie que aprender a montar en bicicleta para hacer la entrega, pero una vez que aprendas a montar, no habrá vuelta atrás: serás más rápido y tendrás una ventaja significativa sobre los repartidores que no saben montar.

Con los tests ocurre lo mismo. **No es que no tengas tiempo para hacer tests; ¡lo que probablemente falte es tiempo para aprender a realizarlos y aplicarlos a tu proyecto!**

"Muchos bugs no se podrían detectar incluso haciendo tests"

También es cierto que los cinturones de seguridad no evitan la muerte en el 100% de los accidentes, pero nadie cuestiona si es mejor llevarlos puestos, ¿cierto? De la misma manera, los tests no garantizan que no ocurran errores, pero sí ofrecen una medida adicional de seguridad para prevenirlos.

"Mi cliente no quiere tests"

Es comprensible que un cliente no quiera asumir el coste de que un equipo aprenda a escribir tests, pero una vez que el equipo está formado, la productividad aumenta considerablemente. Ningún cliente se opondría a tener un equipo más productivo.

"Mi código cambia constantemente, no tiene sentido hacer tests"

Aunque implícitamente me estás diciendo que los tests generan más trabajo, quiero señalar dos aspectos:

1. Si tus tests se rompen con cualquier mínimo cambio, quizás estén demasiado acoplados al código y deberías revisarlos.
2. Si los tests están bien hechos y lo que cambia son los requisitos, entonces es un problema de definición del producto, no de ingeniería.

"Mi código nunca cambia, no tiene sentido hacer tests"

¿Podría ser que no cambie precisamente porque no tiene tests y nadie se atreve a modificarlo? En mi experiencia, la única constante en el mundo del software es el cambio; de hecho, la palabra "software" misma lo sugiere.

"Mi código es muy difícil de probar"

Si te encuentras en esta situación, no te preocupes. Existen libros enteramente dedicados a trabajar con código heredado (legacy), que explican cómo introducir tests gradualmente.

En mi experiencia, una vez que un programador aprende a realizar tests correctamente, no hay vuelta atrás: los tests ahorran tiempo y dinero.

Cuando una base de código está bien testeada, se trabaja más rápido y con mayor confianza. El número de errores se reduce drásticamente y los refactores de código son más comunes, lo que contribuye a mantener un código más limpio. Existen estudios[1] que demuestran que la aplicación de TDD (Desarrollo Guiado por Tests) puede reducir el número de bugs entre un 40% y un 90%. Además, compañías líderes como Google o Facebook incluyen el testing en sus entrevistas de trabajo.

Si después de todo esto aún no estás convencido de darle una oportunidad al testing, te pido que leas la siguiente sección. Y si aún así sigues sin estar seguro, me gustaría conocer tus razones para intentar convencerte.

1.4.2: Los tres beneficios principales de los tests

Se puede afirmar que los tests desempeñan tres funciones principales: Documentación, Diseño y Verificación.

1.4.2.1: Documentación

Uno de los problemas más habituales en los equipos de desarrollo es la ausencia de documentación formal del código. A menudo, el conocimiento completo de una aplicación reside en unas pocas personas, convirtiéndolas en cuellos de botella o en lo que se conoce como silos de conocimiento.

Esta situación es tan prevalente que se ha acuñado el término "factor de autobús"[2] para referirse a cuántos desarrolladores tendrían que ser atropellados por un autobús para que el proyecto se vuelva insostenible.

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

> El archivo `Taxes.js` contiene utilidades para calcular los impuestos aplicables a una compra.
>
> Por defecto, se utiliza el IVA general de España del año 2010 (18%).
>
> Se ha migrado todo a `Taxes.new.js`.
>
> Los impuestos de Alemania siguen utilizando `Taxes.js` cuando el producto pertenece a la categoría "Electrónica"...

El problema es que mantener actualizados estos documentos con cada cambio requiere esfuerzo y, con el tiempo, la mayoría de la documentación termina quedando obsoleta.

Veremos que unos tests bien escritos funcionan como *documentación viva*. Documentación porque facilitan la comprensión de lo que debe hacer el código y viva porque, cuando el código cambia, los tests deben actualizarse obligatoriamente con cada nuevo requisito.

1.4.2.2: Diseño.

Aunque no todos se dan cuenta, los tests contribuyen significativamente a mejorar la calidad del código.

Para que un código sea fácil de probar, debe ser una pieza aislada de funcionalidad bien definida. Generalmente, si un código es fácil de probar, es probable que cumpla con los principios SOLID[3].

El acto de escribir tests obliga a reflexionar sobre la funcionalidad: cómo interactuarán los componentes, qué APIs y parámetros se utilizarán, etc.

Este proceso contribuye a que el resultado final sea más maduro y completo que si se hubiera programado la funcionalidad directamente. Metodologías como TDD (Desarrollo Guiado por Pruebas) previenen la generación de código innecesario porque se escribe únicamente el código necesario para pasar los tests.

Además, los tests facilitan los cambios y las refactorizaciones. En un entorno sin tests, es poco común que se mejore el código existente, ya que impera la ley de "si funciona, no lo toques", lo cual a menudo lleva a aplicar parches superficiales que degradan progresivamente la calidad del código. Por el contrario, en un entorno con tests, los refactors son habituales, permitiendo una limpieza y mejora constante del código.

1.4.2.3: Verificación.

Esta es la ventaja que históricamente se ha asociado más con los tests: permiten verificar que se cumplen una serie de requisitos de forma rápida y automática.

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

Durante las primeras etapas de desarrollo, las ventajas del testing pueden no ser evidentes, pero a medida que el equipo y el proyecto crecen, se hacen notorias. Incluso cuando hay solo un programador, es complicado recordar el funcionamiento de todas las características del sistema y, por tanto, es muy difícil asegurar que los cambios introducidos no afecten negativamente a lo ya desarrollado.

Con el tiempo, siempre se llega a una de estas situaciones:

1. El desarrollador no prueba todo el código, y tarde o temprano aparecerán bugs que resultarán en pérdida de tiempo y dinero.
2. El desarrollador prueba todo el código manualmente, invirtiendo demasiado tiempo y recursos.
3. El desarrollador utiliza tests automáticos para probar el código.

¿Cuál te parece la mejor opción?

1.4.3: ¿Debería escribir el test primero?

Mi recomendación es que sí. No conozco a ningún programador que haya empezado a hacer TDD (Desarrollo Guiado por Pruebas) correctamente y se haya arrepentido; una vez te acostumbras, no hay vuelta atrás. Dicho esto, lo realmente importante es que el código esté testeado, ¡da igual cómo!

Entre las ventajas de escribir el test primero destacamos:

Escribir el test después a menudo se pospone

Muchas personas que optan por escribir el test después se encuentran con que el código que han desarrollado no es fácil de probar, deciden dejarlo para más tarde, y todos sabemos lo que significa "dejar algo para más tarde" en el mundo del software.

Obliga a definir claramente el problema desde el principio

Si te esfuerzas en escribir los tests primero, necesitarás razonar sobre la API que va a tener el código, qué parámetros recibe y qué valores devuelve en cada caso. Este esfuerzo resulta en un análisis más profundo del problema en sí.

Desafortunadamente, es común encontrarse con issues mal definidas, donde los casos de uso no están claros, o el alcance de una tarea es nebuloso. Escribir el test primero proporciona una definición formal del criterio de aceptación de la issue, lo cual ayuda tanto al área de negocio como al equipo de desarrollo a analizar en profundidad los posibles escenarios.

Permite escribir menos código

Si el test está bien definido, solamente necesitarás escribir el código necesario para que el test pase.

Facilita retomar el trabajo más fácilmente

Si te interrumpen mientras estás programando y has escrito un test primero, esto te permite retomar el trabajo en el punto exacto donde lo dejaste previamente.

1.5: ¿Por dónde empiezo?

Ya hemos explorado la definición formal, pero, ¿cómo es exactamente un test? ¿Cómo lo escribo?

Lo primero que necesitas es un test runner. Un test runner es una herramienta que busca y ejecuta tus tests, muestra los resultados y puede incluso monitorizar tu código para ejecutar tests automáticamente cuando haces cambios. A día de hoy, Vitest[4] es uno de los test runners más utilizados y modernos en la comunidad JavaScript, destacando por su velocidad y excelente integración con herramientas como Vite. Para empezar a usar Vitest, simplemente necesitas instalarlo:

```
npm install -D vitest
```

Vitest comparte una API muy compatible con otros runners populares como Jest, facilitando la transición. Es común configurar Vitest como el comando de test por defecto en el `package.json`, de manera que al ejecutar `npm run test` o simplemente `npm t`, Vitest buscará automáticamente archivos cuyos nombres terminen en `.test.js`, `.spec.js` (y otros patrones configurables) y ejecutará los tests definidos en ellos.

```
// package.json
{
  "scripts": {
-   "test": "jest"
+   "test": "vitest"
  }
}
```

Ahora, vamos a crear un test de ejemplo llamado `hello.test.js`, donde verificamos que la función replace[5] de JavaScript funciona correctamente:

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

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

- **Nombre o Descripción**: Indica qué código estamos probando y qué resultado esperamos.
- **Arrange**: Una primera parte donde preparamos el entorno para el test.
- **Act**: Una segunda parte donde ejecutamos el código que queremos probar.
- **Assert**: Una parte final donde comparamos el resultado obtenido con el resultado esperado.

Finalmente, podemos ejecutar el test con:

```
npm t
```

Y obtendremos la siguiente salida por consola. La descripción del test nos da una idea clara de lo que estamos verificando, lo que destaca la importancia de escribir buenos tests.

```
✓ replace function should replace <name> with world (13ms)
```

2: Características de un test

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

2.1: Sensibilidad: Sensible vs. Insensible

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

2.2: Especificidad: Específico vs Inespecífico

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

2.3: Flexibilidad: Flexible vs Inflexible

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

2.4: Estabilidad: Estable vs Inestable

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

2.5: Precisión: Preciso vs Impreciso

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

2.6: Mantenibilidad: Mantenible vs Inmantenible

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

2.7: Velocidad: Veloz vs Lento

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

2.8: Profundidad: Profundo vs Superficial

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

2.8.1: Test superficial (shallow)

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

2.8.2: Test profundo (deep)

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

2.9: Conclusión

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

3: Tipos de Tests

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

3.1: Clasificación tradicional

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

3.1.1: Tests de Unidad

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

3.1.2: Tests de Integración

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

3.1.3: Tests de Extremo a Extremo

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

3.1.3.1: Propósito y Valor

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

3.1.3.2: Herramientas y Tecnologías

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

3.1.3.3: Desafíos y Consideraciones

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

3.1.3.4: Estrategias Efectivas

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

3.1.3.5: Ejemplo Práctico

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

3.2: Clasificación alternativa

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

3.3: Conclusión

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

4: Cómo distribuir los tests

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

4.1: Pirámide del testing

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

4.2: Trofeo del testing

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

4.3: El barco del testing

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

5: Ejemplos de frontend testing

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

5.1: Ejemplo I: Los lados de un triángulo

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

5.1.1: Enunciado

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

5.1.2: Vitest (y Jest)

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

5.1.3: Fase 0: Razonamiento

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

5.1.4: Fase 1: Nombre y descripción

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

5.1.5: Fase 2: Arrange

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

5.1.6: Fase 3: Act

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

5.1.7: Paso 4: Assert

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

5.1.8: Paso 5: Ejecutar los tests

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

5.1.9: Resumen

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

5.2: Ejemplo II: La aplicación de venta de entradas

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

5.2.1: Fase 1: Tests básicos

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

5.2.2: Fase 2: Añadiendo contexto

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

5.2.3: Fase 3: Añadiendo más contexto.

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

5.2.4: Fase 4: Compartiendo contexto

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

5.2.5: Fase 5: Simulando dependencias.

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

5.2.5.1: Test unitarios sociables

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

5.2.5.2: Test unitarios solitarios

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

5.2.6: Resumen

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

5.3: Ejemplo III: Viajando en el tiempo

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

5.3.1: Paso 0: Automatizando los tests

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

5.3.2: Paso 1: Abrir el navegador y cargar la aplicación

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

5.3.3: Paso 2: Comprobar que se muestra un botón con el texto “start”

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

5.3.4: Paso 3: Hacer click en el botón

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

5.3.5: Paso 4: Esperar hasta que el texto sea visible

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

5.3.6: Paso 5: Intervalos de tiempo más grandes

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

5.3.7: Conclusiones

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

5.4: Ejemplo IV: Validando contraseñas

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

5.4.1: 1. Resumen de la arquitectura

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

5.4.2: 2. Dos campos vacíos

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

5.4.3: 3. Escribiendo en el primer campo

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

5.4.4: 4. Escribiendo contraseñas diferentes en ambos campos

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

5.4.5: 5. Escribiendo contraseñas iguales en ambos campos

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

5.4.6: 6. Comprobar la interacción con el servidor.

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

5.4.7: 7. Comprobar errores de accesibilidad

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

5.4.8: Conclusión

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

6: Otros tipos de testing

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

6.1: E2E Testing

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

6.1.1: Cypress.io

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

6.1.1.1: Instalar Cypress

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

6.1.1.2: Configurar Cypress

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

6.1.1.3: Escribir un test

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

6.1.1.4: Ejecutar Cypress

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

6.1.2: Playwright

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

6.1.2.1: Instalar Playwright

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

6.1.2.2: Escribir un test

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

6.1.2.3: Ejecutar Playwright

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

6.2: Visual Regression Testing

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

6.2.1: BackstopJS

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

6.2.2: Chromatic

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

6.3: API Testing

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

6.3.1: Postman

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

6.4: Contract Testing

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

6.4.1: Pact

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

6.5: Snapshot testing

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

6.6: Property-based Testing

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

6.6.1: fast-check

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

6.7: Performance testing

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

6.7.1: Lighthouse

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

6.8: Manual testing

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

7: Testing doubles

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

7.1: Dummy

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

7.2: Stubs

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

7.3: Spies

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

7.4: Fake

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

7.5: Mock

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

8: Buenas prácticas

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

8.1: SWA: Should when and

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

8.1.1: ¿Qué unidad se prueba?

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

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

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

8.1.3: ¿En qué contexto realizamos esta prueba?

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

8.2: Partes de un test (AAA)

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

8.2.1: Arrange

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

8.2.2: Act

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

8.2.3: Assert

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

8.3: Un test solamente debería fallar por una razón

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

8.4: No incluir lógica en el test

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

8.5: No probar la implementación

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

8.6: No probar métodos privados.

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

8.7: Probar comportamiento en lugar de estado

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

8.8: Exportar Servicios

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

8.9: No confiar ciegamente en la cobertura

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

9: Conclusión

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

Notas

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

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

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

4 https://vitest.dev/

5 https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String/replace