Engineer

Prueba de extremo a extremo

🏃 prueba que recorre todo el sistema como un usuario

Qué es

Una prueba de extremo a extremo comprueba un sistema completo en ejecución tal como lo encuentra un usuario. Inicia la aplicación con su interfaz, backend, base de datos y configuración reales, recorre una tarea como registrarse o pagar un pedido y comprueba lo que vería el usuario al terminar. Su abreviatura habitual es prueba E2E, y la práctica se denomina pruebas de extremo a extremo.

En *Software Engineering at Google*, Adam Bender clasifica las pruebas en dos ejes. El tamaño se refiere a los recursos: una prueba pequeña permanece dentro de un proceso; una mediana puede utilizar varios procesos en una máquina y una grande puede abarcar varias máquinas. El alcance indica cuánto código ejercita una prueba; las de alcance amplio reciben nombres como pruebas funcionales, de extremo a extremo y de sistema. Joseph Graves, en el capítulo del libro sobre pruebas de mayor tamaño, añade que las de mayor alcance suelen incorporar varias dependencias reales y utilizar menos dobles de prueba.

Esas dependencias reales aportan fidelidad, que Graves define como cuánto refleja una prueba el comportamiento real del sistema probado. Una prueba unitaria de un formulario de pago demuestra que su código es correcto en un entorno aislado; una de extremo a extremo del mismo formulario demuestra que la página desplegada, la ruta que hay detrás, las reglas de precios y la base de datos concuerdan entre sí. El coste es la hermeticidad: cuanto más se parece un entorno a producción, más difícil resulta aislarlo de todo lo demás. Graves señala que ambas suelen entrar en conflicto directo.

En una aplicación web, la interfaz del usuario es el navegador, así que ahí se sitúa la prueba de extremo a extremo. Graves lo dice claramente: para el usuario final, «la API pública es la propia aplicación». Playwright Test, utilizado en el ejemplo siguiente, es el ejecutor que desarrolla el proyecto Playwright para las pruebas

de extremo a extremo de aplicaciones web. Incluye el ejecutor, las aserciones, el aislamiento y la ejecución en paralelo, y controla Chromium, Firefox y WebKit, con ventana o sin ella.

Utilice el concepto en una decisión

Aplique «¿Qué es una prueba de extremo a extremo?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Un equipo que lleva una idea del boceto a un producto al que sus clientes pueden acceder, sin pasarla entre tres proveedores.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Ingresos multiplicados por 10 con mapas GIS — 10x crecimiento de ingresos impulsado desde el producto cartográfico

Ingresos multiplicados por 10 con mapas GIS Póngase en contacto →

Por qué importa

Las pruebas pequeñas dejan huecos que solo puede cerrar una prueba del sistema montado, y Graves enumera los habituales. Un doble de prueba puede alejarse del componente que sustituye, y nada avisa a la prueba de que la implementación real ha cambiado. Una unitaria cubre el código dentro de un binario, mientras que la configuración del despliegue, los scripts de arranque y los ajustes de las instancias de producción quedan fuera. El tercer hueco son los problemas que solo aparecen bajo carga; el cuarto, comportamientos que nadie previó, que es como llegan al usuario la mayoría de los problemas. Bender escribe que Google reserva las pruebas grandes principalmente para ejecuciones de extremo a extremo del sistema completo, que comprueban su configuración más que una pieza concreta de código.

También comprueban la interfaz del sistema completo que utiliza el usuario. Un formulario que envía a una ruta recién renombrada por el backend, o un lanzamiento cuya configuración apunta al servicio equivocado, superan todas las pruebas unitarias de ambos lados y fallan la primera vez que los intenta utilizar una persona real.

El precio está en la velocidad y la fiabilidad. Graves señala que, de forma predeterminada, Google concede a una prueba grande 15 minutos o una hora antes de agotar el tiempo de espera. Pueden no ser herméticas, compartir recursos con otras pruebas y tráfico y, por tanto, ser no deterministas. Bender identifica las fuentes habituales de esa falta de determinismo: el reloj, las demoras de red, el orden de ejecución de los hilos, las interrupciones de hardware y el motor de representación del navegador. Por eso la orientación

aproximada del libro es una pirámide: alrededor del 80 por ciento de pruebas unitarias de alcance reducido, 15 por ciento de integración y 5 por ciento que ejercitan el sistema completo de extremo a extremo. Según Bender, las grandes sirven para comprobar que lo esencial sigue funcionando a medida que crece el producto; no deberían ser la principal forma de detectar errores del equipo. Los equipos de Google suelen reservarlas para las compilaciones y los lanzamientos, dejando su coste fuera del ciclo de desarrollo más rápido.

Cómo se aplica en la práctica

Considere una tienda que vende una tetera y pide a la API de un transportista una estimación de entrega. La especificación de Playwright para la primera compra de un invitado se lee como el propio recorrido:

```
// e2e/checkout.spec.ts
import { test, expect } from "@playwright/test";

test.beforeEach(async ({ page }) => {
    // The carrier's API belongs to someone else: answer for it here.
    await page.route("https://api.carrier.example/**", (route) =>
        route.fulfill({ status: 200, body: '{"days": 2}' }),
    );
});

test("a guest can buy a teapot", async ({ page }) => {
    await page.goto("/shop/teapot");
    await page.getByRole("button", { name: "Add to basket" }).click();
    await page.getByRole("link", { name: "Basket" }).click();
    await expect(page.getByText("Delivery in 2 days")).toBeVisible();
    await page.getByLabel("Email").fill("buyer@example.com");
    await page.getByRole("button", { name: "Place order" }).click();
    const thanks = page.getByRole("heading", { name: "Thank you" });
    await expect(thanks).toBeVisible();
    await page.getByRole("link", { name: "View your order" }).click();
    await expect(page.getByText("Teapot")).toBeVisible();
});
```

Cada localizador nombra lo que ve el usuario: un botón llamado «Add to basket», un campo etiquetado «Email» y un encabezado que dice «Thank you». Playwright recomienda primero localizadores por rol, y sus localizadores son estrictos: si coincide más de un elemento, la acción lanza una excepción en lugar de adivinar. Antes de cada clic o introducción de datos, espera a que se pueda actuar sobre el elemento, y una aserción como `toBeVisible()` sigue reintentando hasta que se cumple la condición; por eso la especificación no contiene esperas fijas. La

llamada `page.route` responde a las solicitudes del navegador al transportista con un cuerpo fijo, de modo que la prueba nunca depende de un servidor que el equipo no gestione. El patrón glob debe coincidir con toda la URL y `**` también abarca las barras. El enrutamiento solo alcanza las solicitudes que hace la propia página: si es el servidor de la tienda el que solicita datos al transportista, el navegador nunca ve esa solicitud y hay que colocar un doble en el lado del servidor. Cada prueba recibe además un contexto de navegador nuevo, que Playwright compara con un perfil de navegador recién creado; las cookies y la cesta de una prueba no pueden filtrarse a la siguiente.

La configuración inicia la aplicación e indica a Playwright dónde está:

```
// playwright.config.ts
import { defineConfig } from "@playwright/test";

export default defineConfig({
    testDir: "e2e",
    retries: process.env.CI ? 2 : 0,
    use: {
        baseURL: "http://localhost:3000",
        trace: "on-first-retry",
    },
    webServer: {
        command: "npm run start",
        url: "http://localhost:3000",
        reuseExistingServer: !process.env.CI,
    },
});
```

El bloque `webServer` ejecuta `npm run start` y espera a que responda la URL antes de la primera prueba. En un portátil reutiliza un servidor ya en marcha y en CI siempre inicia el suyo. `baseURL` permite escribir `/shop/teapot` en lugar de una dirección completa. `npx playwright test` ejecuta la batería sin ventana; `--headed` muestra la ventana del navegador y `npx playwright show-report` abre el informe HTML. De forma predeterminada, Playwright no reintenta ninguna prueba fallida; aquí CI permite dos reintentos. Una que falla y después pasa se informa como inestable en lugar de aprobada, y en el primer reintento se registra una traza para mostrar qué fue distinto.

A mayor escala, la capa de navegador representa la parte más pequeña del trabajo. La batería de pruebas creada entre Go, SQL y el navegador, creada en el marco de un puesto desempeñado desde 2025, tiene 981 funciones de prueba

Go en 310 archivos, 543 especificaciones de Vitest y Playwright, incluidos 48 archivos de extremo a extremo y 30 especificaciones de navegador, y 494 archivos de pruebas de comportamiento SQL. La tarea era comprobar el comportamiento de cada capa allí donde reside, en lugar de hacerlo todo desde fuera mediante un navegador: una función almacenada se prueba en la base de datos y no a través de tres capas de aplicación por encima. La página reconoce el coste: la batería es lenta, añade una carga a cada cambio y, con ese tamaño, necesita su propio mantenimiento.

Dónde surgen los problemas

La batería tiene forma de cucurucho de helado. Es el nombre de Bender para una batería cargada de pruebas de extremo a extremo y escasa en integración y unitarias. Suele ser lenta, poco fiable e incómoda, una forma habitual en proyectos que pasaron deprisa del prototipo a producción. Graves añade que, cuando el código es difícil de probar de forma unitaria, las de extremo a extremo son las únicas pruebas que se pueden escribir, y el proyecto ha creado código heredado en unos días. Cada servicio nuevo también multiplica los caminos por el sistema, por lo que una batería de extremo a extremo nunca puede cubrirlos todos. Lleve cada caso a la prueba más pequeña capaz de detectarlo, reserve el navegador para los recorridos que solo puede mostrar el sistema entero y avance hacia la pirámide desde los primeros días del proyecto.

Los selectores siguen al marcado. Un localizador como `button.buttonIcon.episode-actions-later`, el ejemplo de Playwright de lo que se debe evitar, se rompe cuando alguien renombra una clase aunque el usuario no vea ningún cambio. Graves advierte de que las pruebas de interfaz tienen fama de poco fiables y caras, en parte porque los cambios de aspecto las rompen mientras el comportamiento sigue igual. Localice elementos por rol, etiqueta o texto, como los encuentra el usuario, o por un identificador de prueba controlado por el equipo. Si el backend tiene una API pública, divida el recorrido en el límite entre interfaz y API, y ejecute el resto de la prueba de extremo a extremo a través de la API.

La prueba duerme. Una espera fija es demasiado corta un día lento y demasiado larga cualquier otro día. Graves señala que las pruebas basadas en pausas y tiempos de espera empiezan a fallar todas cuando se sobrecargan las máquinas que las ejecutan, y los reintentos añaden carga. Las buenas prácticas de Playwright muestran una variante más sutil: `expect(await page.getByText("welcome").isVisible()).toBe(true)` comprueba una vez y continúa, mientras que `await expect(page.getByText("welcome")).toBeVisible()` espera a la

condición. Utilice aserciones que esperen las condiciones de la web; cuando una prueba espere al backend, consulte periódicamente si ha cambiado el estado, con un tiempo de espera máximo.

Las pruebas comparten datos. Una crea una cuenta que espera encontrar la siguiente, o dos procesos paralelos compran la última tetera disponible. La batería pasa en un orden y falla en otro, o solo al ejecutarse en paralelo. Playwright aconseja aislar por completo cada prueba, con su propio almacenamiento, sesión, datos y cookies, y controlar los datos de cualquier base que utilicen. Cree lo que necesita cada prueba con nombres únicos y nunca dependa de lo que dejó una anterior.

El servidor de otra entidad forma parte de la prueba. Incluir en la prueba un proveedor de pagos, un transportista o un servicio de mapas reales hace depender la batería de un sistema que el equipo no controla, y enviarle tráfico puede costar dinero. Graves desaconseja que las pruebas automatizadas llamen a una API real de terceros y considera esa dependencia una frontera importante para dividir las pruebas; Playwright aconseja probar solo lo que se controla. Responda en lugar del tercero con `page.route` cuando la llamada proceda del navegador, como en el ejemplo, o con un doble al que se configure el servidor para llamar cuando este haga la solicitud. Deje que esa dependencia marque el final de la prueba.

Los reintentos ocultan la inestabilidad. Reintentar en CI convierte una compilación en rojo en una en verde, y el coste se acumula. Bender desarrolla las cuentas: con 10.000 pruebas al día, una probabilidad del 0,1 por ciento de falso fallo en cada una supone 10 fallos intermitentes diarios que investigar; al acercarse la inestabilidad al 1 por ciento, las pruebas empiezan a perder valor. Reejecutar usa tiempo de máquina para ahorrar tiempo de ingeniería y solo aplaza la causa raíz. La documentación de pytest lo explica desde otro ángulo: las pruebas de mayor nivel dependen de más estado y, por ello, fallan de forma intermitente más a menudo. Trate cada prueba que Playwright informe como inestable como un error, abra la traza del reintento y corrija la prueba o elimínela.

Lecturas adicionales

- Visión general de las pruebas — El capítulo de Adam Bender en *Software Engineering at Google*: tamaño y alcance, pirámide, cucurucho de helado y coste de las pruebas inestables.

- Pruebas de mayor tamaño — Joseph Graves sobre fidelidad, huecos que dejan las unitarias, límites con la interfaz y los terceros, y eliminación de la inestabilidad.
- Instalación — La introducción de Playwright: instalar Playwright Test, ejecutar la prueba de ejemplo, modo con ventana e informe HTML.
- Buenas prácticas — Probar el comportamiento visible, aislar pruebas, excluir servidores de terceros, localizadores y aserciones que esperan las condiciones de la web.
- Reintentos — Cómo reintenta Playwright las pruebas fallidas y clasifica cada una como aprobada, inestable o fallida.
- Pruebas inestables — La explicación de pytest de por qué fallan de forma intermitente: estado del sistema sin controlar, orden de ejecución, seguridad al trabajar con varios hilos y qué hacer al respecto.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es una prueba de extremo a extremo?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/end-to-end-test/