

O'REILLY®

Full Stack Testing

A Practical Guide for
Delivering High Quality Software



Gayathri Mohan
Foreword by Dr. Rebecca Parsons

Elogios para *Full Stack Testing*

Desde las pruebas exploratorias manuales hasta la creación de estrategias de pruebas en varias dimensiones de la calidad y el trabajo con tecnologías emergentes, este libro abarca mucho terreno tanto para los analistas de calidad principiantes como para los experimentados. Gayathri ha hecho un trabajo fenomenal al destilar la teoría suficiente para introducir el tema y seguirlo con ejemplos prácticos para que puedas aplicarlos en tus proyectos con las herramientas y marcos existentes.

—Bharani Subramaniam, responsable de tecnología
de Thoughtworks India

Un amplio estudio de estrategias y patrones de comprobación que cubre el tema con amplitud y profundidad. Los fundamentos teóricos de las distintas formas de comprobación están respaldados por ejemplos prácticos en varios capítulos. El libro de Gayathri debería llegar a los escritorios de las personas que escriben (y, por tanto, están obligadas a probar) software.

—Saleem Siddiqui, autor de *Aprendizaje del desarrollo basado en pruebas*

Este libro proporciona una visión a vista de pájaro de las pruebas de pila completa y te ayudará a aprender sobre las pruebas y a mejorar los procesos corporativos relacionados con las pruebas de software. Recomendaría el libro a ingenieros de control de calidad, directores de proyectos técnicos y arquitectos de software. El libro ofrece un mapa ferroviario de los distintos caminos y enfoques que pueden aplicarse e investigarse en función del alcance de la aplicación, el presupuesto y los plazos.

—Nigar Akif Movsumova, ingeniera de software en
EPAM Systems

El término desarrollo full stack se refiere a las habilidades adicionales que debe tener un desarrollador para llevar a cabo su trabajo. La prueba de pila completa se refiere al software que se prueba, y abarca todas las tecnologías, procesos, habilidades de las personas y diversos tipos de pruebas que deben realizarse para mejorar el software. Full Stack Testing, de Gayathri Mohan, cubre con perspicacia estos temas polifacéticos, capacitando a los lectores para entregar software de alta calidad.

—Srinivasan Desikan, profesor adjunto y autor de
Software Testing: Principios y Prácticas

Como los proverbiales miembros de un equipo con los ojos vendados que intentan tantear individualmente su camino para comprender a un elefante, el libro de Gayathri proporciona la perspectiva necesaria para que los equipos comprendan una visión holística de las pruebas. Aunque las pruebas individuales producen resultados positivos, comprender la pila completa permite obtener mejores resultados de todo el proyecto.

—Neal Ford, director/arquitecto de software/meme wrangler en Thoughtworks y autor de *Arquitectura de software: The Hard Parts*

Pruebas Full Stack

Guía práctica para entregar software de alta calidad

Gayathri Mohan



Beijing • Boston • Farnham • Sebastopol • Tokyo

Pruebas Full Stack

por Gayathri Mohan

Copyright © 2022 Gayathri Mohan. Todos los derechos reservados.

Impreso en los Estados Unidos de América.

Publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Los libros de O'Reilly pueden adquirirse para uso educativo, empresarial o promocional de ventas. También están disponibles ediciones en línea de la mayoría de los títulos (<http://oreilly.com>). Para más información, ponte en contacto con nuestro departamento de ventas a empresas/instituciones: 800-998-9938 o corporate@oreilly.com.

- Editora de Adquisiciones: Melissa Duffield
- Editora de Desarrollo: Jill Leonard
- Editor de producción: Jonathon Owen
- Correctora: Rachel Head
- Correctora: Liz Wheeler
- Indexador: nSight, Inc.
- Diseñador de interiores: David Futato
- Diseñadora de la portada: Karen Montgomery
- Ilustradora: Kate Dullea
- Junio 2022: Primera edición

Historial de revisiones de la primera edición

- 2022-06-03: Primera publicación

Consulta <http://oreilly.com/catalog/errata.csp?isbn=9781098108137> para conocer los detalles del lanzamiento.

El logotipo de O'Reilly es una marca registrada de O'Reilly Media, Inc. *Full Stack Testing*, la imagen de portada y la imagen comercial relacionada son marcas registradas de O'Reilly Media, Inc.

Las opiniones expresadas en esta obra son las del autor y no representan las del editor. Aunque el editor y el autor se han esforzado de buena fe por garantizar que la información y las instrucciones contenidas en esta obra sean exactas, el editor y el autor declinan toda responsabilidad por errores u omisiones, incluida, sin limitación, la responsabilidad por daños derivados del uso de esta obra o de la confianza depositada en ella. El uso de la información y las instrucciones contenidas en esta obra es por tu cuenta y riesgo. Si alguna muestra de código u otra tecnología que esta obra contenga o describa está sujeta a licencias de código abierto o a derechos de propiedad intelectual de terceros, es tu responsabilidad asegurarte de que su uso cumple dichas licencias y/o derechos.

Este trabajo forma parte de una colaboración entre O'Reilly y Harness. Consulta nuestra [declaración de independencia editorial](#).

978-1-09810-813-7

[LSI]

Prólogo

El término *desplazar a la izquierda*, que se refiere a realizar una actividad antes o a la izquierda en una línea temporal, es cada vez más común. Oímos hablar de por qué es importante desplazar a la izquierda el diseño, la seguridad y, lo más relevante aquí, las pruebas. Adelantar las pruebas en el ciclo de vida del desarrollo de software disminuye el coste y la complejidad de la corrección de errores, porque se encuentran más cerca del momento en que se crearon, lo que establece un mayor contexto de lo que podría haber causado que las cosas fueran mal. Cuando pensamos en cosas como las pruebas de rendimiento, podemos empezar a buscar tendencias antes de preocuparnos realmente por los valores concretos. Esto también nos permite detectar cuando el rendimiento empeora significativamente. Entonces podemos explorar si esto significa que hemos dado con algo que es fundamentalmente menos eficaz o si tal vez simplemente cometimos un error que causó la degradación del rendimiento.

Aunque desplazar las pruebas a la izquierda significa que éstas se ejecutan contra un software que se sabe incompleto y sujeto a cambios, la mayor capacidad de solucionar los problemas que surjan compensa con creces los costes de las pruebas continuas, sobre todo cuando una parte significativa del conjunto de pruebas está automatizada. Aunque algunas pruebas y algunos estilos de pruebas, como las pruebas exploratorias, deben hacerse manualmente, las pruebas que puedan automatizarse deben hacerse.

Hay que hacer todo tipo de pruebas, y el título del libro de Gayathri es acertado. Pruebas de pila completa ofrece una visión global de las pruebas en toda la pila, examinando el rendimiento, la interfaz de usuario, el contrato, las pruebas funcionales de extremo a extremo, las pruebas unitarias e incluso las pruebas de accesibilidad. La cuestión para muchos implicados en las pruebas gira en torno a saber cómo realizar pruebas en toda la pila. Ahí es donde entra este

libro. Aunque hay muchos libros sobre pruebas e incluso sobre pruebas ágiles, que abogan por desplazar las pruebas hacia la izquierda, el libro de Gayathri analiza en profundidad cada aspecto de las pruebas de una aplicación moderna. El libro describe los problemas que surgen en cada aspecto de las pruebas y examina los principios y estrategias que se aplican a ese aspecto de las pruebas.

A continuación, cada una de estas secciones incluye una serie de ejercicios prácticos que demuestran concretamente cómo realizar realmente dichas pruebas. Ahora bien, reconozco que los ejercicios concretos y las herramientas incluidas en los ejercicios pueden cambiar y evolucionar con el tiempo. Sin embargo, estos ejercicios son valiosos aunque cambien las herramientas, porque muestran cómo utilizarlas para construir los tipos de pruebas adecuados. Los ejercicios concretan el enfoque de las pruebas; las herramientas proporcionan la capacidad de experimentar con pruebas de esa naturaleza. Las herramientas seguirán evolucionando inevitablemente, pero las estrategias de comprobación que aprenderás tendrán una vida útil mucho más larga.

La gama de enfoques de pruebas del libro de Gayathri es amplia, y abarca el análisis estático, las estrategias de pruebas de datos e incluso las pruebas exploratorias. Dada la creciente complejidad de nuestros sistemas de software, el papel de las pruebas exploratorias es cada vez más importante. Además, las pruebas de seguridad tienen su propio capítulo, pues todos sabemos lo vulnerables que son nuestros sistemas ante los piratas informáticos. Las pruebas de accesibilidad también tienen su capítulo, en el que se describe cómo podemos hacer que nuestros sistemas sean más fáciles de usar, incluso para los discapacitados.

Cada aspecto de las pruebas requiere analizar qué tipo de cosas pueden salir mal y, a continuación, crear una estrategia de pruebas para descubrir las cosas que han salido mal. Un conjunto de pruebas correctamente construido en toda la gama de tipos de pruebas proporciona la red de seguridad que nos permite hacer evolucionar

nuestros sistemas de software con confianza. El libro de Gayathri, basado en su experiencia probando distintos tipos de sistemas, guía a los profesionales del software en la creación de estrategias y conjuntos de pruebas adecuados.

Dra. Rebecca Parsons

Director de Tecnología de Thoughtworks, coautor de Building Evolutionary Architectures

Prefacio

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

Si trabajas en la industria del software, es muy poco probable que no te hayas puesto el sombrero de las pruebas al menos una vez, independientemente de tu función. Esto se debe a que las pruebas son un aspecto integral de la ingeniería de software, entrelazado en cada etapa del ciclo de entrega del software. Con la adopción exponencial de la digitalización en la actualidad, en la que diversas aplicaciones web y móviles se han integrado tanto en la vida cotidiana de las personas, las pruebas en diversas dimensiones de la calidad se han convertido en un imperativo.

Cuando observamos las pruebas como disciplina del software, podemos ver cómo han experimentado su propia trayectoria de evolución a lo largo de sus muchas décadas de existencia, creciendo para incorporar nuevas prácticas, marcos, metodologías y herramientas. Las pruebas manuales han evolucionado hasta convertirse en pruebas exploratorias manuales, y siguen siendo una parte fundamental de la disciplina de las pruebas en la actualidad. Entretanto, el auge de las pruebas automatizadas combinadas con las prácticas de integración continua e implementación continua (CI/CD) ha hecho que el valor derivado de las pruebas se dispare. Más allá de los casos de uso funcionales, la prueba automatizada de requisitos interfuncionales como el rendimiento, la seguridad y la fiabilidad para recibir información holística y ofrecer continuamente software de alta calidad es la necesidad crítica del momento. Esta es la razón por la que las pruebas de pila completa se consideran una especialización deseable hoy en día en la industria. Supongo que

estás aquí porque quieres transpilarte en un probador de pila completa para poder ofrecer software de alta calidad en el trabajo: en primer lugar, felicitaciones por tu compromiso, y en segundo lugar, ibienvenido a bordo!

Por qué escribí este libro

Me gustaría decirte humildemente que muchos expertos en pruebas antes que yo podrían haber escrito este libro, y no hacía falta que fuera yo. Tal vez sus responsabilidades no les dejaron tiempo, o les faltó inclinación; sea cual sea la razón, la oportunidad ha recaído en mí, iy estoy agradecido por ello! (Aunque si algún otro experto hubiera escrito este libro cuando yo era una principiante en las pruebas, me habría ahorrado mucho esfuerzo: Tuve que rebuscar en cientos de blogs y probar docenas de herramientas yo mismo para adquirir las habilidades de comprobación que he acumulado a lo largo de muchos años).

A través de mi experiencia asesorando a clientes en mi trabajo diario, he observado que los equipos que han aplicado una estrategia de pruebas inteligente han tenido éxito en su mayoría, mientras que la mayoría de los que no lo hicieron fracasaron estrepitosamente. Por ejemplo, he visto equipos de clientes que confiaban exclusivamente en las pruebas de extremo a extremo basadas en la interfaz de usuario y se quemaban con las tareas de mantenimiento, o que sólo hacían pruebas manuales y se enfrentaban a un montón de defectos de producción. Algunos equipos sólo hacían pruebas funcionales, sin descubrir problemas no funcionales críticos. En general, estos equipos se caracterizaban por una mala calidad del software, un equipo insatisfecho y una falta de perímetro competitivo. Me sorprende que siga existiendo tal sesgo en la comprensión de las prácticas de comprobación, cuando la comprobación como disciplina existe desde hace décadas. Sólo puedo suponer que esto se debe en gran medida a la falta de

talento para las pruebas en el sector, y con la actual guerra fría entre las empresas de software para saquear a los mejores talentos, es justo compartir y difundir ampliamente los conocimientos.

Aunque existen varios tutoriales de pruebas sobre herramientas concretas, no hay una narrativa coherente sobre cómo actualizarse en las tendencias actuales de las pruebas, con ejemplos prácticos que utilicen distintas herramientas. Y para muchas competencias especializadas, como las pruebas de seguridad y accesibilidad, no hay muchos materiales consumibles para que los lean los principiantes. Este libro pretende ser un recurso exhaustivo que permita a un principiante en pruebas perfeccionarse hasta un nivel de principiante avanzado en todas las habilidades esenciales para las pruebas de aplicaciones web y móviles de hoy en día.

Si te preguntas qué quiero decir con principiante avanzado, me refiero al modelo Dreyfus de adquisición de habilidades, que elabora cinco etapas a través de las cuales una persona progresiona a medida que adquiere una habilidad: principiante, principiante avanzado, competente, competente y experto. Este libro está escrito con el ambicioso objetivo de catapultar a sus lectores a través de las dos primeras etapas en 10 destrezas de comprobación diferentes, con ejemplos prácticos. Dado que la tercera etapa es la de competente, que sólo puede alcanzarse con una práctica exhaustiva, creo que el libro lleva a sus lectores tan lejos como puede!

¿Quién debería leer este libro?

Este libro está pensado principalmente para principiantes en pruebas de software y para profesionales de pruebas de software ya existentes que deseen ampliar sus conocimientos. Dicho esto, cualquier función de software cuyas responsabilidades se solapen con las pruebas, como un desarrollador de aplicaciones o un ingeniero DevOps, podría beneficiarse del libro. En todos los casos, un requisito fundamental es poseer algunos conocimientos de

codificación, especialmente en Java, ya que el libro contiene ejercicios prácticos en Java y, en algunos lugares, en JavaScript. Además, si eres un lector nuevo en la industria del software, te recomendaría que hicieras una lectura preliminar sobre los procesos de desarrollo de software, como las metodologías ágil y en cascada, antes de sumergirte en este libro.

Navegar por este libro

El libro comienza con una introducción a las pruebas de pila completa y profundiza en las 10 habilidades de pruebas que son esenciales para ofrecer aplicaciones web y móviles de alta calidad. Una vez establecidos los fundamentos, hay 10 capítulos independientes de desarrollo de habilidades. Cada uno de estos capítulos contiene los siguientes elementos estructurales:

- Los temas esenciales para la contextualización se agrupan bajo el epígrafe "Bloques de construcción". Si eres nuevo en esta habilidad, esta sección te dará una idea de lo que implica y de por qué y dónde hay que aplicarla.
- Le sigue una sección de estrategia, que explica cómo aplicar la habilidad en una situación determinada.
- Luego hay ejercicios que guían a los lectores con instrucciones paso a paso sobre la ejecución de la habilidad utilizando múltiples herramientas.
- También hay una sección "Explora más herramientas" en algunos capítulos, en la que se discuten más a fondo herramientas paralelas similares a las discutidas en la sección de ejercicios, u otras herramientas que pueden añadir valor en algún momento para los lectores durante su práctica, con el fin de enriquecer la comprensión de la habilidad por parte del lector.

- Por último, encontrarás mis puntos de vista, basados en observaciones y experiencias personales, en algunos de los capítulos, seguidos de los puntos clave, que son un resumen conciso de las lecciones aprendidas en cada capítulo.

Tras los 10 capítulos de desarrollo de habilidades, el libro habla de cómo avanzar en el testing con la ayuda de los primeros principios y las habilidades blandas individuales. También hay un capítulo extra para lectores entusiastas que sirve de introducción a las pruebas en tecnologías emergentes. Presenta un resumen sobre las pruebas en cuatro tecnologías emergentes -AI/ML, blockchain, IoT y AR/VR- con la intención de ayudar a los lectores a iniciar su aprendizaje también en esas áreas.

Convenciones utilizadas en este libro

En este libro se utilizan las siguientes convenciones tipográficas:

Cursiva

Indica nuevos términos, URL, direcciones de correo electrónico, nombres de archivo y extensiones de archivo.

Constant width

Se utiliza en los listados de programas, así como dentro de los párrafos para referirse a elementos del programa como nombres de variables o funciones, bases de datos, tipos de datos, variables de entorno, sentencias y palabras clave.

Constant width bold

Muestra comandos u otros textos que deben ser tecleados literalmente por el usuario.

Constant width italic

Muestra el texto que debe sustituirse por valores proporcionados por el usuario o por valores determinados por el contexto.

CONSEJO

Este elemento significa un consejo o sugerencia.

NOTA

Este elemento significa una nota general.

ADVERTENCIA

Este elemento indica una advertencia o precaución.

Aprendizaje en línea O'Reilly

NOTA

Durante más de 40 años, *O'Reilly Media* ha proporcionado formación tecnológica y empresarial, conocimientos y perspectivas para ayudar a las empresas a alcanzar el éxito.

Nuestra red única de expertos e innovadores comparten sus conocimientos y experiencia a través de libros, artículos y nuestra plataforma de aprendizaje online. La plataforma de aprendizaje en línea de O'Reilly te ofrece acceso bajo demanda a cursos de formación en directo, rutas de aprendizaje en profundidad, entornos de codificación interactivos y una amplia colección de textos y vídeos

de O'Reilly y de más de 200 editoriales. Para más información, visita <https://oreilly.com>.

Cómo contactar con nosotros

Dirige tus comentarios y preguntas sobre este libro a la editorial:

- O'Reilly Media, Inc.
- 1005 Gravenstein Highway Norte
- Sebastopol, CA 95472
- 800-998-9938 (en Estados Unidos o Canadá)
- 707-829-0515 (internacional o local)
- 707-829-0104 (fax)

Tenemos una página web para este libro, donde se enumeran erratas, ejemplos y cualquier información adicional. Puedes acceder a esta página en <https://oreil.ly/full-stack-testing>.

Envía un correo electrónico [a bookquestions@oreilly.com](mailto:a.bookquestions@oreilly.com) para comentar o hacer preguntas técnicas sobre este libro.

Para noticias e información sobre nuestros libros y cursos, visita <https://oreilly.com>.

Encuéntranos en LinkedIn: <https://linkedin.com/company/oreilly-media>

Síguenos en Twitter: <https://twitter.com/oreillymedia>

Míranos en YouTube: <https://www.youtube.com/oreillymedia>

Agradecimientos

Al principio de mi carrera, ni en mi imaginación más descabellada habría pensado en escribir un libro técnico completo, ¡y además para O'Reilly! Fueron la inspiración, la motivación y el entorno enriquecedor que me proporcionó Thoughtworks los que me llevaron por este camino, y estoy increíblemente agradecida por estar asociada a un grupo tan encantador de tecnólogos apasionados y líderes alentadores. Sin ningún orden en particular, me gustaría expresar mi agradecimiento y reconocer el apoyo que he recibido de algunas de las increíbles personas de Thoughtworks: Prasanna Pendse, que anima a todo el mundo a fijarse metas altas y, cuando me lo propuse, se aseguró de que recibiera el apoyo adecuado hasta el final; Bharani Subramanian, que trabajó estrechamente conmigo hasta la finalización del libro, compartiendo sus ideas esclarecedoras que dieron forma a cada uno de los capítulos; y Pallavi Vadlamani, una amiga íntima más que una colega, que también trabajó estrechamente conmigo desde las primeras fases y revisó cada capítulo. Satish Viswanathan, Kief Morris, Sriram Narayan, Neal Ford y Sudhir Tiwari son algunos de los que me han prestado su apoyo en las distintas fases de desarrollo de este libro; ¡es realmente inestimable contar con personas tan expertas que comparten sus sabias y oportunas orientaciones! También me gustaría dar las gracias especialmente a la Dra. Rebecca Parsons, CTO de Thoughtworks y mi modelo a seguir, que escribió el prólogo y tuvo la amabilidad de ofrecerse voluntaria para revisar los capítulos desde las fases de borrador. ¡¿Qué más apoyo podría pedir realmente a una organización?!

Mi más sincero agradecimiento al equipo de O'Reilly: especialmente a Jill Leonard y Melissa Duffield, por crear el espacio adecuado para que el libro se lanzara con éxito, y a los revisores técnicos, Chris Northwood, Alexander Tarlinder, Srinivasan Desikan, Saleem Siddiqui, Ian Molyneaux y Nigar Movsumova, que han aportado sus

comentarios sobre cada detalle y han conseguido que el libro llegue al estado en que se encuentra hoy.

También quiero dejar constancia de mi exquisito aprecio y gratitud a mi mentor a largo plazo, Dhivya Arunagiri, que ha pasado varios años reforzando mi confianza y ayudándome a dar forma a mi carrera, y a mis amigos, que han sido una sólida fuente de consuelo siempre que me agotaba escribir junto con el trabajo y los compromisos familiares en medio de una pandemia. También aprovecho esta oportunidad para expresar mi más sincero amor y agradecimiento a mis siempre alentadores y comprensivos padres.

Por último, pero no por ello menos importante, una llamada especial a mi querido marido, Manoj Mahalingam, que es una inspiración, un amigo y un guía, y sin el cual este libro no existiría hoy. Me gustaría dedicarle este libro a él y a mi encantadora hija, Magathi Manoj, por permitirme el espacio mental y el tiempo tan necesarios durante varias noches, fines de semana y vacaciones durante más de un año mientras trabajaba en este proyecto.

De hecho, mientras escribo esto, pienso en lo bendecida que soy por estar rodeada de un grupo tan increíble de familiares, amigos y colegas. ¡Muchas gracias a todos! Os estaré eternamente agradecida.

Capítulo 1. Introducción al Full Stack Testing

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

En el mundo actual, la digitalización es necesaria para sostener y hacer crecer cualquier negocio. Muchas empresas están abriendo camino en este aspecto, mientras que otras se encuentran en las primeras fases de modernización de sus plataformas digitales existentes.

La digitalización es clave para ampliar el alcance de una empresa desde una comunidad local a una escala global, lo que se traduce en más adopción y más ingresos. Casi todas las empresas a pequeña y gran escala de diversos sectores, como la sanidad, el comercio minorista, los viajes, el mundo académico, los medios sociales, la banca y el entretenimiento, diseñan planes para avanzar en sus estrategias digitales como medida fundamental para llegar a nuevos segmentos de clientes y obtener mayores beneficios.

En este viaje hacia la digitalización y la modernización, la innovación se convierte en un motor crucial. Las empresas que innovan constantemente siguen siendo relevantes y prosperan durante muchas décadas. Netflix es un ejemplo clásico: empezó como un portal de alquiler de DVD en línea en los años 90, y luego se aventuró en el streaming en línea en 2007, canibalizando su propio negocio de alquiler de DVD. Más tarde empezó a producir contenido original, llamado *Netflix Originals*. A finales de 2021, Netflix era el

mayor servicio de streaming online, con más de 200 millones de suscriptores en todo el mundo.

El espacio tecnológico ha ido evolucionando paralelamente a estos negocios innovadores para adaptarse a sus necesidades cada vez más avanzadas. Atrás quedaron los días en que la gente estaba dispuesta a hacer cola para comprar entradas de cine, conducir hasta una tienda en un lugar remoto para comprar un producto especializado, o llevar encima una lista de la compra escrita a mano en busca de detalles concretos. La tecnología facilita esas tareas cotidianas. Podemos sentarnos en casa y transmitir nuestros programas de entretenimiento favoritos con sólo pulsar un botón, probarnos un vestido nuevo virtualmente, programar la entrega periódica de los artículos de nuestras listas de la compra, preparar un café con un comando de voz, y mucho más.

Con el rápido ritmo de evolución de la tecnología, las estrategias de producto deben ser versátiles, atendiendo a las distintas necesidades de los clientes para defenderse de la competencia en el sector de que se trate. Ya no basta con construir un sitio web; hay que ampliar horizontes. Piensa en empresas de transporte como Uber y Lyft, que ofrecen distintas formas de acceder a sus servicios: la web, plataformas móviles Android e iOS, incluso un **chatbot de WhatsApp**. Este tipo de estrategia de producto versátil ha ayudado a estas empresas a expandirse por todo el mundo y superar a sus competidores.

La innovación y la versatilidad ayudan a las empresas a tener éxito en la adquisición de una masa crítica de clientes. Pero el reto consiste entonces en prosperar aún más, obteniendo más ingresos y ganando más clientes. Hemos visto cómo gigantes del sector como Amazon aprovechan su base de clientes existente para hacer ventas cruzadas de servicios y productos como estrategia de expansión. Amazon, que empezó como una librería en línea, ahora realiza ventas cruzadas de productos que van desde artículos frescos de despensa hasta electrónica, pasando por ropa, joyas y mucho más,

satisfaciendo la demanda de bienes de consumo de los clientes en casi todos los segmentos del mercado.

¿Por qué hablamos de estos detalles en un libro sobre pruebas de software? Porque la industria del software actual satisface todas estas necesidades empresariales, proporcionando las herramientas para innovar nuevas ideas de productos, darles vida y ampliarlas para llegar a nuevos segmentos de clientes en todo el mundo.

Indudablemente, los equipos de desarrollo de software están al perímetro, especialmente cuando la necesidad del momento es entregar *con alta calidad*. De hecho, la calidad del software se ha convertido en un criterio innegociable en el competitivo mercado actual. Comprometerse con la calidad del software equivale a correr una carrera sabiendo que la perderás. Esto se ha visto reforzado por muchos ejemplos del mundo real. Por ejemplo, en octubre de 2014, los gigantes indios del comercio electrónico Snapdeal y Flipkart se enfrentaron en su venta de temporada tras meses de marketing. Por desgracia, **el sitio web de Flipkart se colapsó** varias veces durante la venta del "Big Billion Day" debido a la abrumadora demanda, lo que le hizo perder muchos clientes y una gran cantidad de ingresos en favor de Snapdeal. Del mismo modo, Yahoo! no pudo seguir el ritmo de sus competidores (a pesar de ser uno de los primeros de su clase en el mercado), en parte porque no prestó atención a la **calidad de su producto de búsqueda** y en parte por el daño a la marca causado por unas medidas de seguridad deficientes, que provocaron la **mayor violación de datos de la historia**, exponiendo 3.000 millones de cuentas de usuarios en 2013. ¡Tal es el impacto de la calidad del software hoy en día!

Hay muchos ejemplos similares en todo el mundo que refuerzan la observación de que las empresas, a pesar de las novedosas ideas de producto que puedan tener, se enfrentan a una pendiente empinada y resbaladiza cuando se compromete la calidad, ya que los clientes se pasan rápidamente a competidores más fiables. A veces, las empresas pueden verse obligadas a elegir el tiempo de

comercialización en lugar de la calidad del software, pero deben ser conscientes de que sólo se han creado una deuda que deben resolver antes de que sus competidores la utilicen en su beneficio. Así pues, podemos afirmar con rotundidad que la calidad es la quintaesencia para sostener un negocio a largo plazo, y sólo se puede conseguir una alta calidad mediante una combinación de desarrollo hábil y pruebas meticulosas, prestando atención a cada aspecto de la aplicación en toda su pila. Para iniciarte en este camino, este capítulo te presentará lo que implica la comprobación de toda la pila de una aplicación web o móvil típica.

Pruebas completas para una alta calidad

Para empezar, unámonos en una comprensión común de la *calidad del software*. Antes, la calidad del software se equiparaba a una aplicación sin errores, pero cualquiera que trabaje hoy en el mundo del software estará de acuerdo en que ya no se trata sólo de eso. Si pides a los usuarios finales que definan la calidad, les oirás hablar de facilidad de uso, aspecto y tacto, privacidad de los datos, rapidez en la presentación de la información y disponibilidad de los servicios 24 horas al día, 7 días a la semana. Si pides a las empresas que definan la calidad, oirás hablar de rentabilidad de la inversión, análisis en tiempo real, tiempo de inactividad cero, no dependencia de un proveedor, infraestructura escalable, seguridad de los datos, cumplimiento de la legislación, y mucho más. Todos estos son aspectos de lo que hace que una aplicación sea hoy un software de alta calidad. Los fallos en cualquiera de estas áreas afectarán a la calidad de un modo u otro, ipor eso tenemos que comprobarlos meticulosamente!

Aunque la lista de requisitos de calidad parece alta, disponemos de herramientas y metodologías para satisfacer la mayoría de esas necesidades. Por tanto, el puente hacia la alta calidad está formado por el conocimiento de esas herramientas y, lo que es más

importante, por la habilidad para aplicarlas en un contexto determinado, tanto desde la perspectiva del desarrollo como de las pruebas. Este libro pretende ayudarte a construir ese puente, enseñándote las habilidades de prueba que necesitas para ofrecer aplicaciones web y móviles de alta calidad.

Probar, en pocas palabras, es una práctica para validar que el comportamiento de la aplicación es el esperado en todo momento. Para que las pruebas tengan éxito, deben practicarse a nivel micro y macro. Tiene que estar entrelazada con los aspectos granulares de la aplicación, como probar cada método de una clase, cada valor de datos de entrada, mensaje de registro, código de error, etc. Del mismo modo, tiene que centrarse en aspectos macro, como probar funciones, integraciones entre funciones y flujos de trabajo de extremo a extremo. Pero no podemos quedarnos ahí. Tenemos que seguir probando los aspectos holísticos de calidad de la aplicación - seguridad, rendimiento, accesibilidad, usabilidad, etc.- para lograr el objetivo final de ofrecer un software de alta calidad. Podemos resumir todo esto diciendo que tenemos que hacer *pruebas de pila completa*. Como se representa en [la Figura 1-1](#), las pruebas de pila completa implican probar diferentes aspectos de la calidad de la aplicación en cada capa (base de datos, servicios e interfaz de usuario), y la aplicación en su conjunto.

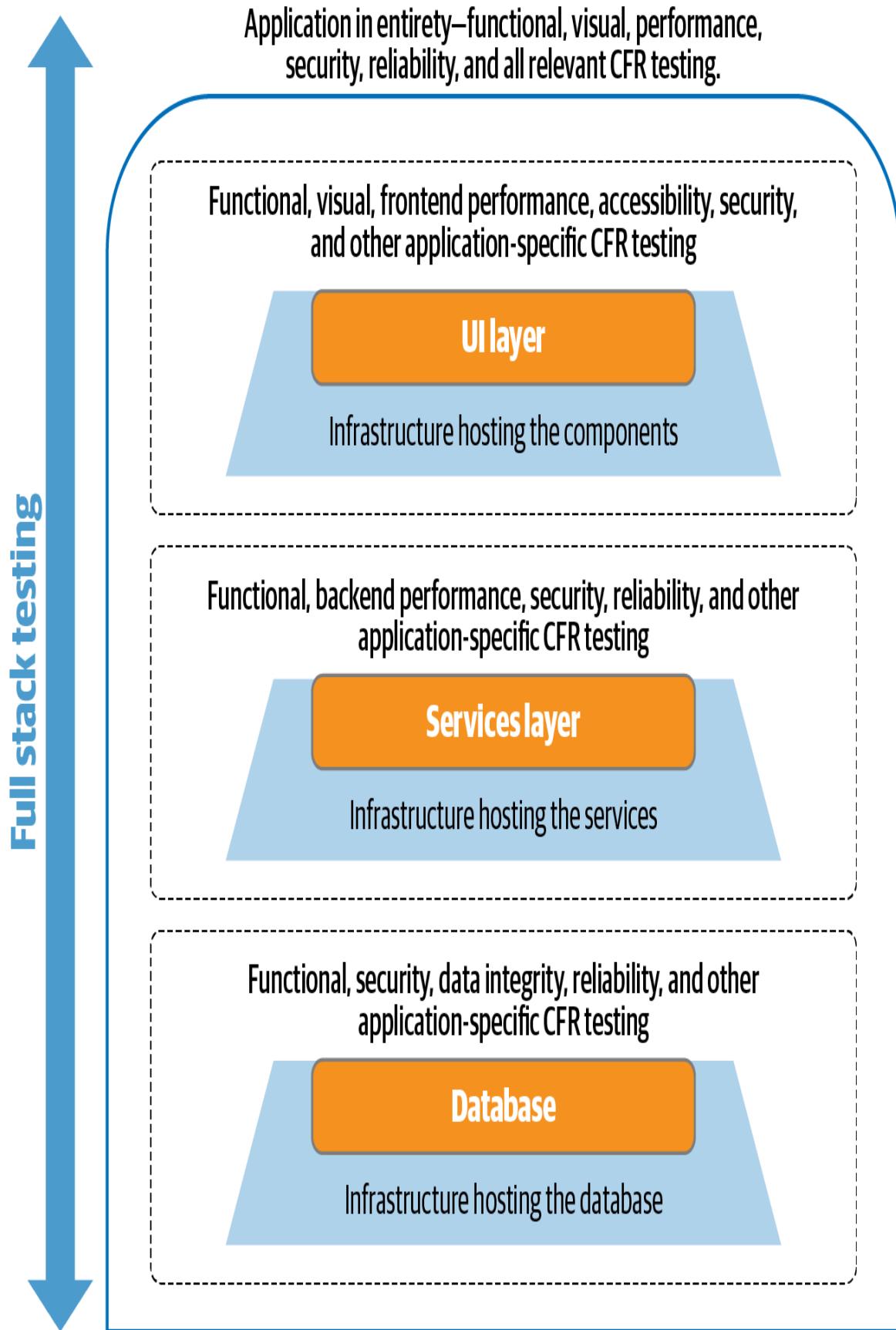


Figura 1-1. Una representación de las pruebas de pila completa

De hecho, las pruebas de pila completa y el desarrollo deberían ser inseparables, como los dos raíles de una vía férrea. Debemos avanzar por ambos raíles simultáneamente para incorporar calidad al producto; de lo contrario, tenemos garantizado el descarrilamiento. Por ejemplo, supongamos que estamos escribiendo un pequeño bloque de código para calcular el importe total del pedido de una aplicación de comercio electrónico. Tenemos que comprobar si el código está calculando la cantidad correcta y si es seguro en paralelo. Si no lo hacemos, podemos acabar teniendo lagunas en la línea ferroviaria, y si seguimos desarrollando sobre esta línea fracturada, tendremos una integración deficiente y una funcionalidad subóptima. Para arraigar las pruebas a un nivel tan elemental, los equipos tienen que dejar de pensar en ellas como una actividad aislada posterior al desarrollo, como se hacía tradicionalmente. Las pruebas de pila completa deben comenzar en paralelo con el desarrollo y practicarse a lo largo del ciclo de entrega, proporcionando una retroalimentación más rápida. La práctica de comenzar las pruebas en una fase temprana del ciclo de entrega se conoce como *pruebas por turnos*, y es un principio clave que hay que seguir para que las pruebas de pila completa den los resultados adecuados.

Prueba de desplazamiento a la izquierda

Si escribiéramos en la secuencia de actividades de un ciclo de vida de desarrollo de software tradicional, sería: *análisis de requisitos, diseño, desarrollo y pruebas*, con las pruebas al final. Como se ve en la Figura 1-2, la prueba por desplazamiento a la izquierda sugiere desplazar las actividades de prueba al principio del ciclo para producir resultados de alta calidad.

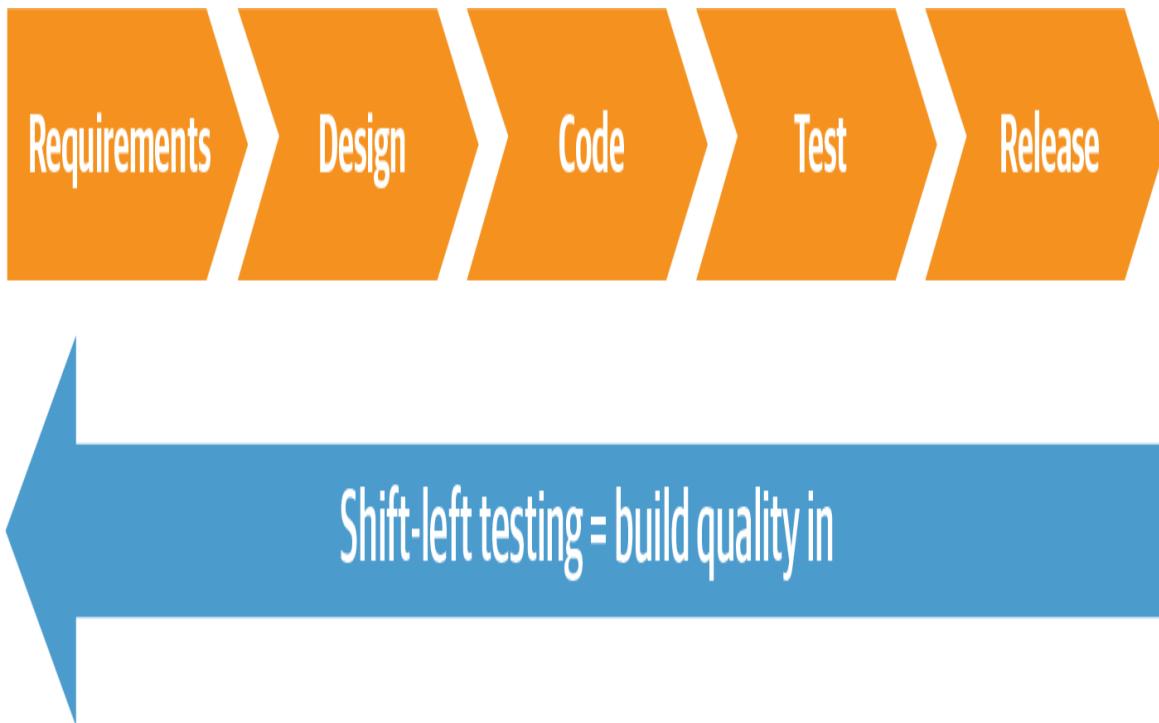


Figura 1-2. Prueba de cambio a la izquierda

Consideremos una analogía para ilustrar mejor este concepto. Imagina que tu equipo está construyendo una casa. ¿Te parece sensato terminar la construcción por completo y sólo entonces comprobar la calidad? ¿Y si descubres que las habitaciones no tienen las dimensiones correctas, o que las paredes interiores no son lo bastante resistentes para soportar la carga? Éstos son los tipos de problemas que las pruebas por turnos intentan evitar, aplicando controles de calidad desde la fase de planificación y continuándolos durante toda la fase de desarrollo. Esto permite que el producto final sea de la mayor calidad posible.

Continuar las comprobaciones de calidad a lo largo de la fase de desarrollo significa repetirlas de forma iterativa para cada pequeño trozo de trabajo, de modo que los cambios necesarios puedan incorporarse sin problemas. En la analogía de la construcción de una casa, significa realizar estas comprobaciones a medida que se construye cada pared, de modo que cualquier problema se corrija inmediatamente. Para realizar pruebas tan exhaustivas, las pruebas por turnos se basan en gran medida en pruebas automatizadas y

prácticas de CI/CD, en las que se automatizan las comprobaciones de calidad a nivel micro y macro, y se ejecutan continuamente contra cada pequeño trozo de trabajo en el servidor de CI. Esto garantiza que la aplicación se comprueba continuamente, con un coste y un esfuerzo mínimos en comparación con la comprobación manual de cada pequeño fragmento de trabajo para múltiples aspectos de calidad.

Para ver lo que esto significa en un contexto de software, desglosemos las pruebas por turnos en actividades cotidianas. Considera un equipo de software que sigue un ciclo de desarrollo iterativo, como en el desarrollo ágil. En la [Figura 1-3](#) se muestran algunas de las comprobaciones de calidad que pueden realizar en distintas fases de la entrega para desplazar las pruebas hacia la izquierda.

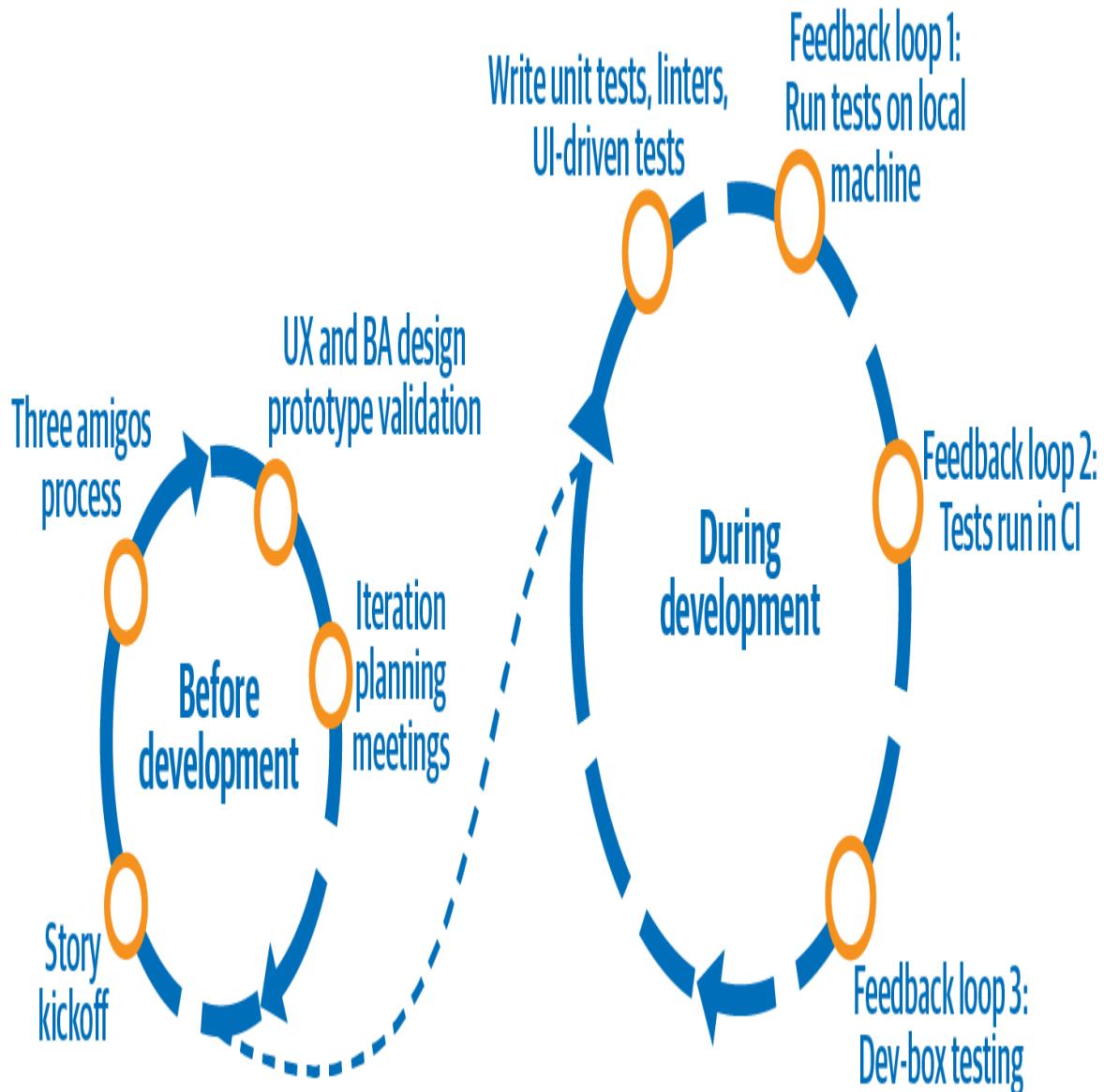


Figura 1-3. Un conjunto de controles de calidad desplazados a la izquierda

Leyendo **la Figura 1-3** desde la izquierda, comienza destacando una serie de comprobaciones de calidad que realiza el equipo antes de que una historia de usuario se considere lista para el desarrollo:

- En la fase de análisis se lleva a cabo una ceremonia denominada *proceso de los tres amigos*. En ella, los representantes de la empresa, los desarrolladores y los probadores se reúnen brevemente para reflexionar a fondo sobre la función. El objetivo del proceso es recoger las

perspectivas de las tres partes para que no se pasen por alto las integraciones, los casos de perímetro y otros requisitos empresariales. Este es el primer paso del cambio a la izquierda, en el que se validan los requisitos de una función para empezar.

- Paralelamente, el representante comercial del equipo trabaja con el diseñador de experiencia de usuario (UX) para validar y mejorar el diseño de la aplicación.
- Una vez completados estos dos pasos , se lleva a cabo una *reunión de planificación de la iteración* (IPM) al principio de la iteración/sprint para discutir en detalle las historias de usuario de esa iteración. Esto proporciona un espacio abierto para que el equipo valide colectivamente los requisitos una vez más.
- Durante la iteración, justo antes de que una historia de usuario sea seleccionada para su desarrollo, se celebra un *inicio de historia*. Se trata de una versión reducida del proceso de los tres amigos, en la que el debate se centra en los requisitos y los casos extremos de esa historia de usuario en concreto. En esta fase, podemos decir que el equipo ha comprobado/validado los requisitos con diligencia.

Del mismo modo, mientras se desarrolla una historia de usuario, se implantan y utilizan las siguientes comprobaciones de calidad para obtener un feedback rápido:

- Los desarrolladores escriben pruebas unitarias como parte de cada historia y las integran con CI. También añaden herramientas de linting y plug-ins para el análisis estático del código y los integran con CI para obtener retroalimentación continua.
- En algunos equipos, los desarrolladores también escriben las pruebas funcionales basadas en la interfaz de usuario como parte del desarrollo de las historias de usuario y las integran con

la IC. En otros equipos, los probadores las escriben después del desarrollo; ambas son prácticas habituales.

- Antes de confirmar los últimos cambios, los desarrolladores ejecutan un conjunto de pruebas automatizadas en sus máquinas locales para obtener el primer nivel de información.
- El segundo nivel de retroalimentación se obtiene del conjunto de pruebas automatizadas (unidad, servicio, interfaz de usuario, etc.) que se ejecutan durante la IC para cada confirmación.
- El tercer nivel de retroalimentación se recibe de un proceso llamado *dev-box testing*, en el que los probadores y los representantes de la empresa hacen una ronda rápida de pruebas exploratorias manuales en la máquina de un desarrollador para verificar rápidamente la funcionalidad recién desarrollada.

Con un enfoque tan riguroso en proporcionar una retroalimentación más rápida, el equipo obtendrá casi la mitad de la retroalimentación que, de otro modo, se habría obtenido mediante pruebas manuales posteriores al desarrollo, antes incluso de que la historia de usuario llegue a la fase de pruebas propiamente dicha. En otras palabras, el equipo acaba de desplazar las pruebas hacia la izquierda, y en el proceso da a los probadores del equipo la libertad de explorar a fondo la historia de usuario en busca de diversos aspectos de calidad, en lugar de limitarse a verificar los comportamientos funcionales esperados.

Así pues, las pruebas por turnos permiten evitar los defectos (al tener varias rondas de validación de los requisitos) y ayudan a detectar pronto cualquier defecto que aparezca, ya sea en la máquina de un desarrollador local o en CI. Además, garantiza la entrega de software de alta calidad, dando a los probadores el espacio necesario para explorar en profundidad diversos aspectos de la calidad.

NOTA

La Programación Extrema (XP) es un sabor del marco de desarrollo ágil de software que incorpora pruebas por turnos. Si quieres profundizar en las metodologías y prácticas de XP, te recomendamos que leas *Extreme Programming Explained* (Addison-Wesley Professional) de Kent Beck.

Este concepto de incorporar las pruebas en una fase más temprana del ciclo de entrega no se limita a las pruebas funcionales de las aplicaciones. Puede aplicarse a las pruebas en general, incluidas las pruebas de seguridad, las pruebas de rendimiento, etc. Por ejemplo, una de las muchas formas de desplazar las pruebas de seguridad hacia la izquierda es utilizar una herramienta de escaneado previo al commit como Talisman, que escanea el commit en busca de secretos y alertas incluso antes de comprobar el código. En cada uno de los próximos capítulos, verás enfoques prácticos de las pruebas de desplazamiento a la izquierda.

En general, este enfoque encarna el aforismo "La calidad es responsabilidad del equipo", ya que la realización de comprobaciones de calidad en cada fase del ciclo de vida del desarrollo de software - validación de prototipos de diseño de aplicaciones, requisitos, etc., como se ha comentado antes- debe ser responsabilidad de los distintos miembros del equipo. Por tanto, podemos decir que desarrollar las habilidades de comprobación pertinentes para realizar diversas comprobaciones de calidad es crucial para que todas las funciones de un equipo entreguen con éxito software de alta calidad.

Diez aptitudes para las pruebas Full Stack

Cuando pensamos en habilidades de comprobación, tendemos a consolidarlas en dos amplias habilidades: comprobación manual y automatizada. Pero la tecnología ha evolucionado en el transcurso de las últimas décadas, y estos términos generales enmascaran las

nuevas habilidades esenciales que hay que aprender para realizar diversas comprobaciones de calidad y ofrecer aplicaciones web y móviles de alta calidad. **La Figura 1-4** muestra las 10 habilidades de las pruebas de pila completa que nos permitirán realizarlas con eficacia.

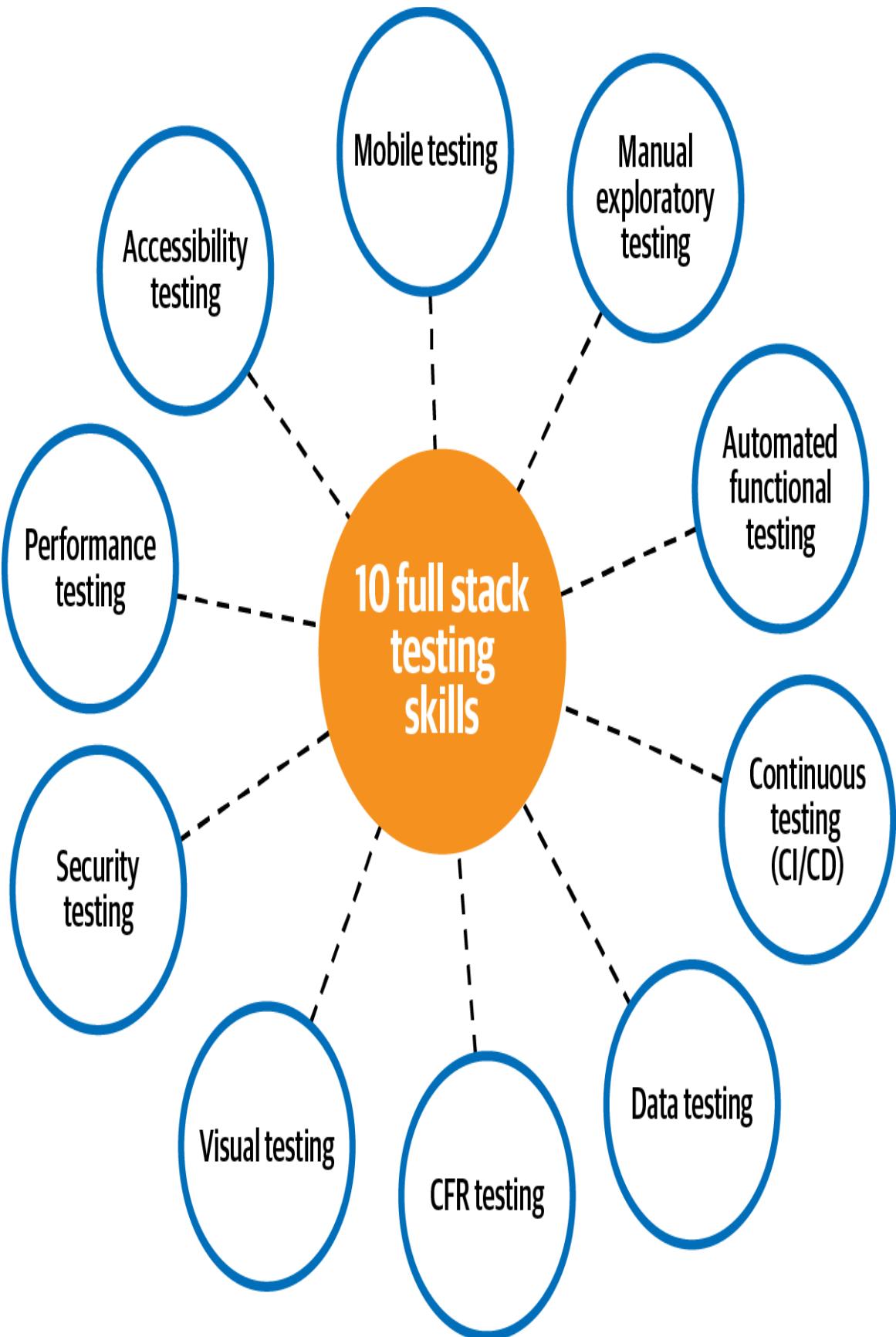


Figura 1-4. Diez habilidades de prueba de pila completa necesarias para ofrecer aplicaciones web y móviles de alta calidad

Exploraremos estas 10 habilidades y por qué deberías preocuparte por aprenderlas:

Pruebas exploratorias manuales

En primer lugar, para aclararlo, las pruebas manuales exploratorias son diferentes de las pruebas manuales. Estas últimas se refieren a la verificación de una lista dada de requisitos y no exigen necesariamente una mentalidad analítica. En cambio, las pruebas exploratorias manuales consisten en profundizar en los detalles de la aplicación, plantear distintos escenarios de la vida real aparte de lo documentado en las historias de usuario, simularlos en un entorno de pruebas y observar el comportamiento de la aplicación. Exige una mentalidad lógica y analítica, y es la primera y principal habilidad necesaria para crear una aplicación sin errores. Existen varias metodologías y enfoques que pueden aprenderse para estructurar estas sesiones de exploración, que trataremos en el **Capítulo 2**.

Pruebas funcionales automatizadas

Esta es una de las habilidades básicas de para las pruebas por turnos, como ya se ha comentado. Hacer pruebas automatizadas también reduce significativamente el esfuerzo de las pruebas manuales, sobre todo cuando la aplicación crece e incluye más funciones. En términos sencillos, la habilidad aquí consiste en escribir código para probar los requisitos de las funciones automáticamente, sin intervención humana. Para ello necesitamos herramientas y, por tanto, para adquirir esta habilidad es necesario conocer las distintas herramientas que pueden utilizarse para escribir pruebas en las distintas capas de la aplicación. Pero la cosa no acaba ahí; también hay que saber qué antipatrones hay que buscar en las pruebas automatizadas y

cómo mantenerse alejado de ellos. Trataremos los distintos aspectos de esta habilidad en [el Capítulo 3](#).

Pruebas continuas

La entrega continua es una práctica en la que las características se entregan de forma incremental a los usuarios finales en ciclos cortos, en lugar de a través de un único lanzamiento a lo grande. Mediante la entrega continua, la empresa obtiene beneficios antes y puede evaluar y reajustar rápidamente su estrategia de producto basándose en los comentarios de los usuarios finales. Para potenciar la entrega continua, tenemos que probar la aplicación continuamente para que siempre esté en un estado listo para ser lanzado. Por obvio que parezca, la forma inteligente de hacerlo es automatizar e integrar las comprobaciones de calidad en tus conductos CI/CD y ejecutarlas con frecuencia para facilitar el proceso de prueba. La habilidad de las pruebas continuas implica determinar qué tipos de pruebas automatizadas deben ejecutarse en cada fase del ciclo de entrega, de modo que el equipo pueda obtener información más rápidamente e integrarlas eficazmente en los conductos CI/CD. Estos aspectos esenciales se tratan en detalle en [el Capítulo 4](#).

Prueba de datos

Puede que hayas oído los refranes "Los datos son dinero" y "Los datos son el nuevo petróleo". Estas ideas ponen de relieve lo importante que es hoy en día comprobar la integridad de los datos en . Cuando se pierden los datos de los usuarios, o la aplicación muestra datos erróneos a los usuarios finales, éstos pierden la confianza en la propia aplicación. La habilidad de la comprobación de datos requiere conocimientos sobre los distintos tipos de sistemas de almacenamiento y procesamiento de datos que se suelen utilizar en las aplicaciones web y móviles (bases de datos, cachés, flujos de eventos, etc.) y la capacidad de derivar casos de prueba apropiados. En [el Capítulo 5](#) se tratan

estos temas, y cómo el flujo de datos entre los componentes de la aplicación crea nuevos casos de prueba aparte de los flujos funcionales.

Pruebas visuales

El aspecto y la sensación de la aplicación contribuyen en gran medida al valor de marca de la empresa, y especialmente cuando se trata de grandes productos de empresa a cliente (B2C) utilizados por millones de personas, una baja calidad visual puede afectar al valor de marca al instante. Por tanto, es esencial validar que los usuarios finales tengan una experiencia visual armoniosa y agradable realizando pruebas visuales de la aplicación. Las pruebas visuales requieren comprender cómo interactúan los componentes de la interfaz de usuario entre sí y con el navegador, en el caso de las aplicaciones web. Estas comprobaciones también pueden automatizarse, utilizando herramientas distintas de las utilizadas para las pruebas funcionales automatizadas. Hablaremos de esta habilidad, y de las marcadas diferencias entre estos dos tipos de automatización, en el [Capítulo 6](#).

Pruebas de seguridad

Las brechas de seguridad se han vuelto demasiado frecuentes en el mundo actual, y ni siquiera gigantes como Facebook y Twitter están excluidos de estos ataques. Los problemas de seguridad tienen un alto coste tanto para los usuarios finales como para la empresa en términos de pérdida o exposición de información sensible, sanciones legales y reputación de la marca. Hasta ahora, las pruebas de seguridad se han considerado una habilidad de nicho en el sector, y los probadores de penetración cualificados suelen contratarse sólo hacia el final del ciclo de desarrollo para buscar problemas de seguridad. Pero con la falta de talento profesional disponible para las pruebas de seguridad y la creciente incidencia de las brechas de seguridad, es

aconsejable que los equipos de software incorporen las pruebas de seguridad básicas como parte de su trabajo diario. En el **Capítulo 7** hablaremos de cómo pensar como un hacker y buscar problemas de seguridad en la funcionalidad de las aplicaciones, junto con herramientas para automatizar los análisis de seguridad.

Pruebas de rendimiento

Incluso un ligero descenso en el rendimiento de la aplicación puede acarrear enormes pérdidas financieras y de reputación para una empresa -recordemos el ejemplo de Flipkart comentado antes-. La habilidad de las pruebas de rendimiento implica medir un conjunto de indicadores clave de rendimiento en diferentes capas de la aplicación. Las pruebas de rendimiento también pueden automatizarse e integrarse con los procesos CI para obtener información continua. En el **Capítulo 8** hablaremos de una estrategia de pruebas de rendimiento por turnos, junto con las herramientas pertinentes.

Pruebas de accesibilidad

Las aplicaciones web y móviles se han convertido en productos cotidianos en . Hacerlas accesibles a las personas con discapacidad permanente o temporal no sólo es un mandato legal en muchos países, sino que éticamente es lo correcto. Para adquirir la habilidad de realizar pruebas de accesibilidad, primero debemos comprender las normas de accesibilidad exigidas por la ley. A continuación, podemos utilizar herramientas de auditoría de la accesibilidad, tanto manuales como automatizadas, para validar si se cumplen esas normas. Hablaremos de esta habilidad, y de por qué incorporar funciones de accesibilidad puede ser incluso una opción lucrativa para las empresas, en el **Capítulo 9**.

Pruebas de requisitos interfuncionales

Hemos visto que los usuarios finales y las empresas de tienen una larga lista de requisitos de calidad, como disponibilidad, escalabilidad, mantenibilidad, observabilidad, etc., aparte de necesitar simplemente una funcionalidad sin errores. Son los llamados *requisitos interfuncionales* (CFR) de una aplicación. Aunque los requisitos funcionales de suelen acaparar la mayor atención, son los CFR los que imbuyen calidad a la aplicación, y no probarlos provocará la insatisfacción de los equipos empresariales o de software, de los usuarios finales, o de ambos. Por tanto, la habilidad para probar los CFR es una habilidad de prueba fundamental. En [el Capítulo 10](#) hablaremos de las metodologías y herramientas de prueba para validar los distintos CFR.

NOTA

Muchos en la industria también se refieren a los CFR como *requisitos no funcionales* (NFR). Discutiremos las sutiles diferencias entre estos dos términos en [el Capítulo 10](#).

Pruebas móviles

El gran número de aplicaciones disponibles en en las principales tiendas de aplicaciones (Google Play y Apple App Store) en 2021 puede sorprender: un total de **5,7 millones**. La explosión del número de aplicaciones móviles se debe principalmente al mayor uso que hacemos de los dispositivos móviles. De hecho, la empresa de análisis web Global Stats anunció en 2016 que sus datos mostraban que el uso de Internet en móviles y tabletas en todo el mundo había **superado al de los ordenadores de sobremesa**. Por tanto, la capacidad de probar aplicaciones móviles y la compatibilidad de los sitios web en los dispositivos móviles es una habilidad fundamental hoy en día.

Aunque todas las habilidades mencionadas anteriormente son necesarias para probar aplicaciones móviles, también requiere un cambio de mentalidad. Además, hay que aprender todo un conjunto de herramientas de comprobación específicas para móviles con el fin de realizar diversas comprobaciones de calidad en las aplicaciones móviles. Por lo tanto, las pruebas para móviles se consideran aquí como una habilidad independiente. Recorreremos los matices del panorama móvil en [el Capítulo 11](#).

Juntas, estas 10 habilidades de pruebas de pila completa te permitirán probar todo el alcance de los aspectos de calidad holística de las aplicaciones web y móviles. Como ya se ha mencionado, es importante que cada función del equipo adquiera cierto grado de competencia en cada una de estas habilidades. El libro te mostrará cómo, habilidad por habilidad, con ejemplos prácticos.

Puntos clave

Éstos son los puntos clave de este capítulo:

- La calidad del software ya no puede equipararse sólo a una funcionalidad sin fallos. Una aplicación puede considerarse de calidad subóptima si sus dimensiones holísticas de calidad (seguridad, rendimiento, calidad visual, etc.) no están a la altura.
- Las pruebas de pila completa consisten en probar todas las dimensiones de calidad de una aplicación de forma holística en todas las capas, con lo que se consigue un software de alta calidad.
- Para que las pruebas de pila completa cumplan su objetivo de entregar software de alta calidad, los equipos deben desplazar las pruebas hacia la izquierda, de modo que comiencen en

paralelo con el análisis y continúen durante todo el ciclo de entrega.

- Las pruebas por turnos encarnan el aforismo "La calidad es responsabilidad del equipo", ya que exige que cada función del equipo asuma la responsabilidad de realizar determinadas comprobaciones de calidad en distintas fases de la entrega. Esto exige que todos los miembros del equipo se actualicen, adquiriendo las habilidades de comprobación pertinentes en distintos niveles de competencia.
- Las dos categorías monolíticas clásicas de habilidades de comprobación, manual y automatizada, enmascaran un vasto conjunto de nuevas habilidades de comprobación necesarias para realizar pruebas de pila completa con eficacia. En este capítulo se han presentado 10 habilidades de comprobación diferentes que son esenciales para ofrecer aplicaciones web y móviles de alta calidad hoy en día, y que exploraremos en el transcurso de los siguientes capítulos.

Capítulo 2. Pruebas exploratorias manuales

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

No todos los que vagan están perdidos.

-J.R.R. Tolkien

La prueba exploratoria manual es una actividad intensa en la que ejercitas la aplicación de prueba con el objetivo de explorar y comprender sus comportamientos en diversas situaciones que no están articuladas explícitamente en ninguna parte, ya sea en el documento de requisitos o en las historias de usuario . Como resultado de la exploración, a menudo se descubrirán nuevos flujos de usuario que no se habían previsto durante la fase de análisis o desarrollo, y fallos en los flujos de usuario existentes. Cuando se producen estos descubrimientos, es una alegría refrescante para la persona que los ha encontrado, iya que pone de manifiesto sus complejas habilidades analíticas y su aguda capacidad de observación!

Normalmente, las pruebas exploratorias manuales se llevan a cabo en un entorno de pruebas, donde se despliega toda la aplicación. Los probadores se toman la libertad de manipular a su antojo los distintos componentes de la aplicación, como la base de datos, los servicios o los procesos en segundo plano, para simular distintos escenarios en tiempo real y observar el comportamiento de la aplicación. Este estilo exploratorio de pruebas difiere de las pruebas manuales tradicionales, que se refieren a la tarea de ejecutar

manualmente un conjunto concreto de acciones descritas como criterios de aceptación en las historias de usuario o en el documento de requisitos y verificar si las expectativas declaradas se cumplen con éxito. En otras palabras, las pruebas manuales no ejercitan necesariamente ninguna capacidad analítica, mientras que las pruebas exploratorias ponen un campo verde ante los probadores, invitándoles a ir más allá de lo que está documentado, ie incluso más allá de lo que se *sabe* hasta ahora sobre la aplicación!

Dado el solapamiento entre las pruebas manuales y las pruebas exploratorias, algunos equipos, incluso hoy en día, subestiman el valor de las pruebas exploratorias. También suele existir la percepción de que la cantidad de análisis realizado como parte de la elaboración y el desarrollo de historias de usuario es suficiente para entrar en funcionamiento, especialmente cuando se complementa con pruebas automatizadas([el Capítulo 3](#) trata en detalle las pruebas automatizadas). Sin embargo, esta creencia pasa por alto el hecho de que el análisis realizado durante la creación de la historia de usuario suele ser principalmente desde el punto de vista de la empresa, y durante el desarrollo, los desarrolladores pueden centrarse en el alcance actual de la funcionalidad y limitar su pensamiento a esa pequeña parte. Esto deja un vacío obvio, en el que la aplicación no se explora desde la perspectiva del usuario final y con una perspectiva global en un entorno de implementación. Esa laguna puede dejar espacio para problemas de integración y flujos de usuario final perdidos, razón por la cual los equipos necesitan una fase de pruebas exploratorias manuales posteriores al desarrollo.

NOTA

Las pruebas exploratorias reúnen los tres ángulos -los requisitos de la empresa, los detalles de la implementación técnica y las necesidades del usuario final- y cuestionan todo lo que se considera *cierto* desde todos estos ángulos. Una buena práctica es considerar que una funcionalidad está completa sólo después de que los nuevos flujos de usuario y los casos de prueba descubiertos a través de las pruebas exploratorias estén también automatizados.

Puede que no sea necesario asignar a una persona independiente la realización de estas pruebas exploratorias posteriores al desarrollo, aunque ese enfoque podría dar mejores resultados debido a la acumulación de conocimientos sobre la aplicación y a que la tarea exige a alguien con agudas dotes de observación y análisis. Si el coste o la disponibilidad lo impiden, los miembros actuales del equipo deberían asumir la responsabilidad de realizar las pruebas exploratorias de forma rotatoria durante cada iteración. De hecho, el desarrollo de habilidades para las pruebas exploratorias puede ayudar a que cada función rinda mejor.

Si eres uno de esos miembros del equipo que busca desarrollar sus habilidades para las pruebas exploratorias, este capítulo es para ti. Hablaremos de los marcos existentes en la industria que pueden ayudar en las pruebas exploratorias, y de una estrategia para abordar esta tarea. Los ejercicios del capítulo se centran en la realización de pruebas exploratorias de interfaces de usuario web y API, concretamente. También examinaremos un conjunto de prácticas útiles para mantener la higiene del entorno de pruebas, ya que un entorno de pruebas totalmente implementado desempeña un papel fundamental en el éxito de las pruebas exploratorias manuales.

TÉRMINOS DE USO COMÚN

Los siguientes son algunos términos de uso común que verás en este capítulo:

- Una *característica* o *funcionalidad* es la forma en que la aplicación proporciona valor a sus usuarios finales. Por ejemplo, el inicio de sesión es una función que proporciona seguridad a los usuarios finales.
- Un *flujo de usuario* es un conjunto de acciones que el usuario final realiza en la aplicación para conseguir el valor proporcionado por la funcionalidad. Por ejemplo, para iniciar sesión, el usuario final tiene que introducir sus credenciales e iniciar sesión; éste es el flujo de usuario de inicio de sesión.
- Un *caso de prueba* es un conjunto de acciones de que validan que la funcionalidad funciona como se espera. Por ejemplo, introducir un nombre de usuario y una contraseña válidos y comprobar que el inicio de sesión se realiza correctamente es un caso de prueba. Del mismo modo, introducir un nombre de usuario no válido y verificar que aparece un mensaje de error también es un caso de prueba. El primero es un caso de prueba positivo, ya que permite al usuario final alcanzar con éxito el valor proporcionado por la funcionalidad, mientras que el segundo es un caso de prueba negativo, ya que no permite alcanzar el valor. Para explorar completamente una funcionalidad, hay que simular y observar tanto los casos de prueba positivos como los negativos.
- Un *caso de perímetro* es un caso de prueba negativo que ocurre muy raramente.

Bloques de construcción

Empecemos echando un vistazo a ocho marcos de pruebas exploratorias, con ejemplos prácticos de su uso. Después practicaremos la exploración de una funcionalidad.

Marcos de pruebas exploratorias

El objetivo de los marcos de pruebas exploratorias es ayudarnos a formar modelos mentales que puedan aplicarse intuitivamente a contextos relevantes de la aplicación. Pretenden reducir el alcance de las pruebas dando claridad y estructura a una parte de la funcionalidad. Por ejemplo, los campos de entrada numéricos son habituales en las aplicaciones. En lugar de probar aleatoriamente todos los valores numéricos posibles para probar dicho campo, los marcos nos prestan estructuras para compartimentar lógicamente las entradas en conjuntos de muestras. Del mismo modo, hay marcos que tratan de estructurar las reglas de negocio y, por tanto, nos ayudan a ver los distintos flujos de usuarios y casos de prueba. Vamos a sumergirnos en ellos uno a uno, con ejemplos.

Como primer ejemplo, tomemos una página web que pide los ingresos del usuario como entrada, como se ve en [la Figura 2-1](#), y muestra la cantidad de impuestos que debe el usuario como salida. [La Figura 2-1](#) también muestra los distintos tramos impositivos utilizados para el cálculo en la parte derecha.

Check your income tax!

Enter total
income:

Submit

(See the table on the right for more information on tax calculation.)

Tax details

Income	Tax
\$0 - \$5000	5%
\$5000 - \$15000	10%
>\$15000	30%

Figura 2-1. Un ejemplo sencillo de calculadora de impuestos

Para comprobar si la lógica de cálculo de impuestos funciona como se espera, tenemos que identificar casos de prueba positivos y negativos para probarlos como entradas. Un punto a tener en cuenta aquí es que los ingresos son un valor numérico continuo que va de 0 a infinito. Para llegar a nuestros casos de prueba positivos y negativos, tenemos que acotar lógicamente el conjunto adecuado de valores numéricos de entrada y verificar la salida. Hay dos marcos

que pueden ayudarnos a hacerlo: la partición de clases de equivalencia y el análisis de valores límite.

Partición de clases de equivalencia

El marco de partición de clases de equivalencia sugiere que dividamos las entradas que dan lugar a la misma salida o se someten a un procesamiento similar en clases de partición, y que basta con elegir una sola entrada de muestra de cada partición para probar la funcionalidad por completo.

Aplicando esta sugerencia al ejemplo de la calculadora de impuestos, el primer conjunto de clases de partición serán los propios tramos impositivos: [0 - 5000], [5001 - 15000] y [>15000]. Estas tres clases pueden considerarse *clases de equivalencia*, ya que cada entrada dentro de cada clase estará sujeta a las mismas reglas, y para validar los casos de prueba positivos basta con probar con tres puntos de datos de entrada, uno de cada clase. Por ejemplo, a menos que estés aburrido y quieras probar más, probar con las entradas 2.000, 10.000 y 20.000 será suficiente para validar los casos de prueba positivos. A continuación, se puede aplicar el mismo marco para derivar los casos de prueba negativos. Las clases de entradas que deben provocar un error son [*valores negativos*], [*/letras*], [*símbolos*], etc. De nuevo, un valor de cada clase es suficiente para probar los casos de prueba negativos.

Además de las pruebas exploratorias manuales, este marco es útil en las pruebas unitarias (tratadas en el [Capítulo 3](#)). También puede aplicarse a cualquier otro contexto relevante de la aplicación, como probar resultados temporales (antes y después de un evento), estados internos del sistema, etc.

Análisis del valor límite

El análisis del valor límite amplía el método de partición de clases de equivalencia comprobando explícitamente las condiciones límite en cada una de las clases. Esto es útil para encontrar errores, ya que

las condiciones límite suelen estar vagamente definidas y mal implementadas. Por ejemplo, en nuestro sencillo ejemplo de calculadora de impuestos, los requisitos para los tramos impositivos pueden haberse establecido como "5% de impuestos para ingresos inferiores a 5.000 \$, 10% para ingresos entre 5.000 \$ y 15.000 \$, y 30% de impuestos para ingresos superiores a 15.000 \$". Sin embargo, esto no define claramente las condiciones límite, es decir, en qué clases deben incluirse los valores 5000 y 15000. El marco de análisis del valor límite llama la atención sobre estas cuestiones y ayuda a resolverlas probando los valores límite en cada una de las clases de equivalencia, además de elegir una entrada dentro del rango de la clase.

Apliquemos este marco al ejemplo de la calculadora de impuestos analizando los valores límite de cada una de las clases de equivalencia que hemos encontrado antes. La primera clase, [0 - 5000] tiene 0 y 5.000 como valores límite. Pero, lógicamente, cuando los ingresos son 0, no debería haber impuestos. Así que se forman nuevas clases de equivalencia: [0] y [1 - 5000]. Por tanto, los valores límite que debemos comprobar para cubrir todos los casos de prueba positivos son [0, 1, 5000, 5001, 15000, 15001], como se ve en [la Figura 2-2](#).

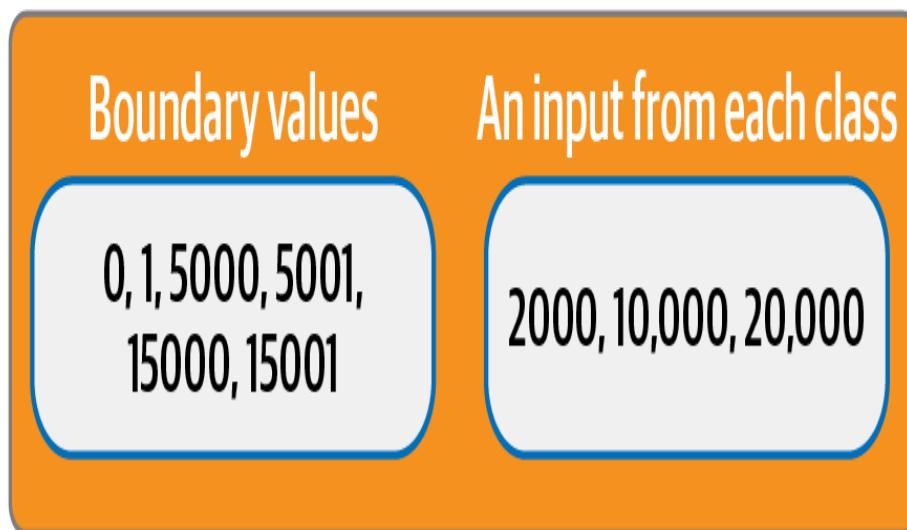
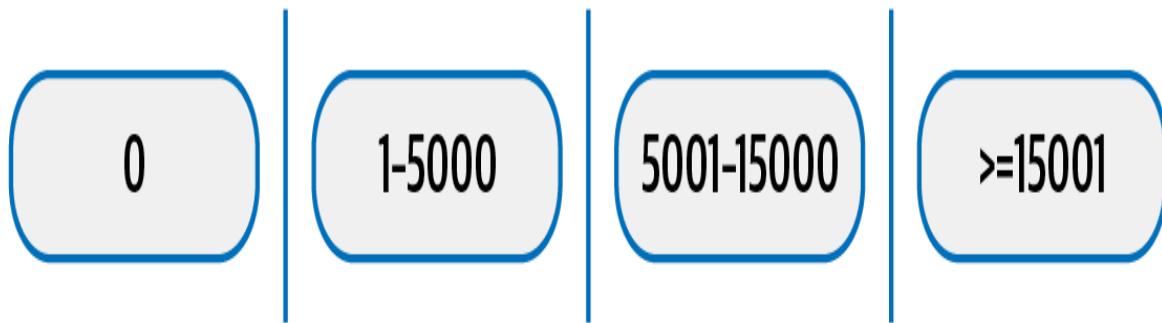


Figura 2-2. Clases de equivalencia con condiciones límite

Como ilustra este ejemplo, aunque se trata de un marco de pruebas, rinde los máximos beneficios para el equipo cuando se aplica en todas las fases de entrega, empezando por el análisis.

Una vez analizados dos marcos que ayudan a estructurar los valores de entrada de un solo campo y a traducirlos en un conjunto mínimo de casos de prueba positivos y negativos, es hora de pasar a los marcos que se ocupan de situaciones ligeramente más complejas en las que múltiples combinaciones de entrada dan lugar a diferentes salidas. Para ayudarnos en este debate, tomemos el ejemplo clásico de una página de inicio de sesión que toma dos entradas, una dirección de correo electrónico y una contraseña, y analicemos cómo los marcos de transición de estados, tabla de decisiones y causa-efecto ayudan a visualizar los distintos casos de prueba.

Transición de estado

El marco de transición de estados es útil para derivar casos de prueba en situaciones en las que el comportamiento de la aplicación cambia en función del historial de entradas. Por ejemplo, nuestra página de inicio de sesión podría mostrar un mensaje de error la primera y la segunda vez que el usuario introduce una contraseña incorrecta, pero la cuenta podría bloquearse la tercera vez. En tales escenarios podemos dibujar un árbol de transición, como se ve en la **Figura 2-3**, para derivar casos de prueba. En el árbol de transición, puedes observar que cada estado de la aplicación se representa como un nodo. Los posibles resultados de una acción se muestran como subnodos, y las acciones/eventos que desencadenan los resultados se indican como etiquetas de las ramas.

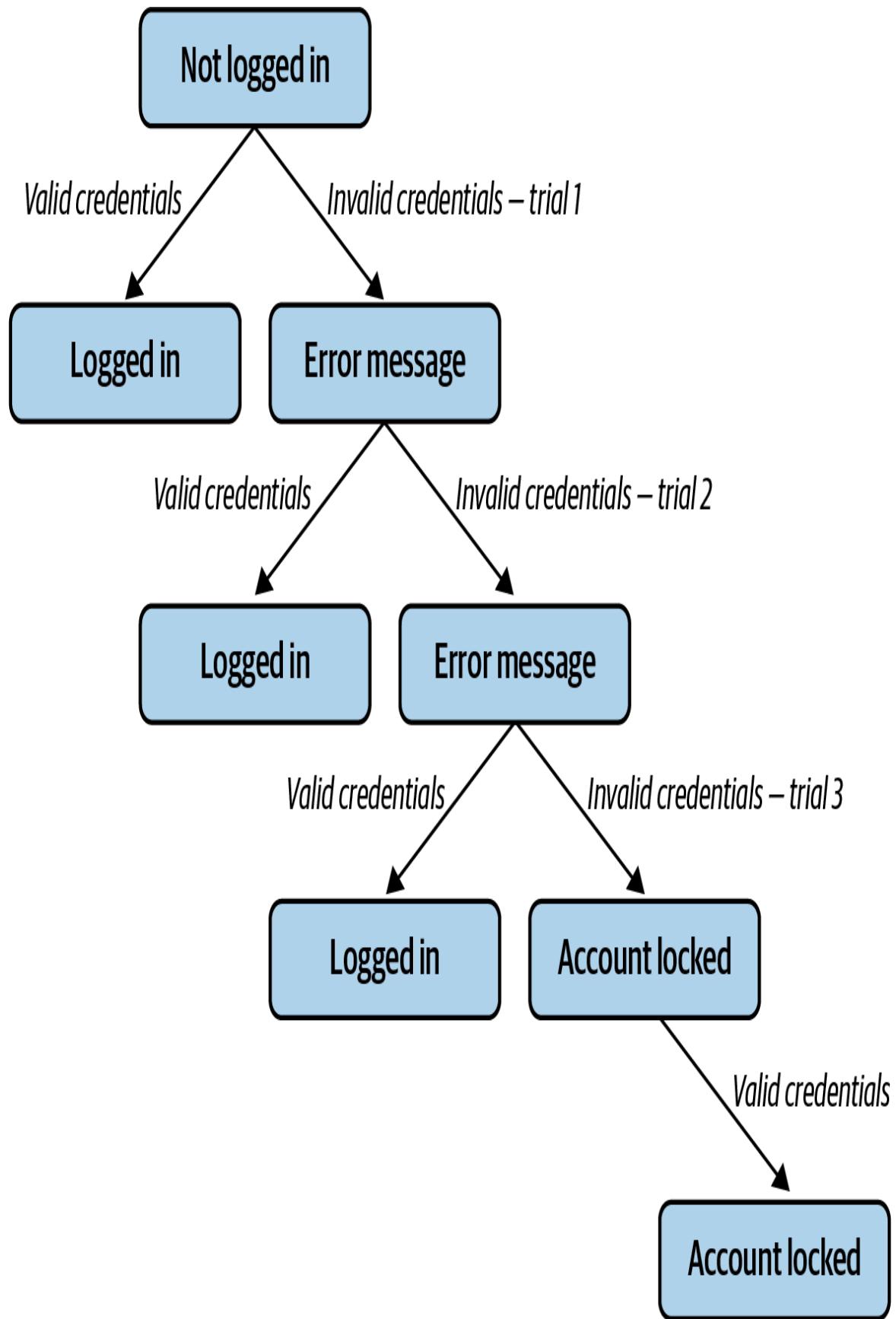


Figura 2-3. Árbol de transición de estados para el escenario de inicio de sesión no válido

Este árbol ofrece una imagen clara de cada caso de prueba con su estado inicial, la acción que cambia el estado de la aplicación y los resultados que se espera validar. La visualización también nos da una estimación realista de la cantidad de esfuerzo necesario para probar una característica al aclarar el número de estados y transiciones, lo que ayuda en la fase de planificación.

Las transiciones de estado pueden ser mucho más complicadas que esto, como en un sistema de gestión de pedidos en el que los pedidos pasan por estados como pago completado, pendiente, enviado, cancelado, cumplido, etc. En tales casos, visualizar cada estado como un nodo y las acciones que llevan el pedido a cada posible estado siguiente dará una visión clara de la propia función.

Tabla de decisiones

Cuando las entradas están ligadas lógicamente (AND, OR, etc.) para producir resultados, se pueden utilizar tablas de decisión para derivar casos de prueba. Esto puede ahorrar mucho tiempo durante las pruebas, ya que tienes todas las posibles combinaciones de entrada y los resultados esperados claramente marcados en la tabla de antemano. En el ejemplo del inicio de sesión, el correo electrónico y la contraseña están vinculados lógicamente por el operador AND; es decir, tanto el correo electrónico como la contraseña tienen que ser correctos para que el inicio de sesión se realice correctamente. **La Tabla 2-1** muestra la tabla de decisiones que podemos crear para este escenario.

Tabla 2-1. Tabla de decisiones para el escenario de inicio de sesión

Tabla de decisiones		Caso de prueba 1	Caso de prueba 2	Caso de prueba 3
Condiciones	Envía un correo electrónico a	Verdadero	Falso	Falso
	Contraseña	Falso	Verdadero	Falso
Acciones	Iniciar sesión	-	-	-
	Mensaje de error	Verdadero	Verdadero	Verdadero

El método también puede ahorrar tiempo al permitirnos eliminar ciertos casos de prueba innecesarios. Por ejemplo, en el escenario de inicio de sesión, el caso de prueba 3, en el que ambas entradas son incorrectas, puede eliminarse, ya que el inicio de sesión falla si una sola de las entradas es incorrecta.

Gráficos de causa-efecto

El gráfico causa-efecto es otra forma de visualizar las entradas vinculadas lógicamente y sus posibles resultados. Este marco ayuda a ver el panorama general de una función y, por tanto, es especialmente útil en la fase de análisis. Una vez creado el gráfico, puedes traducirlo en una tabla de decisiones para derivar casos de prueba detallados. **La Figura 2-4** muestra el diagrama causa-efecto del mismo ejemplo de inicio de sesión.

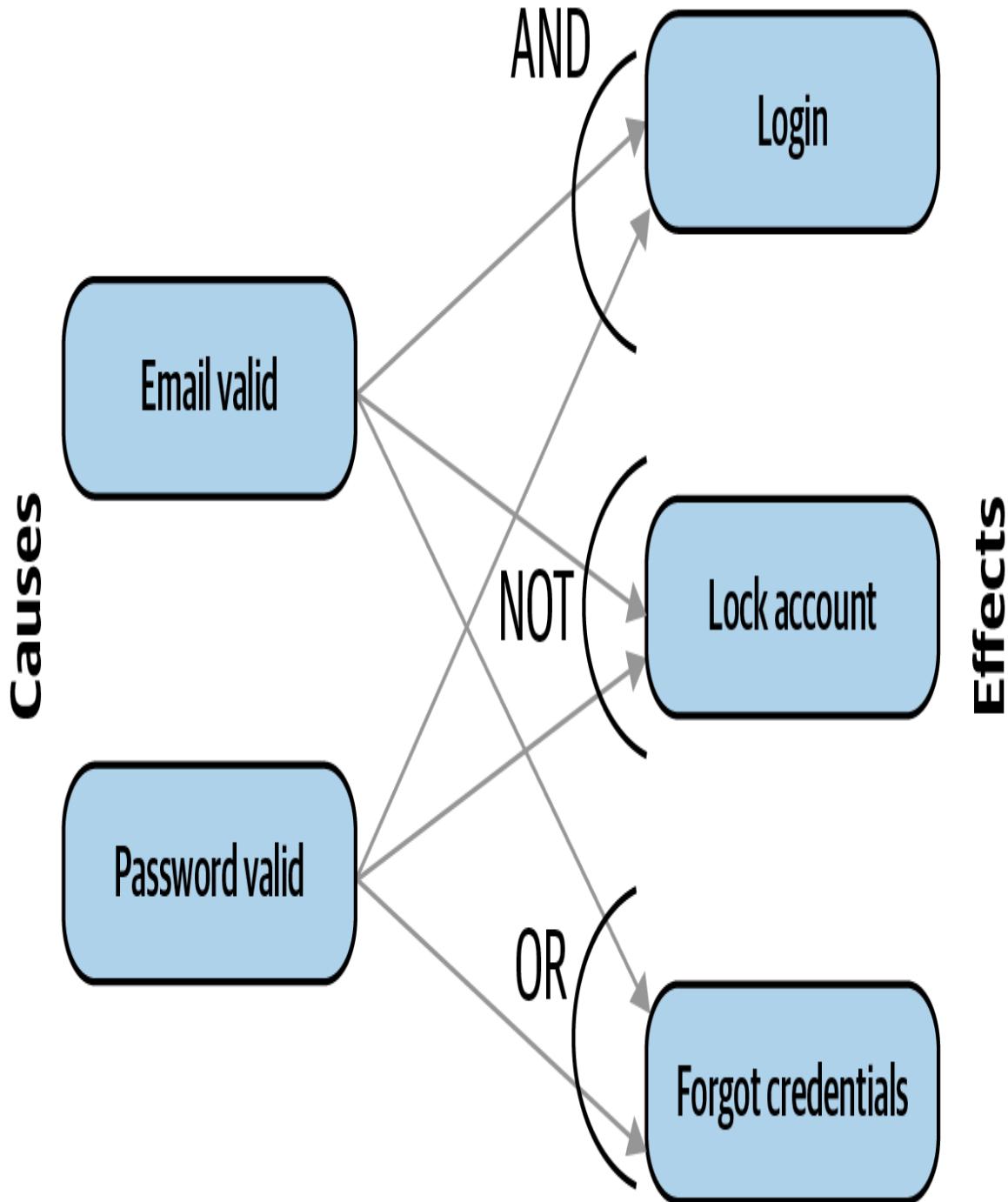


Figura 2-4. Gráfico causa-efecto para el escenario de inicio de sesión

Las causas se enumeran por un lado y los efectos por otro, y las rutas de navegación entre ellos se establecen con ayuda de operadores lógicos.

Los marcos que hemos visto hasta ahora son útiles para estructurar entradas relacionadas entre sí. A continuación, veremos dos marcos que pueden ayudarnos a tratar con múltiples variables independientes y grandes conjuntos de datos.

Pruebas por parejas

A menudo tenemos que tratar con más de un valor de entrada en las aplicaciones, y puede ser una lucha gestionar sus variaciones y derivar casos de prueba. Las pruebas por pares, también conocidas como pruebas de todos los pares, son un marco que ayuda a condensar los casos de prueba al mínimo cuando son varias las variables/entradas independientes que determinan los resultados. Hagamos un pequeño ejercicio para ilustrar cómo funciona.

Considera un formulario que toma tres entradas independientes: tipo de sistema operativo (SO), fabricante del dispositivo y resolución. El campo SO puede tomar dos valores: Android o Windows. El campo dispositivo puede tomar tres valores: Samsung, Google u Oppo. Por último, el campo resolución puede tomar como valores Pequeña, Mediana y Grande. Así, cuando estamos probando este formulario, tenemos $2 * 3 * 3 = 18$ combinaciones de entrada, como se ve en [la Tabla 2-2](#).

Tabla 2-2. Ejemplos de casos de prueba sin aplicar el método de prueba por parejas

Casos de prueba	Dispositivo	Resolución	OS
1	Samsung	Pequeño	Android
2	Samsung	Medio	Android
3	Samsung	Grande	Android
4	Google	Pequeño	Android
5	Google	Medio	Android
6	Google	Grande	Android
7	Oppo	Pequeño	Android
8	Oppo	Medio	Android
9	Oppo	Grande	Android
10	Samsung	Pequeño	Windows
11	Samsung	Medio	Windows
12	Samsung	Grande	Windows
13	Google	Pequeño	Windows
14	Google	Medio	Windows
15	Google	Grande	Windows
16	Oppo	Pequeño	Windows
17	Oppo	Medio	Windows
18	Oppo	Grande	Windows

Las pruebas por pares sugieren que basta con probar una vez cualquier par de entradas, ya que son variables independientes. Esto reducirá nuestra lista de casos de prueba a sólo nueve, como se ve en la Tabla 2-3.

Tabla 2-3. Casos de prueba reducidos cuando aplicamos el método de prueba por parejas

Casos de prueba	Dispositivo	Resolución	OS
1	Oppo	Pequeño	Android
2	Samsung	Pequeño	Windows
3	Google	Pequeño	Android
4	Oppo	Medio	Windows
5	Samsung	Medio	Android
6	Google	Medio	Windows
7	Oppo	Grande	Android
8	Samsung	Grande	Windows
9	Google	Grande	Android/Windows

La nueva tabla condensada ha reducido la repetición de varios pares. Por ejemplo, los pares [Google, Medium] y [Google, Windows] ahora sólo aparecen una vez cada uno.

Muestreo

Hasta ahora, hemos tratado con entradas que son pequeñas y consumibles por el cerebro humano sin ayuda de herramientas. Pero, ¿y si tenemos que probar grandes conjuntos de datos? Por ejemplo, supongamos que un sistema de seguros heredado se ha migrado a un nuevo sistema, y tenemos que comprobar si los datos de los seguros existentes se han transferido correctamente al nuevo sistema. Podría haber millones de usuarios en el sistema heredado, y no podemos aplicar los marcos discutidos hasta ahora para derivar casos de prueba. Por ejemplo, no podemos determinar clases de equivalencia, ya que cada usuario tendrá sus propias variaciones en cuanto a edad, primas, duración del contrato, tipo de régimen, etc., y no podemos aplicar pruebas por pares, ya que hay demasiadas

variables para identificar y eliminar los pares recurrentes. En estos casos, el muestreo es una técnica útil.

El muestreo, en general, puede aplicarse a cualquier entrada que sea continua y de gran tamaño. Consiste en seleccionar un subconjunto de los valores que se utilizarán para las pruebas, como se ve en la **Figura 2-5**, normalmente mediante una de las siguientes técnicas: muestreo aleatorio o muestreo por criterios específicos.

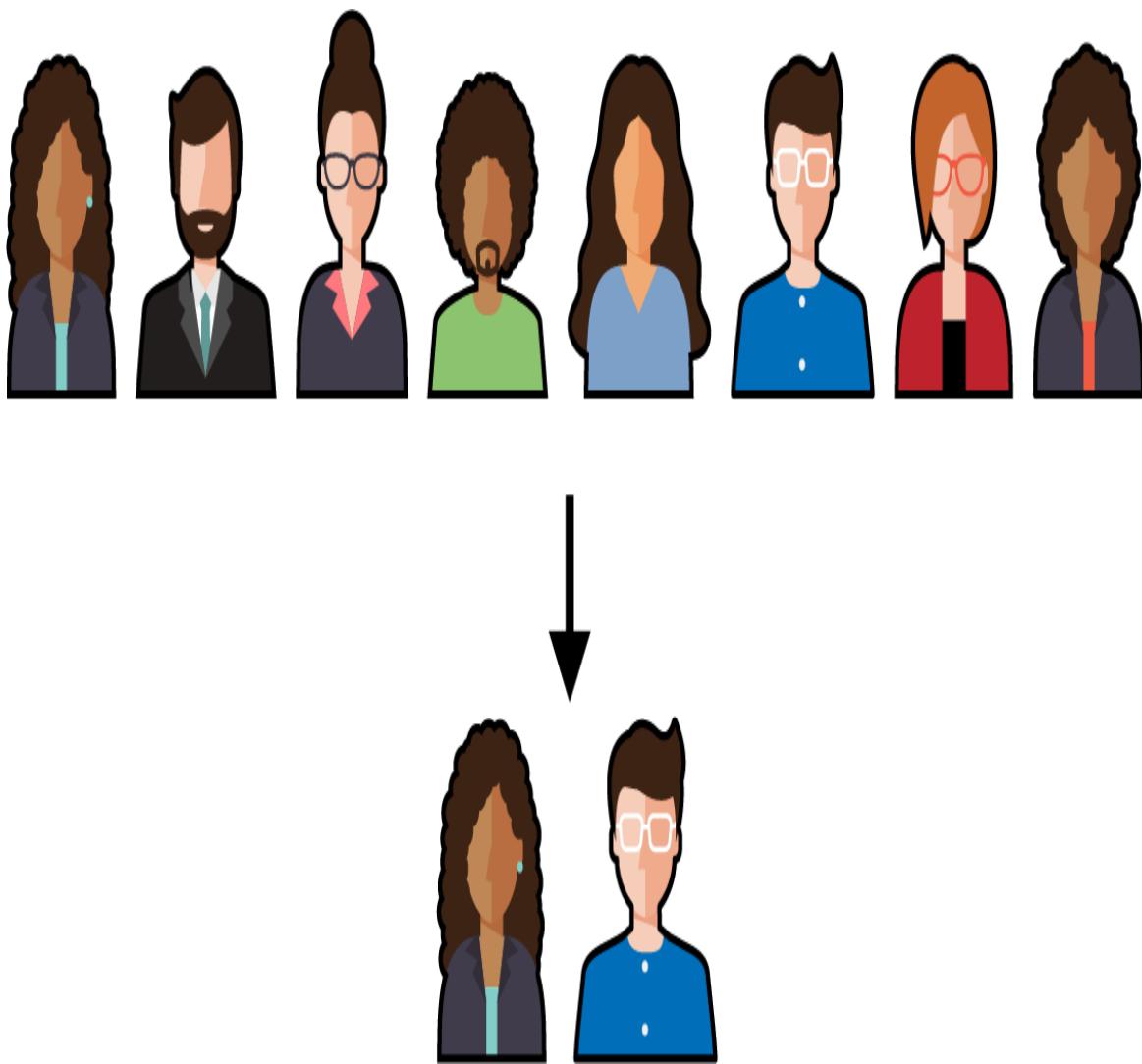


Figura 2-5. Muestreo basado en criterios o muestreo aleatorio

El muestreo aleatorio es cuando elegimos cualquier muestra de datos del conjunto de datos y verificamos los resultados. Por ejemplo, si hay 1.000 usuarios, podemos elegir 50-100 usuarios al

azar del sistema heredado y comparar sus datos en ese sistema con los datos almacenados en el nuevo sistema. Muestreo por criterios específicos es cuando elegimos las muestras identificando algunas características comunes en el conjunto de datos. Por ejemplo, en el sistema de seguros, podríamos hacer un muestreo basado en criterios específicos del usuario, como la edad, la duración del contrato (número de años suscritos), el modo de pago, la profesión, etc., y en criterios específicos de la póliza de seguros, como los intervalos de pago, el precio, etc. Podríamos perfeccionar aún más la técnica haciendo que el recuento de muestras para cada criterio fuera proporcional a la distribución real de los valores en el conjunto de datos. Esto formaría un minijuego de datos representativo y probablemente cubriría todo tipo de casos de prueba.

Eso nos lleva al último marco, que consiste más en ejercitarse en nuestra capacidad de pensamiento analítico y lógico que en un conjunto de directrices fijas. Vamos a ello ahora.

Método de adivinación de errores

La adivinación de errores consiste en predecir posibles fallos basándose en experiencias pasadas. Pueden ser problemas comunes con la integración, la validación de entradas, casos límite, etc. Aunque la experiencia pasada desempeña un papel fundamental en la predicción de casos de error probables, también puedes utilizar tu comprensión de la tecnología y el razonamiento lógico. De hecho, fomentar este tipo de pensamiento potencia tus habilidades de pruebas exploratorias en general.

He aquí algunos tipos de errores que aparecen con regularidad, según mi experiencia:

- Faltan validaciones para valores de entrada no válidos/en blanco y faltan mensajes de error apropiados que indiquen al usuario que corrija la entrada

- Códigos de estado HTTP poco claros devueltos por errores de validación de datos, técnicos y de negocio (echaremos un vistazo a algunos de ellos en "["Pruebas de la API"](#)")
- Condiciones de contorno no manejadas específicas del dominio, tipos de datos, estados, etc.
- Errores técnicos como la caída del servidor, la caducidad de las respuestas, etc. no gestionados en la interfaz de usuario.
- Problemas de interfaz de usuario (como tirones y residuos) durante las transiciones, la actualización de datos y la navegación
- Las palabras clave SQL *like* e *equals* se utilizan indistintamente, lo que cambia por completo los resultados
- Cachés sin borrar y tiempos de espera de sesión indefinidos
- Reposicionar una solicitud cuando el usuario pulsa el botón Atrás en el navegador
- Falta validación de formato de archivo al subir archivos desde plataformas de SO diferentes

Puedes utilizar este paquete de ocho marcos de pruebas exploratorias para estructurar tu proceso de pensamiento en torno a la exploración de una funcionalidad y la derivación de casos de prueba significativos. Ten en cuenta que estos marcos pueden aplicarse a cualquier contexto relevante de la aplicación, y no necesariamente sólo a los datos de entrada. Ahora que estás equipado con estas herramientas, antes de que lleguemos a algunos ejercicios prácticos, echemos un vistazo más de cerca a lo que implica explorar una funcionalidad.

Explorar una funcionalidad

Supón que te piden que realices pruebas exploratorias sobre la funcionalidad de creación de pedidos de una aplicación de comercio electrónico. ¿Por qué caminos de exploración deberías empezar? Esta sección responde a esa pregunta arrojando luz sobre cuatro caminos esenciales que deben explorarse en cualquier aplicación, como se ilustra en la [Figura 2-6](#).

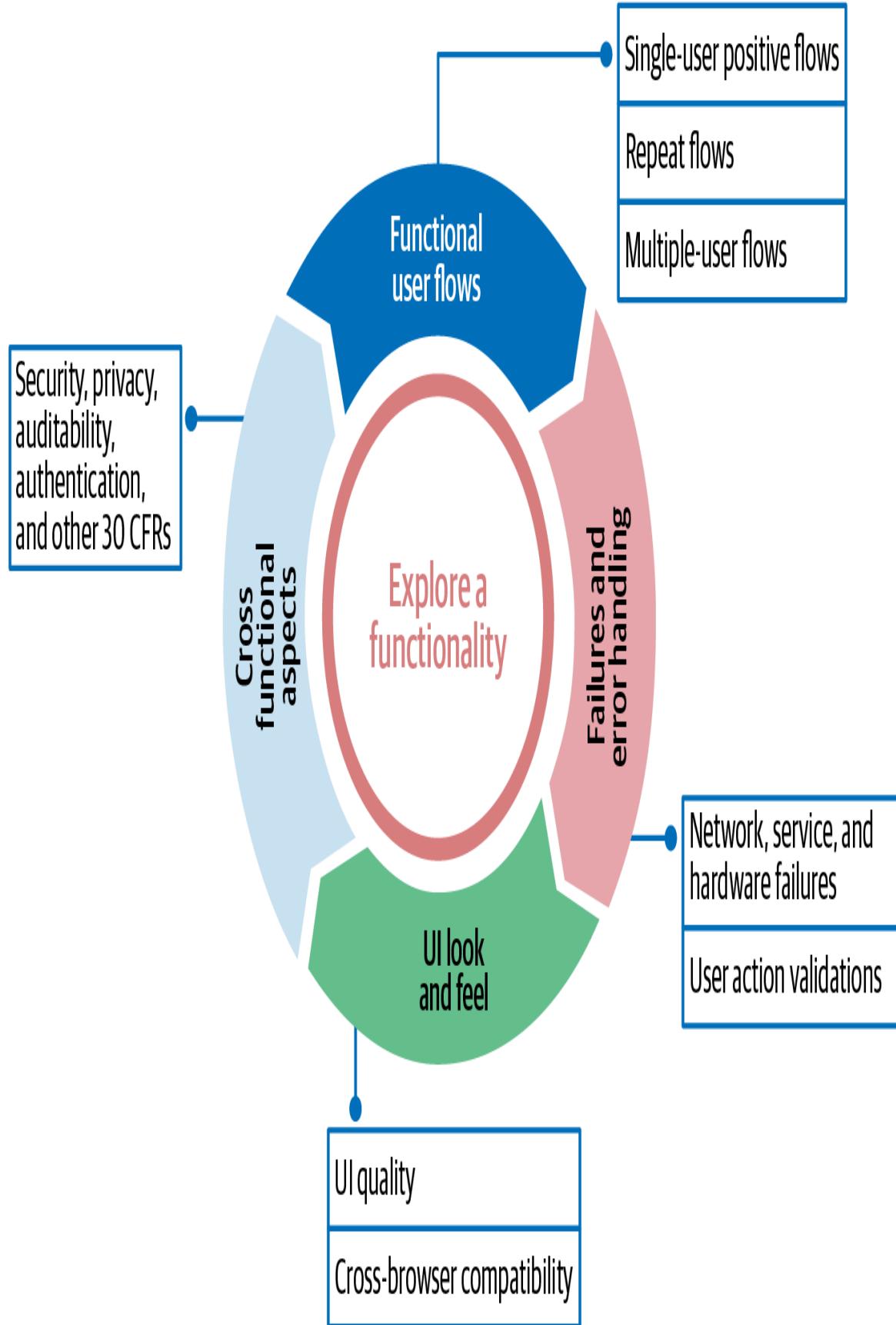


Figura 2-6. Cuatro vías de descubrimiento esenciales al explorar una funcionalidad

Flujos de usuarios funcionales

Los flujos funcionales de usuario de una aplicación se refieren a los recorridos que realiza un usuario final al utilizar la aplicación, como iniciar sesión, buscar un producto, añadirlo a la cesta de la compra, proporcionar una dirección de envío, elegir una opción de entrega, pagar el pedido y, por último, obtener una confirmación del pedido. Se trata de un flujo positivo monousuario, que es lo que debes validar en primer lugar. Tendrás que explorar el flujo positivo con diferentes direcciones de envío, métodos de pago y entrega, y combinaciones de artículos para asegurarte de que funciona por completo.

Mientras exploras, puede que te encuentres utilizando algunos de los marcos de pruebas exploratorias comentados anteriormente. Por ejemplo, puedes utilizar los métodos de partición de clases de equivalencia y de análisis del valor límite para verificar si la aplicación añade la cantidad correcta de impuestos al precio total. Del mismo modo, puedes utilizar un árbol de transición para derivar casos de prueba en torno a la dirección de envío y los métodos de entrega disponibles para esa combinación de direcciones. Una vez que te asegures de que este flujo positivo monousuario funciona perfectamente, deberías empezar a explorar los otros dos tipos de flujos:

Repetir flujos

Los usuarios finales suelen repetir el mismo flujo (o partes de él) varias veces en la aplicación, como buscar distintos productos y añadirlos al carrito de la compra. Pero lo habitual es que un flujo de usuario se pruebe una vez y, si funciona, se ignoren los flujos repetidos bajo el supuesto de que el comportamiento seguirá siendo el mismo. En la práctica, sin embargo, esta suposición puede ser cierta o no. Por ejemplo, si el usuario intenta volver a añadir el mismo artículo a su cesta de la compra, la interfaz de

usuario puede mostrar un mensaje diciendo que el producto ya se ha añadido y comprobando si desea aumentar la cantidad. También deben probarse los flujos de repetición.

Flujos multiusuario

Una funcionalidad puede funcionar perfectamente desde el punto de vista de un solo usuario, pero puede no ser así si varios usuarios interactúan con la aplicación simultáneamente en tiempo real. Por eso es importante explorar posibles escenarios de colisión, en los que las acciones de un usuario repercutan en otro. Por ejemplo, ¿qué ocurre cuando dos usuarios diferentes añaden el último producto disponible a sus carros de la compra en el mismo instante?

En resumen, los flujos funcionales de usuario se eligen a menudo como la primera ruta de descubrimiento que hay que explorar en una aplicación, pero dentro de esa ruta, puede haber varias subramas que explorar. Algunas subderivaciones esenciales son los flujos positivos de usuario único, los flujos repetidos y los flujos de usuarios múltiples.

Fallos y tratamiento de errores

Como se dijo al principio del capítulo, la prueba exploratoria se lleva a cabo en un entorno de prueba e implica entrometerse con los componentes de la aplicación para simular escenarios en tiempo real y observar el comportamiento de la aplicación. Hay dos frases en esa afirmación que requieren tu atención, ya que forman el núcleo de las pruebas exploratorias: la *intromisión en la aplicación* y *los escenarios en tiempo real*. Al considerar los escenarios en tiempo real, también debes pensar en todos los fallos posibles, ya que los fallos son prácticamente inevitables. Por ejemplo, podría producirse un fallo de red entre los componentes de la aplicación, impidiendo que envíe una respuesta al usuario, o la red podría ser lenta entre el usuario final y el servidor de la aplicación, provocando retrasos, o los

servicios de la aplicación podrían estar caídos debido a un fallo de hardware. Todos estos fallos y otros más deben preverse durante las pruebas exploratorias y simularse en el entorno de pruebas.

Además de los fallos de red, servicio y hardware antes mencionados, también puede haber errores debidos a acciones no válidas del usuario. Una funcionalidad sólo puede considerarse completa si tiene validaciones incorporadas para tratar estos casos. En la funcionalidad de creación de pedidos, varias instancias requieren explorar estas validaciones. Por ejemplo, como ya se ha comentado, la página de inicio de sesión debe tener validaciones sobre el correo electrónico y la contraseña, el texto de búsqueda introducido para buscar un producto debe validarse para evitar entradas no válidas, la disponibilidad del artículo, etc. Los otros lugares son la dirección de envío y los detalles de pago, la adición de artículos al carrito de la compra, etc.

Las pruebas exploratorias deben hacer gran hincapié en la identificación de posibles fallos y el tratamiento de errores. Como parte de la gestión de errores, la funcionalidad debe avisar a los usuarios de sus errores y sugerir posibles soluciones mediante un texto de error significativo.

El aspecto de la interfaz de usuario

La IU es lo que ve el usuario final, y no puede haber problemas evidentes con su calidad. Por tanto, su aspecto es otra importante vía de descubrimiento que hay que explorar. Por poner sólo algunos ejemplos, los casos de prueba relacionados con la calidad de la interfaz de usuario en la función de creación de pedidos podrían incluir asegurarse de que se proporciona una cantidad adecuada de espacio para las direcciones de envío (no tan poco que no haya espacio suficiente para mostrar un nombre de calle largo, ni tanto que haya una enorme cantidad de espacio en blanco cuando parte de la dirección sea corta) y que las imágenes de los productos se muestran con la calidad adecuada. Los usuarios finales deben poder

utilizar la aplicación sin problemas desde sus navegadores preferidos, y debe haber un ícono de carga cuando se produzcan retrasos. En el [Capítulo 6](#) se analiza en detalle un enfoque estructurado de las pruebas de calidad de la interfaz de usuario.

Aspectos transversales

Puede haber varios aspectos interfuncionales en una funcionalidad determinada, como la seguridad, el rendimiento, la accesibilidad, la autenticación, la autorización, la auditabilidad, la privacidad, etc., que requieren una atención específica durante las pruebas exploratorias. Muchos de estos aspectos tienen un capítulo entero de este libro dedicado a ellos debido a su importancia. Brevemente, he aquí algunos requisitos interfuncionales que nos gustaría examinar desde el punto de vista de la exploración de la funcionalidad de creación de pedidos:

Seguridad

En el flujo de usuario de creación de pedidos , un usuario abusivo podría introducir consultas SQL en los campos de entrada de la interfaz de usuario e intentar piratear la aplicación. La aplicación debería disponer de validaciones para hacer frente a tales intentos. Del mismo modo, los datos de la tarjeta de crédito de los usuarios no deben almacenarse en texto sin formato en la base de datos de la aplicación y no deben registrarse en texto sin formato en los registros de la aplicación, para que estén seguros incluso en caso de una posible violación. Todos estos aspectos de las pruebas de seguridad y otros más se tratan en detalle en [el Capítulo 7](#).

Privacidad

Los datos privados de los usuarios, como los detalles de la tarjeta de crédito y las direcciones de envío, no deben almacenarse en la base de datos de la aplicación sin su consentimiento. Además, los usuarios deben ser informados de antemano de las formas en

que sus datos pueden utilizarse para análisis o de si se enviarán a servicios de terceros para su procesamiento. Varias cláusulas de privacidad de los datos también están impuestas por normativas legales; trataremos estos temas en [el Capítulo 10](#).

Autenticación/autorización

La mayoría de los sitios web tienen funciones de autenticación de usuarios , lo que exige explorar casos de prueba relacionados con la autenticación, como el inicio de sesión único, la autenticación de dos factores, la caducidad de la sesión, el bloqueo de la cuenta, el desbloqueo, etc. En la aplicación de comercio electrónico, los usuarios finales pueden ver el catálogo de productos sin iniciar sesión, pero no hacer un pedido.

Del mismo modo, podría haber roles (p. ej., administrador, ejecutivo de clientes) y permisos (p. ej., editar un pedido) asignados a distintos usuarios, lo que requiere explorar casos de prueba relacionados con la autorización, como la anulación múltiple de roles, la adición de nuevos permisos a roles existentes, la observación del comportamiento de la aplicación cuando se ejecuta una operación sin los permisos adecuados, etc.

De nuevo, estos son sólo algunos ejemplos; en [el Capítulo 10](#) se tratan unos 30 aspectos interfuncionales diferentes y las formas de probarlos.

Las cuatro vías de descubrimiento descritas aquí deberían conducir a una cobertura de pruebas bien redondeada sobre cualquier funcionalidad dada. Ten en cuenta que, mientras exploras cada uno de estos caminos, es posible que se te ocurran nuevas ideas y casos de prueba que no pertenezcan a ese camino. Es importante anotarlas para que puedas volver a ellas más tarde, o utilizarlas para iniciar la exploración de otra ruta.

Estrategia de pruebas exploratorias manuales

La estrategia de pruebas exploratorias manuales representada en [la Figura 2-7](#) une todo lo que hemos discutido hasta ahora, y además agrupa los procesos del equipo. Esto te proporcionará un esquema práctico para realizar pruebas exploratorias en tu trabajo diario de proyecto. Empecemos por el semicírculo exterior y avancemos hacia el interior.

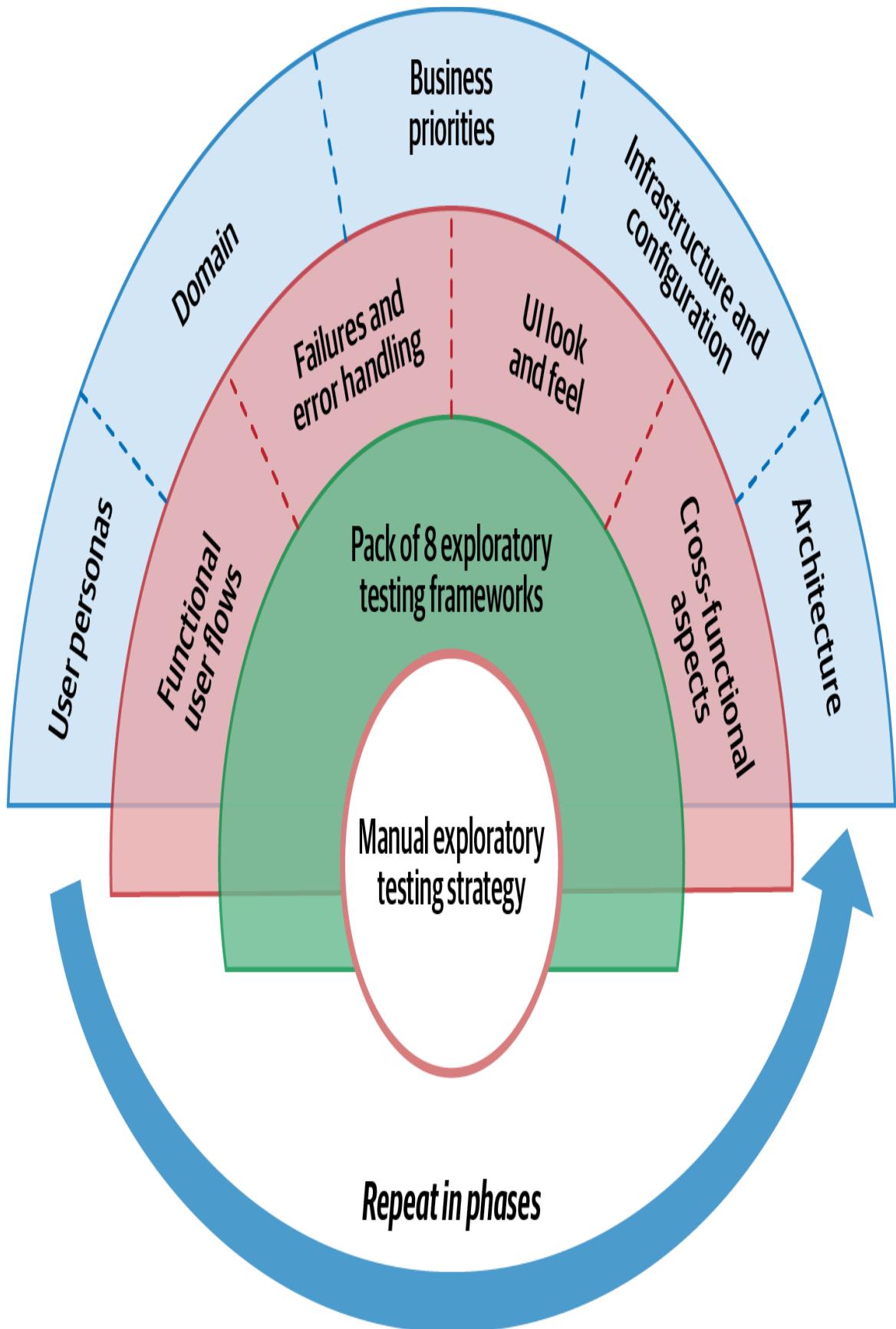


Figura 2-7. La estrategia de pruebas exploratorias manuales

Comprender la aplicación

El semicírculo exterior hace hincapié en comprender los detalles de la aplicación y señala cinco grandes áreas de la aplicación en las que debes centrarte. Recopilar detalles sobre éstas te ayudará a empezar con las pruebas exploratorias, pero, como ya se ha dicho, seguro que encontrarás nueva información sobre la aplicación durante la exploración.

NOTA

A veces, la prueba exploratoria se confunde en con *la prueba del mono*, un enfoque en el que la aplicación se prueba con entradas aleatorias y con un conocimiento nulo de la funcionalidad. Es importante señalar que en las pruebas exploratorias debes tener un conocimiento preciso de la funcionalidad que se está probando, y probar con la mentalidad de explorar las incógnitas.

Aquí tienes un resumen de las cinco grandes áreas en las que puedes centrarte mientras intentas comprender la aplicación:

Personas usuarias

Una **persona** es un personaje que representa a un conjunto de usuarios finales con atributos similares a . En los equipos de software, estos personajes de usuario se crean al principio del proyecto, para que sus necesidades específicas puedan impregnarse en todas las fases del ciclo de vida de la entrega, empezando por el diseño. Un ejemplo de personas usuarias que influyen en las características de una aplicación es un sitio de redes sociales, donde los adultos jóvenes pueden esperar una experiencia extravagante, mientras que los mayores pueden esperar una interacción limpia y clara. Las pruebas consisten en

ponerse el sombrero del usuario final, por lo que es vital conocer el conjunto de personas usuarias a las que la aplicación pretende servir y explorar cómo cada persona percibirá la aplicación e interactuará con ella.

Dominio

Cada dominio -redes sociales, transporte, salud, etc.- tiene un flujo de trabajo, un proceso y un conjunto de terminología o jerga adaptados que hay que comprender para iniciar la exploración. El comercio electrónico es un ejemplo perfecto en el que el conocimiento del dominio resulta fundamental en las pruebas. Por ejemplo, un pedido, una vez creado, pasa por un flujo de trabajo definido: capturar, prometer, confirmar, etc. El flujo de cumplimiento del pedido tiene que interactuar con numerosas partes, como el almacén que almacena los artículos, el socio de envío que transporta los artículos desde el almacén hasta el cliente, y los proveedores que reponen los artículos regularmente. Así que, al observar el comportamiento de la aplicación durante las pruebas exploratorias, tienes que saber cómo explorar todas las rutas de la aplicación. Puede resultarte difícil hacerlo sin un conocimiento básico del dominio.

Prioridades empresariales

Considera el escenario en el que la prioridad empresarial es diseñar la solución **como una plataforma** con fines de extensibilidad y escalabilidad. En estos casos, puede que no baste con probar el flujo de usuario funcional desde la interfaz de usuario. Hay que explorarlo desde el punto de vista de la "plataforma", observando si la IU y los servicios web están estrechamente acoplados o si los servicios web son independientes y pueden integrarse con otros sistemas, y otros ángulos similares.

Infraestructura y configuración

Como ya se ha comentado, las pruebas exploratorias implican entrometerse en el entorno de pruebas para simular escenarios en tiempo real, incluidos casos de fallo. Disponer de información sobre qué componentes de la aplicación están desplegados y dónde, así como sobre las palancas configurables, proporcionará pistas fundamentales para encontrar nuevas vías de descubrimiento. Por ejemplo, los servicios web pueden estar configurados con el número máximo de accesos que pueden servir en un periodo de tiempo, lo que se conoce como *limitación de velocidad*; puede que necesites observar el comportamiento de la aplicación cuando se supera el límite de velocidad. Recopilar alguna información básica sobre la infraestructura y la configuración, como la forma en que están desplegados los servicios y la base de datos (en una sola máquina o repartidos en varias máquinas), la configuración de la limitación de velocidad, la configuración de la pasarela API y similares, te ayudará a descubrir casos de prueba importantes.

Arquitectura de la aplicación

El conocimiento de la arquitectura de la aplicación añadirá ramas a tus rutas de descubrimiento en una sesión de pruebas exploratorias. Por ejemplo, si la arquitectura implica servicios web, puede que necesites realizar pruebas exploratorias de la API (de las que se habla en "[Pruebas de la API](#)") en lugar de limitarte a explorar la interfaz de usuario. Del mismo modo, si la aplicación implica flujos de eventos (de los que se habla en [el Capítulo 5](#)), explorar los casos en torno a la comunicación asíncrona adquiere importancia. Comprender la arquitectura a alto nivel te ayudará a esculpir las vías de descubrimiento en términos de integraciones de componentes internos, flujo de datos entre componentes, integraciones de terceros y gestión de errores. Varios de estos aspectos también se tratan a lo largo del libro.

Una vez que hayas reunido suficiente información sobre estas cinco áreas de aplicación, estarás listo para sumergirte en la fase de pruebas exploratorias propiamente dicha.

Si todo esto te parece un poco abrumador -especialmente las partes sobre arquitectura e infraestructura-, no te preocupes demasiado por estos detalles ahora. Está perfectamente bien abordar las pruebas exploratorias desde un punto de vista funcional y aprender gradualmente a hacer más preguntas en este sentido.

Explora por partes

El siguiente semicírculo del diagrama de la estrategia de pruebas exploratorias manuales de [la Figura 2-7](#) apunta a la exploración por partes.

En su artículo de 2003 "[Exploratory Testing Explained](#)", James Bach define esta práctica como "aprendizaje simultáneo, diseño de pruebas y ejecución de pruebas ". A día de hoy, ésta sigue siendo una de las definiciones más utilizadas de las pruebas exploratorias. Para explicarlo con más detalle, las pruebas exploratorias consisten en realizar una serie de acciones en la aplicación mientras se observa el comportamiento, y así aprender más sobre la aplicación y explorarla de forma incremental. Un proceso así exige que nuestro cerebro esté alerta todo el tiempo y que hagamos *análisis en profundidad*. Como humanos, se nos da mejor prestar una atención intensa y adentrarnos realmente en las profundidades de algo cuando nos centramos en ámbitos de trabajo más pequeños. Por eso debemos explorar partes individuales de la aplicación cada vez. Esas partes pueden ser cualquiera de las vías de descubrimiento comentadas anteriormente, o una subrama de la vía, como un flujo de usuario, una característica o un aspecto interfuncional como la seguridad.

No perder de vista todos estos caminos y ramificaciones mientras exploras en profundidad puede resultar difícil. Una estrategia para

hacer frente a esto es utilizar un mapa mental¹ como el de la [Figura 2-8](#). Puede compartirse con todo el equipo.

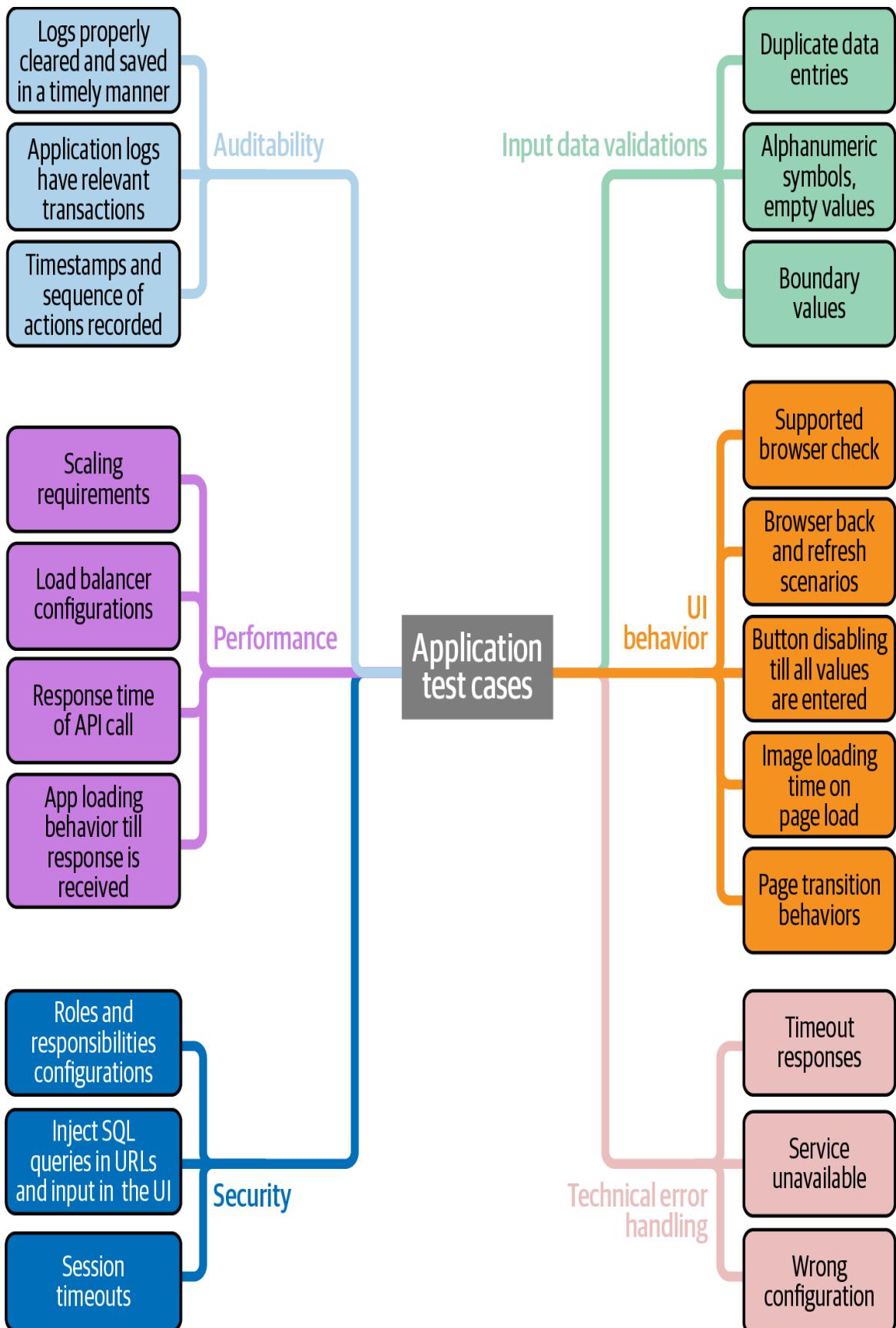


Figura 2-8. Un mapa mental de pruebas exploratorias

Durante esta fase, también puedes necesitar el paquete de ocho marcos de pruebas, representado como el semicírculo interior de la [Figura 2-7](#).

Repetir las pruebas exploratorias por fases

Las pruebas exploratorias no pueden ser una actividad puntual. El equipo estará continuamente añadiendo nuevo código, nuevas funciones, y nuevas integraciones, lo que provocará cambios en el comportamiento de la aplicación y, por tanto, hará necesaria de nuevo la exploración. Considerar las pruebas exploratorias como un proceso continuo te permitirá estructurar el ámbito que debe explorarse en profundidad en un momento o fase concretos. Por ejemplo, algunos equipos ágiles practican *la prueba dev-box*, en la que los representantes del negocio y los probadores realizan pruebas exploratorias limitadas en el tiempo de la historia de usuario que se acaba de desarrollar en la propia máquina del desarrollador. Aquí puedes restringir el alcance a sólo los flujos de usuario positivos, las validaciones y el aspecto y la sensación de la interfaz de usuario. La siguiente fase propicia para la exploración es la fase de prueba de la historia de usuario posterior al desarrollo. Aquí, puedes ampliar el alcance de la exploración para incluir algunos de los aspectos entre navegadores y funciones. Además, algunos equipos ágiles llevan a cabo "*bug bashes*" regulares, en los que todos los miembros del equipo se reúnen y exploran las características de la aplicación desarrolladas hasta el momento. Y, por último, en la fase de pruebas de lanzamiento, puedes centrarte en aspectos interfuncionales como el rendimiento, la fiabilidad y la escalabilidad en profundidad, y explorar los flujos de usuario positivos y las integraciones a un nivel ligeramente superior.

Planificar tus fases de pruebas exploratorias con antelación ayudará

al equipo a obtener una retroalimentación continua y, por tanto, dejará espacio para la mejora continua.

CONSEJO

Las pruebas exploratorias son de naturaleza orgánica. Por lo tanto, puedes descubrir nuevos caminos que no habías previsto y que pueden consumir el tiempo asignado en una iteración. Esto es de esperar. Un consejo en este caso es considerar si una vía puede incluirse en la siguiente fase de pruebas de historias de usuario o en las pruebas de errores, y si es así anótalo y sigue adelante.

Para recapitular la estrategia, cuando empieces las pruebas exploratorias, primero conoce los detalles de la aplicación, y luego anota las vías individuales a explorar. A continuación, continúa tu exploración de las distintas vías a lo largo de las fases del ciclo de entrega, con el fin de proporcionar una retroalimentación continua al equipo.

Ejercicios

Hasta ahora hemos discutido mucha teoría sobre marcos y estrategias. Para aplicarlas a la exploración de las rutas de descubrimiento de una aplicación, puede que necesites aprender algunas herramientas relevantes, como SQL para explorar la base de datos (de la que se habla en [el Capítulo 5](#)), Postman para explorar las API, etc. Este libro cubre varias de estas herramientas. En esta sección, hablaremos de las herramientas de pruebas exploratorias de API y de interfaz de usuario web.

Pruebas API

Una interfaz de programación de aplicaciones, o **API**, proporciona una forma de que los sistemas interactúen entre sí . Las API

básicamente abstraen las complejidades subyacentes de un sistema y simplifican el intercambio de información a través de la red como XML, JSON o texto plano utilizando el protocolo HTTP. Para estandarizar el intercambio de información, se inventaron protocolos como SOAP y especificaciones como REST. Hoy en día, las API RESTful son más frecuentes en que las SOAP, e incluso los sistemas heredados que utilizan SOAP se están reescribiendo con especificaciones REST. Para ayudarte a entender las API REST, consideremos una aplicación básica de comercio electrónico como la de [la Figura 2-9](#), con tres servicios REST (el pedido, la autenticación y los servicios al cliente), una interfaz de usuario y una base de datos.

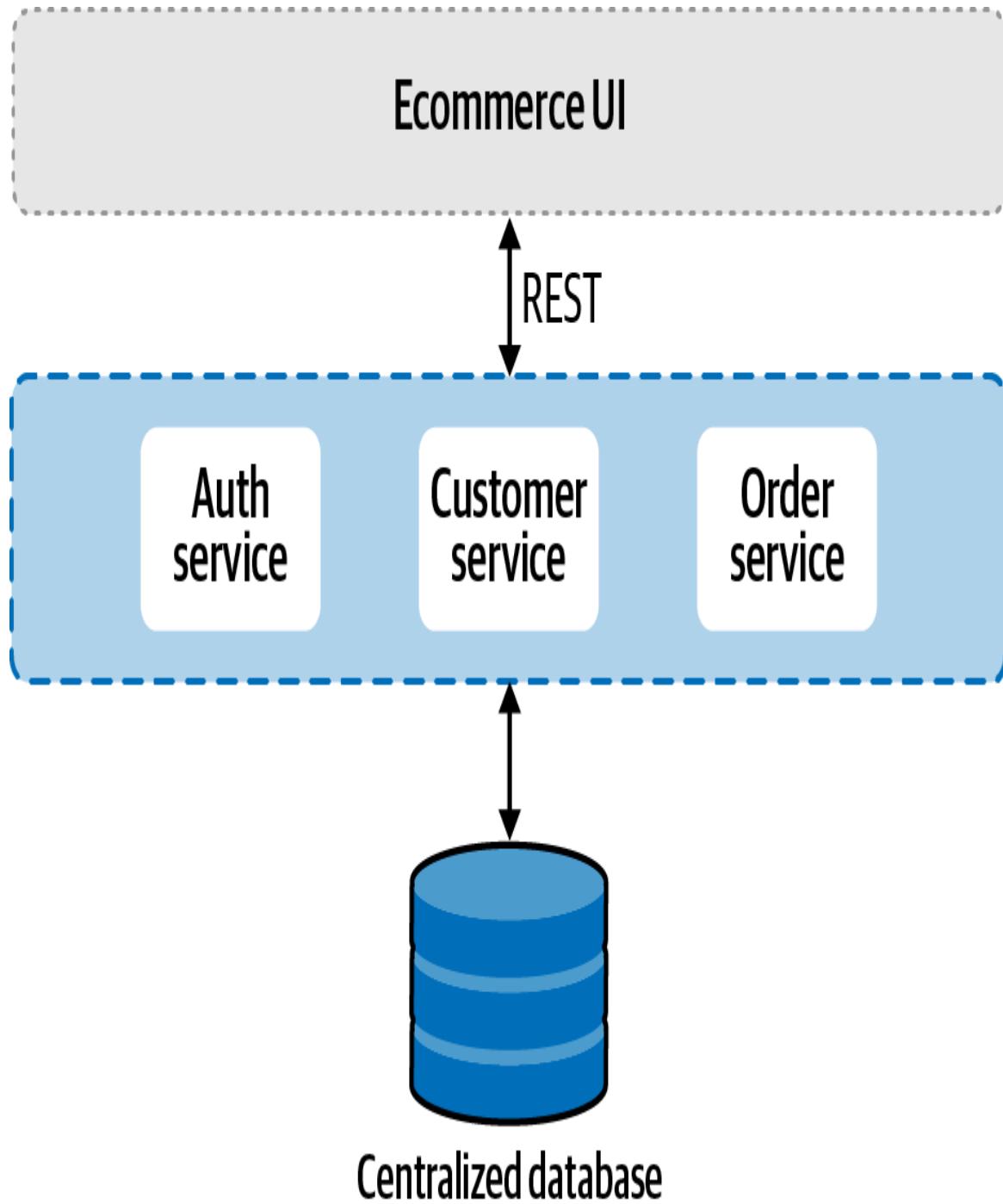


Figura 2-9. Un ejemplo de aplicación de comercio electrónico con una arquitectura basada en servicios

Un servicio web es un componente que sirve a un propósito independiente dentro de una aplicación. Por ejemplo, el servicio de pedidos de nuestra aplicación de comercio electrónico de ejemplo puede tener la responsabilidad de gestionar (crear, actualizar y

eliminar) los pedidos, mientras que el servicio de atención al cliente se encarga de mantener los datos de los clientes. Esto facilita el intercambio de información, ya que los demás componentes de la aplicación, como la interfaz de usuario u otros servicios, pueden acceder a la API del servicio correspondiente para obtener la información que necesitan.

NOTA

Una *arquitectura orientada a servicios* es aquella en la que las funcionalidades centrales de la aplicación se escriben como servicios web, como en la [Figura 2-9](#).

Por poner un ejemplo, supongamos que un usuario final completa el pago de un pedido a través de la IU de comercio electrónico. Como se muestra en [el Ejemplo 2-1](#), la IU enviará inmediatamente una solicitud de creación de pedido al servicio de pedidos con los detalles del pedido, y el servicio de pedidos procesará la solicitud y enviará una respuesta de vuelta a la IU. La interfaz de comercio electrónico se denomina *cliente* en este contexto.

Ejemplo 2-1. Ejemplo de solicitud y respuesta REST

// Request

```
POST method: http://eCommerce.com/orders/new
{
    "name": "V-Neck Tshirt",
    "sku": "ABCD1234",
    "color": "Red",
    "size": "M"
}
```

// Response

Status Code: 200 OK

Response Body:

```
{
```

```
        "Msg": "successfully created",
        "ID": "Order1234227891"
    }
```

Si observas la solicitud del [Ejemplo 2-1](#), verás que llega a la API `/orders/new` utilizando el método HTTP POST. Normalmente, POST se utiliza para crear o añadir nueva información, y GET se utiliza para recuperar información -por ejemplo, obtener una lista de los pedidos realizados por un cliente. También existen los métodos PUT y DELETE, que se utilizan para operaciones de actualización y eliminación, respectivamente. El cuerpo de la solicitud empaqueta los detalles del pedido -en este caso, el nombre del artículo, la unidad de mantenimiento de existencias (SKU), el color y la talla- como un objeto JSON. Toda esta estructura se denomina *contrato*. Si el cliente no se atiene a este contrato, el servicio no procesará la solicitud.

Del mismo modo, la respuesta también se ciñe a un contrato: incluirá un código de estado que indica el éxito o el fracaso de la operación y también puede tener un cuerpo de respuesta que da más información sobre la operación. En el [Ejemplo 2-1](#), el código de estado de la respuesta es `200 OK`, que indica éxito, y el cuerpo de la respuesta tiene un mensaje que dice "creado con éxito" junto con el ID del pedido generado por el servicio de pedidos. Al recibir esta respuesta, la interfaz de usuario del comercio electrónico llevará al usuario a la página de confirmación del pedido y mostrará el ID del pedido en esa página. Ten en cuenta que todas estas acciones se realizarán de forma sincrónica, es decir, la interfaz de usuario de comercio electrónico esperará a recibir una respuesta antes de pasar a la página de confirmación del pedido.

Ahora bien, dado este conocimiento básico de cómo funcionan las API, puede que te surja una pregunta: ¿por qué necesitamos explorar las API por separado cuando podemos probar la funcionalidad de creación de pedidos desde la interfaz de usuario web? La respuesta concisa es que, hoy en día, las API se han

convertido en productos en sí mismas! Una respuesta algo más elaborada es que las API engloban toda la lógica empresarial y las validaciones, lo que las convierte en productos independientes que otros componentes internos y externos pueden reutilizar. Por ejemplo, nuestra hipotética empresa de comercio electrónico podría crear una nueva aplicación móvil de compras y reutilizar la misma API de creación de pedidos, o crear un portal de atención al cliente con las mismas API de atención al cliente. Incluso podrían ramificarse en un dominio completamente nuevo, y reutilizar las API del servicio de autenticación para crear la funcionalidad de inicio de sesión en ese nuevo producto. Por tanto, explorar las API como productos independientes es muy importante en el mundo digital actual.

Éstas son algunas de las vías de descubrimiento, aparte de la lógica empresarial central de , a las que debes prestar atención cuando explores las API:

Validación del contrato de solicitud

La validación debe realizarse de modo que, por ejemplo, si un cliente fraudulento crea un nuevo pedido con formatos de datos no válidos, el servicio de pedidos rechace la solicitud.

Autenticación

La mayoría de las veces, las API están protegidas con algunos mecanismos de autenticación por razones de seguridad, como el envío de un token (una larga cadena cifrada) en la cabecera de la solicitud. Esta es una importante vía de descubrimiento para la exploración.

Permisos

Las API pueden tener restricciones en las operaciones que pueden realizar para sus clientes. Por ejemplo, un administrador puede estar autorizado a editar un pedido existente, pero un

ejecutivo de un cliente puede estar restringido a sólo ver el pedido.

Compatibilidad con versiones anteriores

A veces, a medida que el producto evoluciona, también puede ser necesario cambiar los contratos de las API. Pero como puede haber clientes que utilicen las API, puede ser necesario crear y mantener nuevas versiones paralelamente a las antiguas. La aplicación debe probarse con ambas versiones de la API.

Códigos de estado HTTP

Los códigos de estado devueltos por fallos técnicos y empresariales deben ser relevantes. **La Tabla 2-4** enumera los códigos de estado más comunes.

Tabla 2-4. Códigos de estado HTTP y su significado

Código de estado	Significado
200 OK	Indica el éxito de las peticiones GET, PUT o POST
201 Creado	Indica que se ha creado un nuevo objeto, como un nuevo pedido
400 Petición errónea	Indica que la petición estaba malformada
401 No autorizado	Indica que el cliente no puede acceder al recurso solicitado y debe volver a emitir la solicitud con las credenciales necesarias
403 Prohibido	Indica que la solicitud es válida y que el cliente está autenticado, pero que no se le permite acceder a la página o recurso solicitados por algún motivo
404 No encontrado	Indica que el recurso solicitado no está disponible ahora
500 Error interno del servidor	Indica que la petición es válida pero el servidor no puede gestionarla, posiblemente debido a errores internos
503 Servicio no disponible	Indica que el servidor está inactivo (por ejemplo, cuando está en mantenimiento).

Para explorar todas estas vías de descubrimiento necesitas herramientas, y en las próximas secciones te presentaré algunas.

Cartero

Postman es una herramienta común de para pruebas de API. Los binarios de instalación para la versión de escritorio están disponibles gratuitamente, y también hay disponible una versión web para probar. Aquí te daré una rápida introducción a la aplicación de escritorio:

1. Descarga el binario de instalación para tu sistema operativo desde el [sitio oficial](#).
2. Abre Postman y selecciona Nuevo → Solicitud HTTP. Esto te llevará a la ventana de nueva solicitud.
3. Haz una búsqueda en Google de "pruebas exploratorias" en tu navegador, luego copia la URL y pégala en el campo URL de la nueva ventana de solicitud de Postman, como se ve en [la Figura 2-10](#). Observa que el método HTTP se establece automáticamente en GET en el desplegable situado junto a este campo.
4. Verás que los parámetros de la petición de búsqueda de Google se rellenan automáticamente en la pestaña Parámetros. El parámetro de consulta q mostrará `exploratory+testing`. Puedes cambiarlo por cualquier otra palabra clave para buscarla.
5. Pulsa el botón Enviar para completar la solicitud.
6. Recibirás el código de estado de la respuesta, las cabeceras, el cuerpo y las cookies en el panel inferior, como se muestra en la [Figura 2-10](#).

GET Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	q	exploratory+testing			
<input checked="" type="checkbox"/>	safe	active			
<input checked="" type="checkbox"/>	sxsrf	ALeKk01ktf8sgcZ25yD83QxyRoJJCwBdw%3A161...			

Body Cookies (3) Headers (15) Test Results Status: 200 OK Time: 2.23 s Size: 107.69 KB Save Response

Pretty Raw Preview Visualize HTML

```
1 <!doctype html>
2 <html lang="en-IN">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta content="/images/branding/googleleg/1x/googleleg_standard_color_128dp.png" itemprop="image">
```

Figura 2-10. Crear una nueva solicitud y verificar la respuesta con Postman

En este caso, la respuesta es HTML. Si haces clic en el botón Vista previa, verás exactamente la misma página de resultados de búsqueda en tu navegador. En muchos casos, la respuesta será en cambio un objeto JSON, como vimos en [el Ejemplo 2-1](#); la interfaz de usuario analizará el JSON y mostrará la información correcta.

La búsqueda en Google era una solicitud GET, pero los pasos para crear una solicitud POST son similares: elige POST como método HTTP en el desplegable, introduce la solicitud API en el campo URL y el cuerpo de la solicitud en la pestaña Cuerpo, y pulsa Enviar para ver la respuesta. Si quieras practicar un poco más, [Cualquier API](#) tiene una lista consolidada de 1.400 API REST alojadas públicamente que puedes probar.

ADVERTENCIA

Postman, por defecto, guarda tu espacio de trabajo en la nube bajo tu cuenta Postman. Esta función es útil para [sincronizar tu trabajo](#) en una nueva máquina. Sin embargo, asegúrate de que la sincronización con la nube de Postman no viola ningún acuerdo de confidencialidad (NDA) con clientes o tu política interna de TI.

Postman proporciona varias otras facilidades para explorar APIs. Aquí se enumeran algunas de las disposiciones más utilizadas:

- El token enviado con fines de autenticación puede enviarse junto con la solicitud añadiéndolo en la pestaña Autorización. Puedes añadir allí cadenas no válidas para verificar que la solicitud falla.
- Del mismo modo, cuando se envían cookies junto con la solicitud, se pueden añadir en la pestaña Cookies (la pestaña está debajo del botón Enviar).

- Postman captura el tiempo que se tarda en recibir la respuesta, como se ve junto al código de estado en [la Figura 2-10](#). Esto es útil para explorar rápidamente si el rendimiento se degrada para diferentes entradas.
- En lugar de crear solicitudes manualmente, puedes importar directamente las especificaciones de la API desde Swagger, OpenAPI, etc., con sus enlaces.

Postman también ofrece soporte para probar los servicios GraphQL y SOAP de , además de los servicios REST.

WireMock

[WireMock](#) es una herramienta para crear y modificar *stubs*, que son componentes de software que emulan los comportamientos de otro componente. Los stubs son especialmente útiles cuando se desarrollan y prueban aplicaciones complejas con múltiples integraciones, en las que aún no están listos todos los servicios que las integran. Los equipos acuerdan un contrato de servicios y pueden continuar el desarrollo creando stubs de los servicios integradores. Los stubs se crean programándolos explícitamente para que respondan a determinadas peticiones con una salida definida. Esta función puede utilizarse en las pruebas exploratorias para establecer distintos casos de prueba de integración positivos y negativos. Por supuesto, es imprescindible que vuelvas a probar la funcionalidad de extremo a extremo con los componentes reales una vez que estén listos.

NOTA

De instalar el servidor stub y configurar la aplicación para que apunte al stub puede encargarse un ingeniero DevOps o los desarrolladores del equipo. Sin embargo, los probadores necesitan saber cómo modificar los stubs para simular los casos de prueba. Este ejercicio se incluye específicamente con ese fin.

Para ilustrar el uso de WireMock, volvamos a nuestra aplicación de comercio electrónico de ejemplo. Supongamos que aún no tenemos un servicio de pago real que integrar, pero conocemos el contrato de solicitud y respuesta del punto final `/makePayment`, que la interfaz de usuario del comercio electrónico utiliza para enviar pagos. Para explorar los distintos casos de prueba de esta integración, necesitamos configurar un stub del endpoint `/makePayment` con respuestas positivas y negativas. Estos son los pasos:

1. Descarga el JAR independiente de WireMock desde el [sitio web oficial](#).
2. Abre el terminal y ejecuta el siguiente comando con la versión JAR que has descargado:

```
$ java -jar wiremock-jre8-standalone-x.x.x.jar
```

El comando iniciará un servidor WireMock en el puerto 8080.

3. Para crear un nuevo stub, construye el contrato de la API `/makePayment`, como se ve en el [Ejemplo 2-2](#), y envía una solicitud POST al endpoint http://localhost:8080/_admin/mappings/new utilizando Postman. (Es decir, en la ventana de nueva solicitud de Postman, establece el método HTTP como POST en el desplegable, introduce la URL en el campo URL y el JSON del [Ejemplo 2-2](#) en la sección Cuerpo → raw, y pulsa Enviar).

Ejemplo 2-2. Ejemplo de stub utilizando WireMock

```
{  
    "request": {  
        "method": "POST",  
        "url": "/makePayment"  
    },  
    "response": {  
        "status": 200,  
        "body": "Payment Successful"  
    }  
}
```

4. Ahora comprueba que el stub funciona creando otra solicitud POST para acceder a la *URL* <http://localhost:8080/makePayment>. Deberías recibir una respuesta con el código de estado 200 OK y el mensaje "Pago correcto", tal y como se describe en el stub. Nuestra IU imaginaria de comercio electrónico debería mostrar una página de confirmación del pedido al recibir esta respuesta.
5. Ahora, para modificar el stub para que devuelva una respuesta de fallo, cambia el cuerpo de la respuesta del Ejemplo 2-2 por el siguiente y POSTéalo al mismo endpoint /mappings/new:

```
"response": {  
    "status": 401,  
    "body": "Payment Unauthorized"  
}
```

Al recibir esta respuesta, la interfaz de usuario del comercio electrónico debe mostrar un mensaje de error.

Del mismo modo, puedes configurar otros casos de prueba (solicitudes no válidas, escenarios de servicio no disponible, etc.) con códigos de estado apropiados en el cuerpo de la respuesta y observar si la interfaz de usuario los gestiona adecuadamente. Como puedes ver, los stubs son de gran ayuda en las pruebas exploratorias de la API cuando los servicios de integración reales no están disponibles para probarlos.

Ahora podemos pasar a las herramientas de pruebas exploratorias de la interfaz de usuario web.

Pruebas de interfaz de usuario web

Esta sección arroja luz sobre tres herramientas básicas de pruebas de interfaz de usuario web : navegadores, Bug Magnet y Chrome DevTools.

Navegadores

La primera y principal herramienta para explorar una interfaz de usuario web es el navegador. Una buena práctica es cubrir al menos el 85% de la base de usuarios de tu aplicación durante las pruebas. En el momento de escribir esto, las estadísticas más recientes sobre la distribución global del uso de navegadores de gs.statcounter.com mostraban que Chrome tiene una cuota del 64,5%, seguido de Safari con un 18,8%, luego Edge con un 4,05%, Firefox con un 3,4% y Samsung Internet con un 2,8%. Estas estadísticas indican que debes incluir Chrome y Safari en tus pruebas, pero el tercer puesto oscila entre Edge y Firefox con frecuencia, por lo que es aconsejable incluir ambos para explorar la calidad de la IU. Puedes descargar cualquiera de estos navegadores en tu máquina local, para cualquier SO.

NOTA

A veces es necesario realizar pruebas en navegadores antiguos como Internet Explorer 11 o Edge Legacy, aunque Microsoft ha finalizado oficialmente la compatibilidad con estas versiones. Una forma de hacerlo es [descargar la VM de Windows](#) en tu máquina.

Alternativamente, las plataformas de pruebas alojadas en la nube, como [BrowserStack](#) y [Sauce Labs](#), te absuelven de la necesidad de instalar diferentes versiones de navegadores y sistemas operativos en máquinas locales. Proporcionan acceso virtual a navegadores web en distintos SO, a cambio de un coste. El proceso es sencillo: paga una suscripción (también hay disponibles pruebas gratuitas), inicia sesión en el portal, elige una combinación de versión de navegador y SO (como se ve en [la Figura 2-11](#)), y prueba tu aplicación.

Quick Launch	c	e	f	g	o	y	n
Android	89 Latest	11 Latest	87 Latest	89 Latest	75 Latest	14.12 Latest	5.1 Latest
iOS	90 Beta	10	88 Beta	90 Beta	76 Dev		5
Windows	91 Dev	9	86	91 Dev	74		4
10	88	8	85	88	73		
8.1	87		84	87	72		
8	86		83	86	71		
7	85		82	85	70		
XP	84		81	84	69		
Mac	+ 83		80	83	68		
	+ 81		79	81	67		
	1 more		77 more	66 more	58 more		

Figura 2-11. BrowserStack y otros servicios similares te permiten probar varias combinaciones de versiones de SO y navegador.

BrowserStack también permite **realizar pruebas locales** de aplicaciones privadas, alojadas en entornos de control de calidad (QA) o de ensayo. Dependiendo de tus necesidades de pruebas, puede merecer la pena suscribirse a un servicio de este tipo -pueden ser valiosos, especialmente cuando necesites realizar pruebas en una amplia gama de navegadores antiguos.

Imán para insectos

Bug Magnet es un complemento de navegador disponible para Chrome y Firefox que permite probar casos de perímetro en una aplicación. Proporciona una lista de casos de prueba comunes y los valores adecuados que hay que introducir en los elementos editables de la aplicación para cada caso de prueba. La herramienta ayuda principalmente como lista de comprobación para pruebas exploratorias. Para probarla:

1. Instala el **complemento** en tu navegador Chrome.
2. Abre **la búsqueda de Google** y haz clic con el botón derecho del ratón en el campo de texto de búsqueda.
3. Encontrarás Imán de Bichos en el menú del botón derecho del ratón, como se ve en la **Figura 2-12**. Como puedes ver, te sugiere muchos casos de perímetro, de los que puedes seleccionar uno. Por ejemplo, elige Nombres → Longitud del nombre y selecciona el nombre de pila. El nombre largo se llenará en el cuadro de texto de búsqueda de Google. Si hay validaciones en tu aplicación para la longitud de la cadena de entrada, debería aparecer un mensaje de error apropiado.

Google

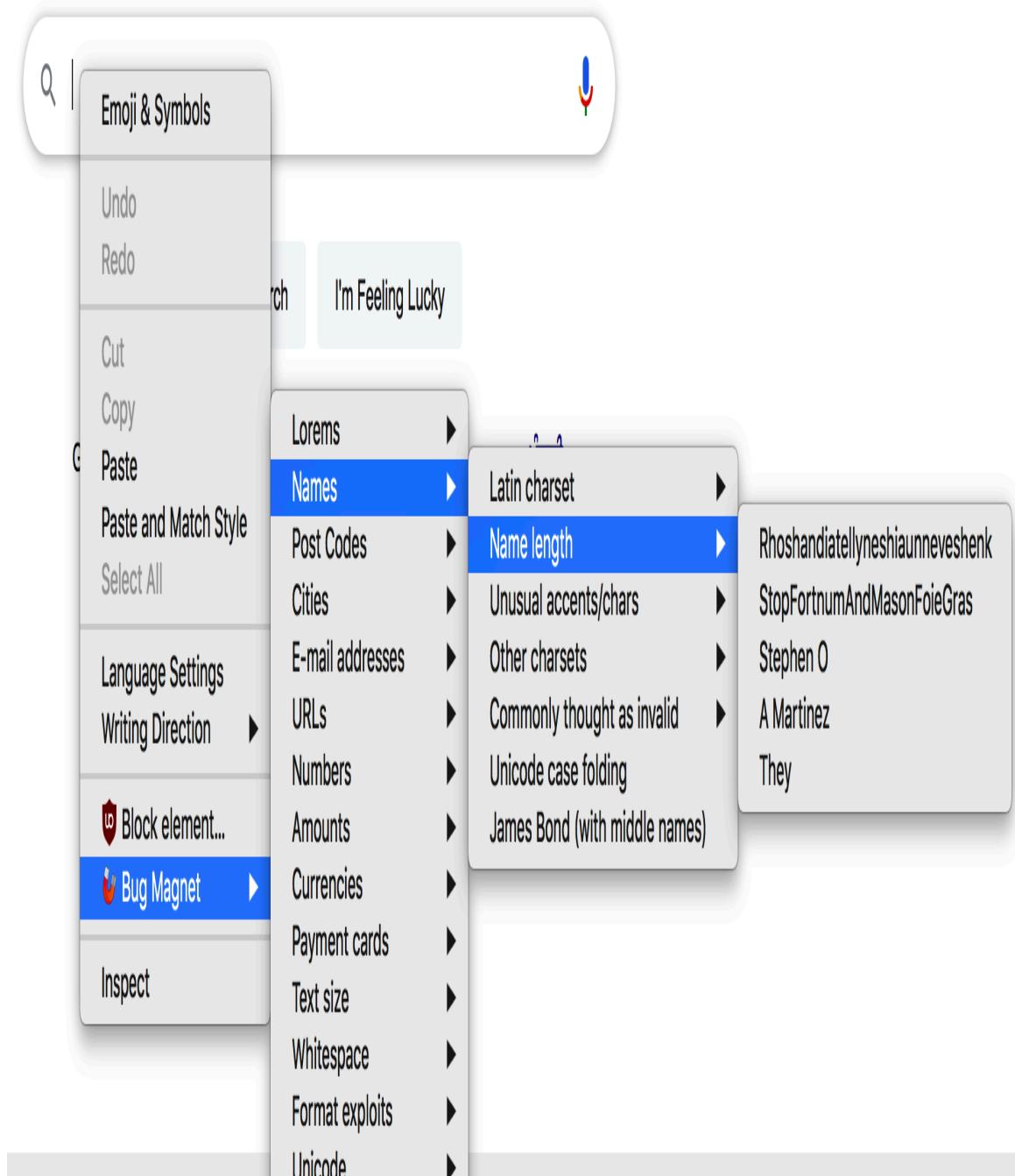


Figura 2-12. Puedes utilizar el complemento Imán de fallos como guía durante las pruebas exploratorias manuales.

CONSEJO

Además de Bug Magnet, también hay varias **hojas de trucos de heurística de pruebas exploratorias** que puedes utilizar para asegurarte de que no se te escapan casos de prueba. Estas son especialmente útiles para los principiantes.

Chrome DevTools

Chrome DevTools es tan versátil como una navaja suiza. Viene en con multitud de disposiciones que ayudan en las pruebas exploratorias, las pruebas de seguridad, las pruebas de rendimiento y mucho más. A lo largo del libro, verás aparecer esta herramienta en distintos lugares. Para hacerte una idea de lo que ofrece

1. Abre tu navegador Chrome y busca "pruebas exploratorias".
2. Haz clic con el botón derecho del ratón en la página de resultados de la búsqueda y selecciona la opción Inspeccionar. Las DevTools se abrirán inmediatamente. También puedes utilizar las teclas de acceso rápido Cmd-Opción-C o Cmd-Opción-I en macOS o Mayúsculas-Ctrl-J en Windows para abrir DevTools.

A partir de ahí, puedes explorar una serie de cosas como las siguientes:

Errores de página

Como se ve en **la Figura 2-13**, la pestaña Consola muestra los errores de la página web. Una página web debería mantenerse libre de errores como práctica general, por lo que es una buena idea comprobar esta pestaña al entrar en cada nueva página de la aplicación de prueba. Los errores notificados en esta pestaña

también pueden ayudarte a depurar cualquier problema que encuentres en una página web. Por ejemplo, si ves que falta una imagen, puedes comprobar la pestaña Consola e incluir el error notificado aquí con tu informe de error.

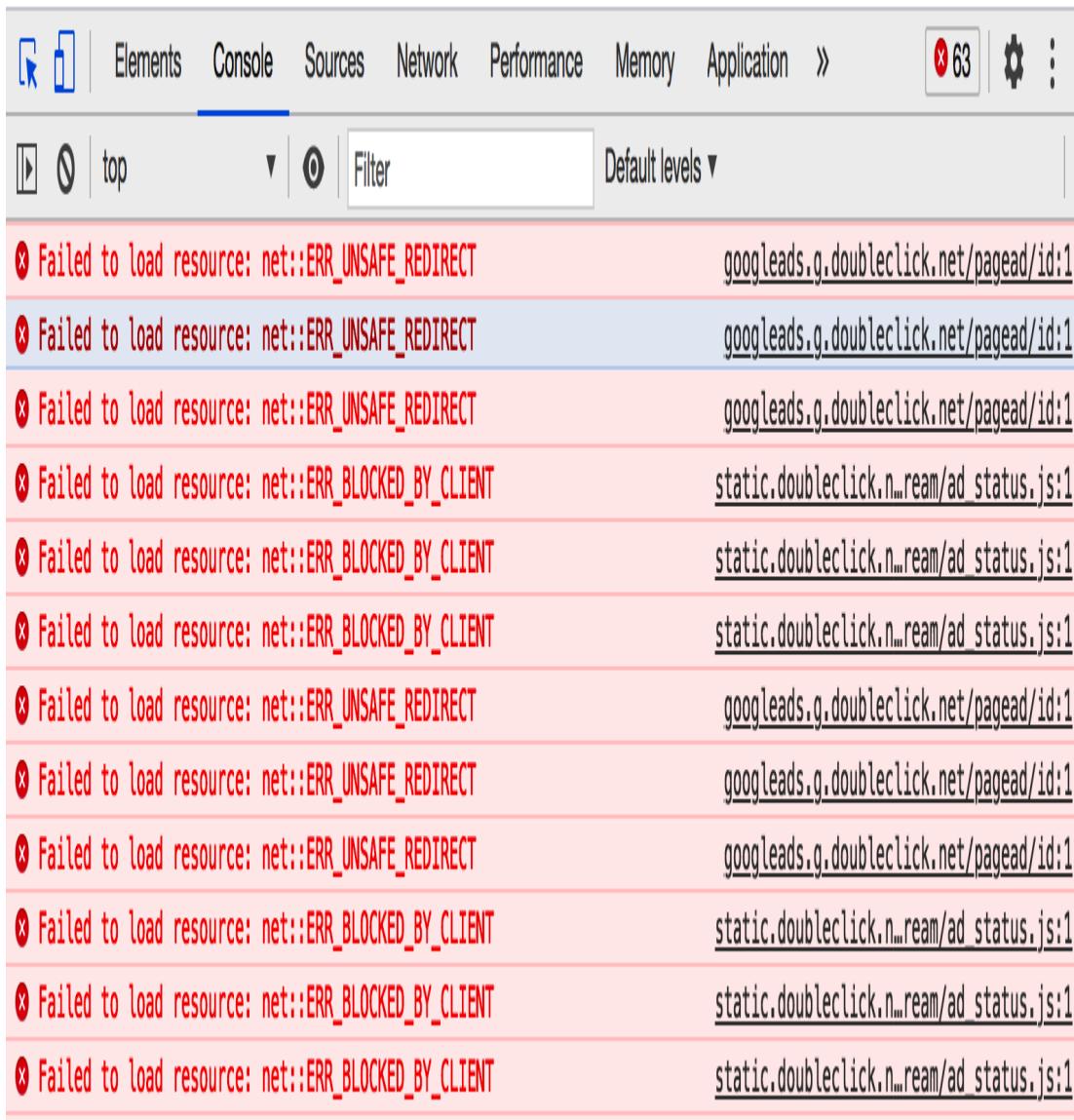


Figura 2-13. La pestaña Consola muestra los errores de una página web

Número de peticiones realizadas desde una página

A veces, debido a errores en la lógica de la aplicación , una página web puede hacer muchas **llamadas a la API no deseadas** y volverse lenta. Puedes detectar estos problemas en la pestaña

Red, que muestra el recuento total de peticiones enviadas desde esa página en la parte inferior izquierda.

Comportamiento del primer usuario

Cuando pruebas repetidamente la misma aplicación , algunos de los recursos (como las imágenes de las páginas web) se almacenan en la caché. Por eso, si se cambia una imagen durante el desarrollo, puede pasar desapercibido. Utiliza la casilla "Desactivar caché" de la pestaña Red para borrar la caché y volver a ver la página. Ten en cuenta también que la caché funciona de forma similar para los usuarios finales, por lo que esta disposición te ayuda a explorar la experiencia de usuario de la aplicación por primera vez.

Comportamiento de la IU en redes lentas

Para explorar la experiencia de un usuario final con un ancho de banda de red limitado, puedes estrangular la red desde la pestaña Red y observar el comportamiento de la interfaz de usuario. Como se ve en [la Figura 2-14](#), hay un menú desplegable junto a la casilla "Desactivar caché" que te permite simular condiciones de red 2G, 3G y 4G. Selecciona una opción del desplegable, borra la caché del navegador y vuelve a cargar la página. DevTools mostrará una serie de capturas de pantalla, como se ve en [la Figura 2-14](#), que cuentan la historia de cómo se carga gradualmente la aplicación con el ancho de banda especificado. Las aplicaciones web progresivas (tratadas en el [Capítulo 11](#)) funcionan incluso sin conexión, y el menú desplegable de estrangulamiento de red también incluye una opción para desconectarse y verificar este comportamiento.

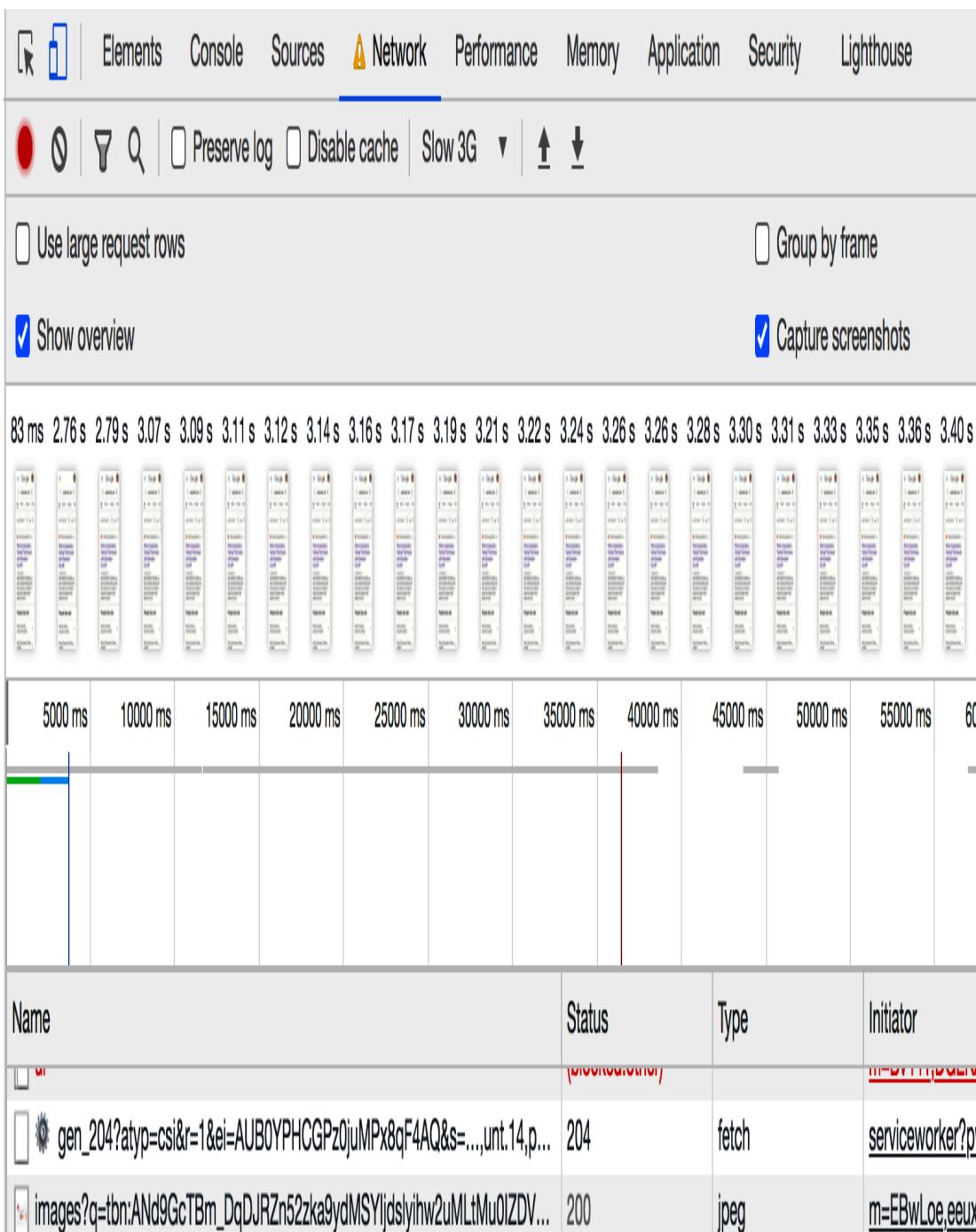


Figura 2-14. Ralentización de la red mediante Chrome DevTools

Integración de UI y API

La pestaña Red captura todas las llamadas a la red en la página web, incluyendo cualquier llamada a servicios web desde la interfaz de usuario. Registra las cabeceras de solicitud y

respuesta (incluidos los tokens de autenticación), los parámetros de consulta, las respuestas y otra información útil sobre cada solicitud realizada, como se ve en la [Figura 2-15](#).

Name	Headers	Preview	Response	Initiator	Timing	Cookies
search?q=exploratory+testing&safe=active&sxsrf=ALE.....	General					
ADGmqu9E1_4NrOMsol4PWTalb9NjzBT7dU9QLm2Y7D...	Response Headers (15)					
4UaGrENHsxJlGDuGo1Oll3Owp5eKQtG.woff2	Request Headers (16)					
4UabrENHsxJlGDuGo1OllU94YtzCwZsPF4o.woff2	Query String Parameters	view source	view URL encoded			
nav_logo289_hr.webp	q: exploratory testing					
searchbox_sprites317_hr.webp	safe: active					
data:image/png;base...	sxsrf: ALeKk01ktf8sgcZ25yD83QxyRoI0JcwBdw:1618230156223					
data:image/svg+xml,...	source: hp					
data:image/gif;base...	ei: jDt0YN_hC_uf4-EPzqKo-Ao					
data:image/png;base...	ifsig: AINFcbYAAAAYHRJnJQx4CUqXBK8Lz1r4Dopxf0ZeDOU					
data:image/png;base...	og: exploratory testing					
data:image/png;base...	gs_lcp: Cgdnd3Mtd2l6EAMyBQgAELEDmgIIADICCAyAggAMgIIADICCAyAggAMgIIADICCAyAggA0gcIIxlgwE6DgguELEDEIMBEmcBEKMCoogTlhCxAxCDAt0HCAAQhwIQFD0ICC4QxwEQrwE6CggAEIcCELEDEBQ6Aggu0QCjoHCAAQsQM0C1CysgFYo90BYKXlAwgCcAB4AIABZogB4AySAQQxOS4xmAEAoAEBqgEHZ3dzLXdperABCg					
data:image/png;base...	sclient: gws-wiz					
data:image/png;base...	ved: 0ahUKFwjf1_-M2fjvAhX7zzgGHU4RCq8Q4dUDCAC					
data:image/svg+xml,...	uact: 5					

Figura 2-15. Detalles de solicitud y respuesta en la pestaña Red

Esta información de solicitud/respuesta puede utilizarse para explorar las integraciones UI/API. Por ejemplo, puedes comprobar si la IU pasa los parámetros de consulta correctos introducidos por el usuario final al punto final correcto. Además, puedes observar el comportamiento de la IU para diferentes respuestas del servicio. Por ejemplo, la IU debería mostrar un mensaje de error "Artículo no disponible" cuando la API de disponibilidad de artículos devuelva un código de estado 404.

Comportamientos de inactividad del servicio

Cuando tengas que simular casos de prueba de fallo de solicitud , puedes bloquear la solicitud específica desde la pestaña Red y observar el comportamiento de la interfaz de usuario. Por ejemplo, encuentra la primera URL de carga de imagen en los resultados de búsqueda de Google de "pruebas exploratorias" en la pestaña Red, haz clic con el botón derecho y elige "Bloquear URL de solicitud" en el menú, y vuelve a cargar la página. Se bloqueará la carga de la imagen correspondiente en la interfaz de usuario. Esta función puede utilizarse para probar un escenario de "caída del servicio" sin que realmente se caiga.

Cookies

Las cookies se utilizan principalmente para almacenar la información de la sesión en una aplicación. La pestaña Aplicación muestra la lista de cookies almacenadas y sus detalles, como se ve en la [Figura 2-16](#). También puedes editar o eliminar valores de cookies desde aquí durante la exploración y observar el comportamiento de la aplicación.

Elements Console Sources Network Performance Memory Application Security Lighthouse

The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. In the top right corner of the main panel, there is a search bar labeled 'Filter' and a checkbox labeled 'Only show cookies with an issue'. Below this, a table lists various cookies. The columns are 'Name', 'Value', 'Domain', 'Path', and 'Expires / Max-Age'. The rows are categorized by source: Manifest, Service Workers, Storage, Local Storage, Session Storage, IndexedDB, Web SQL, Cookies, and Cache. A specific row for 'https://www.google.com' is highlighted in blue.

	Name	Value	Domain	Path	Expires / Max-Age
Manifest	_Secure-3PSIDCC	AjI4QfHt2vOAHLbnTvy7pP9LS8qJ8cUSpsBWrlbJG...	google.com	/	2022-04-12T12:55:3...
Service Workers	HSID	A6092_xIMxDtsJ_u	.google.com	/	2023-04-07T07:54:0...
Storage	_Secure-3PSID	8Qeo6n-GPjh9A_m8PBFUAp_6kSpblHeqn_R-kv6L...	.google.com	/	2023-04-07T07:54:0...
Storage	SID	8Qeo6n-GPjh9A_m8PBFUAp_6kSpblHeqn_R-kv6L...	.google.com	/	2023-04-07T07:54:0...
Local Storage	SAPISID	ActflbTp0w5A942/A4vO3URga8IR16k9e	.google.com	/	2023-04-07T07:54:0...
Session Storage	APSID	qnzKU4FUQOwnDPz4/AZbVLfwClodduWafw	.google.com	/	2023-04-07T07:54:0...
IndexedDB	SSID	AD0xmbxeS3RGDKFBR	.google.com	/	2023-04-07T07:54:0...
Web SQL	SIDCC	AjI4QfGC8u1HKMVYbNVeWhpN6nLfMBZEkmW7Y...	google.com	/	2022-04-12T12:55:3...
Cookies	_Secure-3PAPISID	ActflbTp0w5A942/A4vO3URga8IR16k9e	.google.com	/	2023-04-07T07:54:0...
https://www.google.com	CGIC	EhQxQzVDSEZBX2vJSU44MzJUTgzMIKHAXRleHQ...	google.com	/search	2021-10-09T12:51:1...
	DV	MzquT8a3u9EQOh724ycFEecKTijiJdokGqxF16laQ...	www.google.com	/	2021-04-12T13:01:3...
Cache	OGP	-19014280:	.google.com	/	2021-06-11T12:22:2...
Cache Storage	1P_JAR	2021-04-12-12	.google.com	/	2021-05-12T12:51:3...
Application Cache	OGPC	19014280-1:	.google.com	/	2021-06-11T12:21:4...
	NID	012-WV70nhwBPhn8HlCmmlKu7pDM/mnE8oh_ni	google.com	/	2021-10-10T10:01:1

Figura 2-16. Las cookies se pueden editar o borrar desde la pestaña Aplicación.

Para obtener más información sobre los detalles de todas las funciones de Chrome DevTools, consulta el [sitio oficial](#).

Ahora estás equipado con un puñado de herramientas para empezar a explorar API e interfaces de usuario web. Pero aún queda un tema clave en el camino hacia la consecución de los objetivos de las pruebas exploratorias: mantener una buena higiene en el entorno de pruebas. Hablaremos de ello a continuación.

Perspectivas: Higiene del entorno de pruebas

El entorno de pruebas es el terreno de juego real donde los probadores aplican sus habilidades de pruebas exploratorias, y cuando está mal mantenido , afecta directamente a los probadores y a sus resultados. La siguiente lista describe algunos olores de mantenimiento que puedes encontrar, su impacto y los remedios para superarlos:

Entornos de pruebas compartidos frente a dedicados

En los equipos grandes, a menudo se comparte un único entorno de pruebas entre varios subequipos, y esto impone fuertes restricciones a la capacidad de los probadores de cada equipo para inmiscuirse en el entorno de la forma dictada por sus rutas de descubrimiento. Por ejemplo, si necesitan interrumpir un servicio momentáneamente, tienen que obtener el consentimiento de otros equipos. Peor aún, tienen que coordinarse con otros equipos o esperar a la siguiente implementación programada, que puede ser una vez al día o una vez a la semana, para explorar el código más reciente. Tener un entorno de pruebas dedicado, al menos con los componentes que son competencia de un subequipo individual, dará a sus probadores la libertad de explorar aventuradamente.

Higiene de la Implementación

Una vez que el equipo disponga de su propio entorno dedicado , un enfoque adecuado es disponer de un disparador manual para las nuevas implementaciones, en lugar de implementaciones

automatizadas a través de la tubería de integración continua, ya que éstas podrían alterar las configuraciones del probador en el entorno sin previo aviso. Además, la compilación debe estar disponible para su implementación en el canal de integración continua sólo cuando haya pasado la fase de ejecución de pruebas automatizadas. Esto es para garantizar que no haya defectos abiertos en el último código, que puedan bloquear las pruebas exploratorias. En [el Capítulo 4](#) aprenderás más sobre las estrategias de CI e implementación.

Además, el entorno de prueba debe configurarse de la forma más parecida posible al entorno de producción, con cortafuegos, niveles/componentes separados, configuraciones de límites de velocidad, etc., como parte de la implementación. Sólo entonces podrán explorarse a fondo los casos de prueba de fallos comentados anteriormente.

Higiene de los datos de prueba

Los datos de las pruebas son competencia de los probadores, y deben seguir ciertas prácticas para asegurarse de que no descarrilan involuntariamente su propia exploración. En particular, ten cuidado con los datos y configuraciones obsoletos cuando sigas probando una nueva función en la misma implementación. Una recomendación para evitar estas complicaciones es procurar desplegar una nueva compilación cada vez que se inicie una nueva historia de usuario (suponiendo que una nueva implementación borrará los datos y configuraciones antiguos y restaurará la aplicación a un estado fresco). Otra opción es crear un nuevo conjunto de datos de prueba para cada historia de usuario, como un nuevo usuario, en lugar de explorar con los usuarios existentes, que podrían estar en diferentes estados.

La creación de datos de prueba puede ser compleja cuando hay cientos de tablas vinculadas. Una opción entonces es hacer que

la nueva implementación elimine los datos antiguos y los sustituya por un conjunto estándar de datos de prueba, o hacer que un script SQL cree nuevos datos de prueba apropiados como parte de la implementación. Otra posibilidad es anonimizar los datos de producción y utilizarlos en el entorno de pruebas, aunque esto puede plantear problemas desde el punto de vista de la seguridad si no se actúa con la debida diligencia.

Equipos autónomos

La mayoría de las veces, el acceso a el entorno de pruebas está restringido. Los miembros del equipo pueden no tener credenciales de inicio de sesión, o los permisos necesarios para actualizar configuraciones, mirar los registros de la aplicación o configurar stubs; para realizar acciones como éstas, necesitan solicitar ayuda al equipo de DevOps o de mantenimiento del sistema. Esto es especialmente frustrante durante las pruebas exploratorias, en las que los probadores pueden necesitar acceso a todos los componentes de la aplicación. Asegurarse de que el equipo es autónomo y tiene acceso a todo lo que necesita reducirá los retrasos debidos a dependencias externas y permitirá una entrega sin problemas.

Configuración de servicios de treinta partes

Normalmente, los servicios de terceros se dejan fuera de la configuración del entorno de pruebas, dando por supuesto que las integraciones pueden probarse directamente en producción. Esto puede dar lugar a bloqueos no deseados, especialmente cuando los problemas se descubren demasiado tarde en el ciclo de entrega. Por eso, al configurar el entorno de pruebas es importante asegurarse de que haya alguna forma de explorar las integraciones con servicios de terceros, ya sea empleando stubs o pagando por un acceso limitado a esos servicios.

Ahora que hemos discutido el qué, el por qué y el cómo de las pruebas exploratorias manuales, es importante subrayar que siguen siendo un arte, que extrae su energía de las habilidades analíticas y de observación del individuo. Y debido a esta naturaleza individualista, no existe realmente una forma establecida de validar los resultados de las pruebas exploratorias. En otras palabras, hoy tu cerebro analítico podría iniciar un camino de descubrimiento que te lleve a descubrir fallos, pero puede que mañana no haga lo mismo. Debido a esta imprevisibilidad, es importante mantener un enfoque disciplinado hacia las pruebas exploratorias siguiendo los conceptos mencionados en este capítulo.

Puntos clave

Éstos son los puntos clave de este capítulo:

- Las pruebas exploratorias manuales consisten en deambular por la aplicación de prueba con la intención de explorar y comprender el comportamiento de la aplicación, lo que eventualmente puede llevar a descubrir nuevos flujos de usuario y fallos en los flujos de usuario existentes.
- Las pruebas exploratorias manuales se diferencian de las pruebas manuales en que estas últimas consisten en comprobar una lista de especificaciones, mientras que las primeras se basan en el análisis y la aguda capacidad de observación de una persona.
- Las pruebas exploratorias reúnen las necesidades empresariales, la implementación técnica y la perspectiva del usuario final, al tiempo que cuestionan lo que se sabe que es cierto desde todos estos ángulos.
- Hablamos de un paquete de ocho marcos de pruebas exploratorias que pueden ayudar a estructurar los procesos de

pensamiento del probador y a derivar casos de prueba significativos.

- La estrategia de pruebas exploratorias manuales hace hincapié en comprender los detalles de la aplicación en cinco grandes áreas y, a continuación, iniciar la exploración de cuatro vías esenciales: los flujos de usuario funcionales, los fallos y la gestión de errores, el aspecto y la sensación de la interfaz de usuario, y los aspectos interfuncionales.
- Las pruebas exploratorias deben ser un proceso continuo. Puede planificarse para que se repita en distintas fases del ciclo de vida de la entrega, como en las pruebas de la caja de desarrollo, las pruebas de historias de usuario, las pruebas de errores y las pruebas de lanzamiento.
- Para explorar las distintas vías de descubrimiento de una aplicación, puede que tengas que aprender a utilizar nuevas herramientas. En este capítulo se han tratado herramientas relevantes de pruebas exploratorias de API y de interfaz de usuario web, como Postman, WireMock, Bug Magnet y Chrome DevTools.
- El entorno de pruebas es el terreno de juego de las pruebas exploratorias manuales, y mantener su higiene es fundamental para alcanzar los objetivos de las pruebas exploratorias. Discutimos algunos problemas comunes en el mantenimiento del entorno de pruebas y los remedios para superarlos.
- Las pruebas exploratorias manuales son un proceso muy individual, que depende de las habilidades analíticas y de observación. Estructurar el enfoque hacia las pruebas exploratorias es vital para racionalizar los resultados.

¹ Un mapa mental es una técnica de visualización en la que se plasman las ideas principales junto con sus ramas. Para dibujarlos se pueden utilizar

herramientas como [Coggle](#) y [XMind](#).

Capítulo 3. Pruebas funcionales automatizadas

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

iSube a bordo tu piloto automático!

Las pruebas automatizadas son la práctica de utilizar herramientas en lugar de humanos para realizar acciones similares a las del usuario en una aplicación y verificar su comportamiento esperado. Esta práctica existe desde la década de 1970, y las técnicas y herramientas en este ámbito han evolucionado continuamente junto con el software. Por citar algunos ejemplos, en los años 70 las aplicaciones de software se escribían predominantemente con FORTRAN y se utilizaba la herramienta RXVP para realizar pruebas automatizadas. En los años 80, cuando evolucionaron los PC, se introdujo AutoTester para realizar pruebas automatizadas. En los años 90, con el auge de la World Wide Web, se popularizaron herramientas de automatización de pruebas como Mercury Interactive y QuickTest, y se inventó la herramienta de pruebas de carga automatizadas Apache JMeter. Con el continuo avance de la web, la década de 2000 vio el nacimiento de Selenium, y el número de herramientas de pruebas automatizadas no ha dejado de crecer desde entonces. Hoy en día, disponemos incluso de herramientas de pruebas automatizadas impulsadas por IA/ML que enriquecen la experiencia general de automatización de pruebas.

Esta innovación ha sido impulsada por algunas observaciones clave: las pruebas automatizadas reducen significativamente el coste de las

pruebas y permiten a los equipos de software obtener información sobre la calidad de la aplicación más rápidamente de lo que lo harían con las pruebas manuales. Para mostrar por qué es así, consideremos un escenario en el que sólo realizas pruebas manuales a lo largo del ciclo de desarrollo de tu aplicación, y veamos cómo se comparan las pruebas automatizadas en la misma situación.

Supongamos que, de media, cada función de tu aplicación tiene 20 casos de prueba, y que tardas 2 minutos por caso de prueba en ejecutarlos, o 40 minutos en probar una función manualmente. Cada vez que se desarrolla una nueva función, tienes que probar su integración con las funciones existentes y asegurarte también de que éstas no se rompen debido a los nuevos cambios, práctica que se conoce como *pruebas de regresión*. El riesgo de no realizar las pruebas de regresión con la suficiente antelación es que sólo encontrarás los errores de integración durante las pruebas de lanzamiento, que son muy tardías en el ciclo y podrían retrasar el calendario de lanzamiento. Así, en nuestro ejemplo, las pruebas de regresión junto con las pruebas de nuevas funciones llevarán 80 minutos cuando haya una segunda función, 120 minutos cuando haya una tercera función, y así sucesivamente.

Muy pronto, cuando tu aplicación tenga que ponerse en marcha con 15 funciones, tendrás que planificar 600 minutos de tiempo de pruebas. Para empeorar las cosas, a veces, una aplicación madura tiene que funcionar en diferentes versiones de servicios. Tu tiempo de pruebas aumentará proporcionalmente al número de versiones que haya que soportar. Por ejemplo, si un servicio tiene dos versiones, el tiempo de prueba de tu aplicación se convertirá en 1.200 minutos por cada versión. Además, si encuentras errores, dependiendo de su naturaleza (por ejemplo, un error que requiera un cambio en el esquema de la base de datos), puedes acabar dedicando otros 1.200 minutos a probar la aplicación antes de ponerla en marcha! Este ciclo continuará con un aumento del tiempo de prueba a medida que se añadan nuevas funciones a cada versión.

Las empresas que no invierten lo suficiente en pruebas automatizadas combaten este problema aumentando su capacidad de pruebas manuales, pero siguen obteniendo una respuesta más lenta que con las pruebas automatizadas. Por ejemplo, en el caso de nuestra aplicación hipotética, aunque haya 12 personas realizando pruebas en paralelo, seguirán tardando 100 minutos, mientras que las pruebas automatizadas en las capas adecuadas pueden ejecutarse mucho más deprisa y dar una respuesta más rápida. También es importante no olvidar que, si tienes pruebas automatizadas, no tienes que reunir a tus 12 compañeros de equipo a medianoche para probar una corrección urgente de un defecto de producción antes de lanzarla. E incluso si te atreves, las pruebas manuales pueden ser propensas a errores, ya que dependen en gran medida de la calidad de la documentación y la ejecución de los casos de prueba.

Por supuesto, crear pruebas automatizadas y ejecutarlas con regularidad tiene un coste. Sin embargo, es un coste que debe sopesarse frente al valor de entregar el producto rápida y frecuentemente al mercado, el gasto de las pruebas manuales (en términos de tiempo y capacidad), y la confianza que da al equipo durante el desarrollo y mientras soluciona los problemas de producción.

En resumen, una recomendación para las empresas es que necesitas tanto las pruebas manuales como las automatizadas para ofrecer un producto de alta calidad, y una estrategia inteligente es equilibrarlas: elegir una u otra no es una opción. En palabras sencillas, la estrategia podría ser: utilizar tu capacidad de pruebas manuales para realizar pruebas *exploratorias* manuales para descubrir nuevos casos de prueba, y automatizarlas para atender a las pruebas de regresión.

Ya hablamos de las pruebas exploratorias manuales en [el Capítulo 2](#); el objetivo de este capítulo es capacitarte para realizar pruebas funcionales automatizadas eficaces de aplicaciones web en todas las

capas de la aplicación. Presentaré una estrategia de pruebas funcionales automatizadas que puede dar a tu equipo una respuesta más rápida, y te mostraré cómo utilizar herramientas de automatización y establecer marcos en las distintas capas de la aplicación. El capítulo también ofrece una visión general de las herramientas de IA/ML en el espacio de las pruebas automatizadas, y presenta antipatrones en las pruebas automatizadas a los que debes prestar atención y consejos para paliarlos en las primeras fases. ¿Estás preparado? ¡Vamos a sumergirnos!

Bloques de construcción

Para empezar, permíteme recordar el debate del [Capítulo 1](#), en el que hablamos de que las pruebas deben practicarse tanto a nivel micro como macro de la aplicación para ofrecer una alta calidad. Esto se extiende también a las pruebas funcionales automatizadas.

A la hora de implementar estas pruebas, algunas organizaciones se centran únicamente en las pruebas de macrónivel en las capas superiores de la aplicación, añadiendo cada vez más pruebas funcionales de extremo a extremo basadas en la interfaz de usuario, y pasan por alto totalmente las pruebas de micronivel en las capas inferiores de la aplicación. Por ejemplo, un equipo al que consulté tenía más de 200 pruebas funcionales de extremo a extremo basadas en la interfaz de usuario; el conjunto tardaba 8 horas en ejecutarse cada día, sólo para fallar al final debido a la naturaleza intrínsecamente frágil de las pruebas de macrónivel. Esto es claramente un antipatrón, ya que no sólo va en contra del objetivo de obtener una respuesta rápida mediante la realización de pruebas automatizadas, sino que además no proporciona una respuesta estable. Por eso, los equipos deben incluir pruebas tanto de micronivel como de macrónivel en sus pruebas funcionales automatizadas: las pruebas de micronivel se ejecutan más rápido y son más estables.

Empecemos con una introducción a los distintos tipos de pruebas de nivel micro y macro. Despues, recorreremos algunos ejercicios que te guiarán en su aplicación.

Introducción a los tipos de micropruebas y macropruebas

Mientras hablamos de los distintos tipos de pruebas de , observa cuatro de sus rasgos: el ámbito en el que operan, el propósito que cumplen, su rapidez para dar respuesta y la cantidad de esfuerzo necesario para crearlas y mantenerlas. Esta comprensión fundamental te permitirá adaptar adecuadamente las pruebas automatizadas de tu proyecto (es decir, podrás elegir cuáles utilizar en función de las necesidades de tu proyecto). Para explicar los distintos tipos de pruebas, volveremos a utilizar nuestra hipotética aplicación de comercio electrónico del [Capítulo 2](#).

Como se muestra en [la Figura 3-1](#), la aplicación tiene tres capas: la interfaz de usuario de comercio electrónico, los servicios RESTful (servicios de autenticación, cliente y pedido) y la base de datos (BD). Brevemente, la UI interactúa con los servicios para procesar la información, y los servicios se comunican con la base de datos para almacenar/recuperar la información relevante. La aplicación también se integra con un servicio externo de gestión de la información del producto (PIM) y con los sistemas posteriores (el sistema de gestión de almacenes, etc.) para satisfacer los pedidos. Un flujo de usuario típico en la aplicación sería el siguiente: el usuario introduce sus credenciales en la interfaz de usuario del comercio electrónico, las credenciales se envían al servicio de autenticación para su verificación y, una vez iniciado sesión, el usuario busca productos y realiza pedidos a través de la interfaz de usuario del comercio electrónico. La responsabilidad del servicio de pedidos es recibir los pedidos realizados por el usuario, validar la información del producto con el servicio PIM del proveedor externo, y transmitirla al sistema de gestión de almacenes para activar los procesos de entrega.

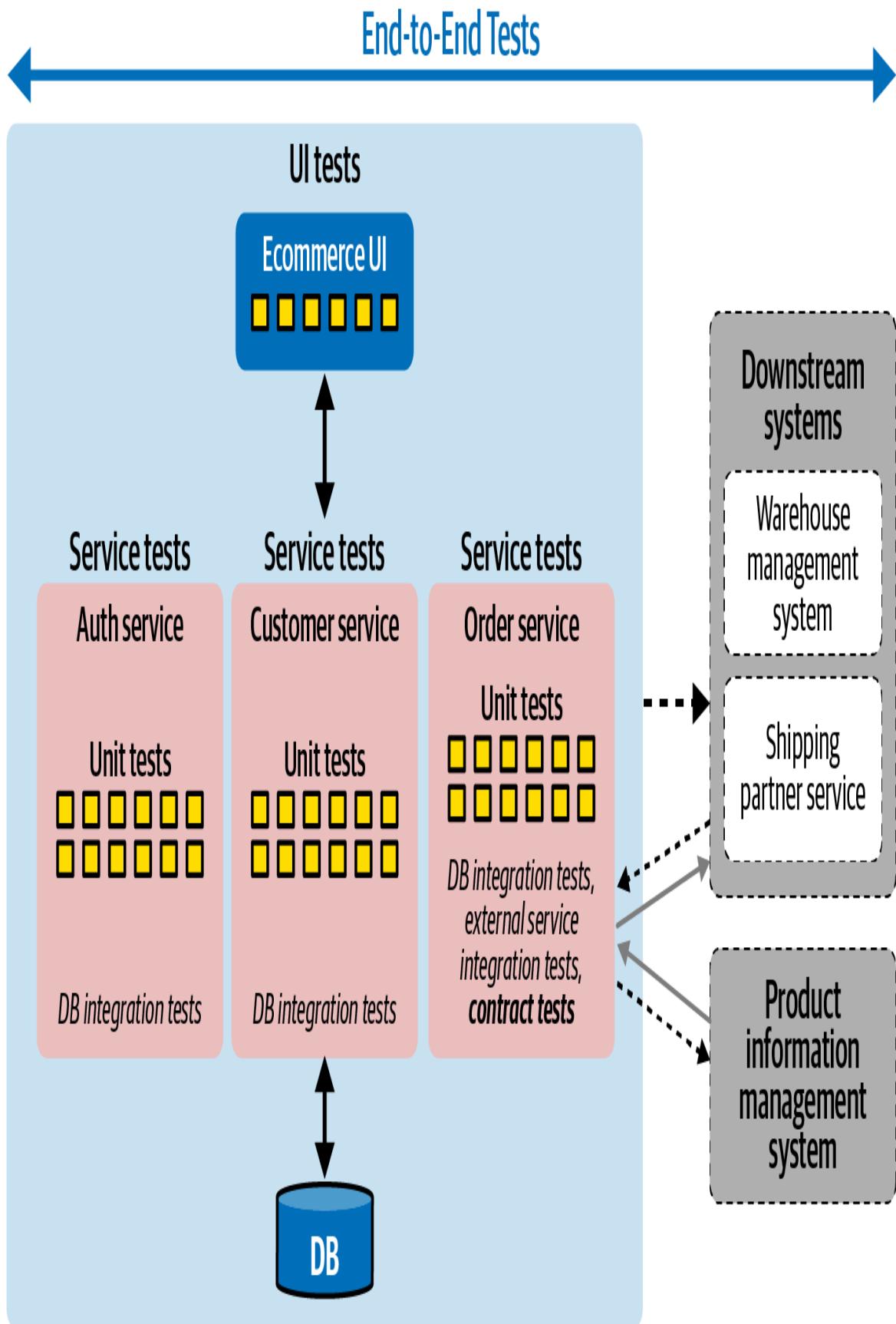


Figura 3-1. Pruebas micro y macro en las capas apropiadas de un ejemplo de aplicación de comercio electrónico orientada a servicios

La Figura 3-1 muestra las distintas pruebas de micro y macrónivel necesarias en las capas apropiadas para satisfacer las necesidades de pruebas funcionales automatizadas de esta aplicación de forma holística. Despleguémoslas una a una.

Pruebas unitarias

Encontrarás pruebas unitarias en todos los servicios de y también en la capa de interfaz de usuario de nuestra aplicación de ejemplo. Estas pruebas pretenden crear redes de seguridad a nivel micro. Validan las partes más pequeñas de la funcionalidad de una aplicación; por ejemplo, una prueba unitaria podría verificar el comportamiento de un método en una clase. Este es el nivel en el que querrás añadir pruebas automatizadas para la mayor parte de la validación básica de entradas.

Supongamos que tenemos un método llamado `return_order_total(item_prices)` en el servicio de pedidos de la aplicación de comercio electrónico, que devuelve el importe total del pedido. A continuación se indican algunas de las pruebas unitarias que se pueden añadir para verificar su comportamiento:

- Devuelve el importe total cuando `item_prices` tenga un valor negativo debido a descuentos.
- Devuelve el importe total cuando `item_prices` está vacío.
- Devuelve el importe total cuando `item_prices` contenga un valor no válido (por ejemplo, un valor que incluya letras, símbolos, etc.).
- Devuelve el importe total cuando los precios de los artículos se envían con símbolos y separadores de moneda diferentes, si la aplicación admite la localización.

- Devuelve un importe total correctamente redondeado con valores decimales fijos.

Las pruebas unitarias residen dentro de la base de código de la aplicación y las escriben los desarrolladores. En los equipos que siguen el desarrollo dirigido por pruebas (TDD), los desarrolladores escriben pruebas unitarias antes que el código de la aplicación, hacen que fallen y luego añaden el código de la aplicación justo para que pasen. Esta práctica ayuda a evitar lógica no deseada y no probada en el código. JUnit, TestNG y NUnit son algunos marcos de pruebas unitarias comúnmente adoptados en el backend. Jest, Mocha y Jasmine son marcos populares de pruebas unitarias en el frontend.

Las pruebas unitarias son las más rápidas de ejecutar. Como residen dentro de la base de código de la aplicación, son fáciles de crear y mantener. Suelen ejecutarse como parte de la fase de creación de la aplicación en la máquina de un desarrollador local, con lo que se consiguen los objetivos de las pruebas por turnos y se proporciona rápidamente y retroalimentación temprana.

Pruebas de integración

En la mayoría de las aplicaciones web medianas y grandes hay bastantes puntos de integración con componentes internos o externos de , como servicios, la interfaz de usuario, bases de datos, cachés, sistemas de archivos, etc., que pueden estar distribuidos a través de los límites de la red y la infraestructura. Para comprobar que todos estos puntos de integración funcionan como se espera de ellos, tienes que escribir pruebas de integración que se ejecuten contra los sistemas de integración reales. El objetivo de estas pruebas debe ser verificar los flujos de integración positivos y negativos, no la funcionalidad detallada de extremo a extremo. En consecuencia, lo ideal es que sean tan pequeñas como las pruebas unitarias.

En el ejemplo de aplicación de comercio electrónico, el servicio de pedidos se integra con componentes internos como la interfaz de usuario de comercio electrónico, la base de datos y otros servicios para intercambiar información. También se integra con el servicio PIM del proveedor externo y con los sistemas posteriores. Tenemos que escribir pruebas de integración para cada servicio a fin de verificar si puede comunicarse correctamente con otros servicios dependientes y con la BD, y ver si hay que añadir pruebas de integración del servicio de pedidos para verificar la integración con esos sistemas y servicios externos en particular.

Las pruebas de integración pueden escribirse utilizando marcos de pruebas unitarias como los mencionados en la sección anterior, junto con herramientas específicas para simular la integración; por ejemplo, JUnit puede utilizarse con [Spring Data JPA](#) para escribir pruebas de integración de BD. Estas pruebas también residen con el código de la aplicación, lo que las hace relativamente fáciles de crear y mantener para los desarrolladores. Su rapidez depende del tiempo que tarde el sistema externo en responder; por tanto, pueden ser más lentas que las pruebas unitarias, que se ejecutan en completo aislamiento.

Pruebas de contrato

Las pruebas de integración pueden no ser factibles si los servicios que las integran también están en desarrollo. Esto suele ocurrir sobre todo en el desarrollo de aplicaciones a gran escala, donde varios equipos trabajan independientemente en diferentes servicios. En tales proyectos, los equipos acuerdan un contrato estándar para cada servicio y trabajan con stubs de los servicios dependientes hasta que están listos. Sin embargo, cuando se utilizan stubs, hay una advertencia: no sabrás si los contratos de los servicios que se integran realmente han cambiado! Si esto ocurre, seguirás construyendo nuevas funciones sobre contratos rotos hasta que lo averigües durante las pruebas de integración reales con servicios

reales, al final del ciclo de desarrollo. Esta es una de las principales razones para tener pruebas de contratos.

Las pruebas de contrato se escriben para validar los stubs frente a los contratos reales de los servicios de integración y para proporcionar información continuamente a ambos equipos a medida que avanzan en el desarrollo. Las pruebas de contrato no comprueban necesariamente los datos exactos devueltos por el servicio integrador, sino que se centran en la propia estructura del contrato. En nuestro ejemplo de aplicación de comercio electrónico, se pueden añadir pruebas de contrato para validar el contrato del servicio PIM del proveedor externo, de modo que cuando cambie, podamos modificar en consecuencia las características del servicio de pedidos. También se pueden escribir pruebas de contrato para las integraciones entre la interfaz de usuario del comercio electrónico y los servicios, si el desarrollo se realiza en paralelo. El flujo de trabajo integral de las pruebas de contratos implica la colaboración entre equipos y se trata en detalle más adelante en el capítulo. Herramientas como Postman y Pact permiten automatizar este flujo de trabajo.

Las pruebas de contrato, en general, se ejecutan muy rápido, ya que su alcance es pequeño (simplemente verifican la estructura del contrato). Residen en la base de código de la aplicación y, por tanto, son relativamente fáciles de crear y mantener para los desarrolladores, aunque no tan sencillas como las pruebas unitarias. La complejidad adicional se debe a la configuración de extremo a extremo, que requiere la colaboración entre equipos.

Pruebas de servicio

Como ya se dijo en [el Capítulo 2](#), las API deben tratarse como productos en sí mismas y probarse a fondo, independientemente del comportamiento de la interfaz de usuario. En esto se centran las pruebas del servicio .

Los servicios manejan esencialmente toda la lógica específica del dominio, como las reglas de negocio, los criterios de error, los mecanismos de reintento, el almacenamiento de datos, etc. Rechazan las solicitudes no válidas tras validar su estructura y formato de valores. Aquí es donde comienzan las pruebas a nivel macro, ya que las pruebas de servicios abarcan las integraciones, los flujos de trabajo del dominio, etc. Por ejemplo, éstas son algunas pruebas de servicio que podríamos añadir a nuestra aplicación de comercio electrónico para el servicio de pedidos:

- Comprueba que sólo un usuario autenticado puede crear un nuevo pedido.
- Comprueba que sólo se crea un pedido si los artículos están disponibles en el momento de la creación.
- Comprueba que se devuelven los códigos de estado HTTP correctos para las entradas positivas y negativas.

Del mismo modo, cada servicio debe tener pruebas de servicio para todos sus puntos finales.

Las pruebas de servicio a veces residen en una base de código independiente, pero para obtener una respuesta rápida es mejor mantenerlas como parte de los propios componentes del servicio. Son algo más complejas de crear y mantener que las pruebas unitarias, ya que implican una configuración real de datos de prueba en la BD. Normalmente, los probadores del equipo son los propietarios de estas pruebas. Se ejecutan más rápido que las pruebas de extremo a extremo basadas en la interfaz de usuario y son algo más lentas que las tres pruebas anteriores de micronivel (pruebas unitarias, de integración y de contrato). Pueden utilizarse herramientas como REST Assured, Karate y Postman para automatizar las pruebas de API.

NOTA

Cualquier entidad que esté bien encapsulada y pueda reutilizarse o sustituirse de forma independiente, como un servicio, se denomina componente. Cuando oigas el término *pruebas de componentes*, puedes pensar, por ejemplo, en las pruebas del servicio .

Pruebas funcionales de la interfaz de usuario

Las pruebas funcionales basadas en la interfaz de usuario se ejecutan en un navegador real e imitan las acciones del usuario en la aplicación. Estas pruebas nos dan información sobre la integración entre múltiples componentes, como los servicios, la interfaz de usuario y la base de datos. Estas pruebas a nivel macro deben centrarse en validar todos los flujos de usuario críticos. Un ejemplo de flujo de usuario crítico en la aplicación de comercio electrónico es buscar un producto, añadirlo al carrito, pagarla y obtener una confirmación del pedido. Esto puede añadirse como prueba funcional de la interfaz de usuario. Al escribir estas pruebas, evita validar de nuevo los mismos detalles cubiertos como parte de las micropruebas de nivel inferior, ya que esto será redundante y aumentará su tiempo de ejecución. Por ejemplo, la comprobación de los totales del pedido para diferentes combinaciones de precios de los artículos debe cubrirse mediante pruebas unitarias y no es necesario validarla de nuevo como parte de una prueba funcional de interfaz de usuario.

Las pruebas funcionales de la interfaz de usuario suelen mantenerse separadas del código de la aplicación, como una base de código independiente. Dependen principalmente del probador, aunque pueden ser de propiedad conjunta con los desarrolladores. Estas pruebas tardan más en ejecutarse y tienden a ser frágiles, ya que dependen de que el comportamiento de toda la pila de la aplicación, incluida la infraestructura, la red, etc., sea estable. Además, requieren un esfuerzo de mantenimiento considerable en comparación con otros tipos de pruebas, ya que los fallos pueden

producirse en cualquier parte de toda la aplicación; por ejemplo, un cambio en el ID de un elemento, un retraso en la carga de la página o la indisponibilidad de los servicios debido a problemas del entorno.

Herramientas como Selenium y Cypress se adoptan popularmente para escribir pruebas de interfaz de usuario automatizadas.

Encontrarás ejercicios para ambas más adelante en el capítulo.

CONSEJO

Cada vez que pienses en añadir una prueba funcional de interfaz de usuario, primero cuestiona la intención de la prueba (por ejemplo, validar la entrada, las reglas de negocio a nivel de servicio, etc.) y comprueba si puedes escribir micropruebas de nivel inferior para conseguir el mismo objetivo.

Pruebas de extremo a extremo

Como su nombre indica, las pruebas de extremo a extremo deben validar toda la amplitud del flujo de trabajo de tu dominio, incluidos los sistemas descendentes. En la aplicación de comercio electrónico, después de realizar un pedido en el sitio web, los sistemas descendentes (como el sistema de gestión de almacenes, los servicios de socios de envío de terceros, etc.) realizan realmente el pedido. Este flujo de dominio de extremo a extremo debe probarse para una integración adecuada.

Dependiendo del contexto de la aplicación, las pruebas funcionales de la interfaz de usuario suelen convertirse en pruebas de extremo a extremo. Si no es así, crea pruebas de extremo a extremo separadas utilizando una combinación de herramientas de pruebas de interfaz de usuario, servicios y bases de datos para cubrir todo el flujo de integración. Obviamente, estas pruebas tardan más en ejecutarse y requieren más cuidado en su mantenimiento, ya que necesitan un entorno estable y una configuración de datos de prueba en varios sistemas. La intención de estas pruebas es determinar si todos los

componentes están integrados correctamente de extremo a extremo, y no probar las funcionalidades de los componentes. Por tanto, puedes limitarte a unas pocas pruebas que activen todos tus componentes.

NOTA

Una práctica comúnmente adoptada es que los desarrolladores escriban todas las pruebas de micronivel durante el desarrollo y los probadores escriban las pruebas de macronivel como parte de la fase de prueba. Sin embargo, esto depende en gran medida de las habilidades disponibles en el equipo y puede variar de un equipo a otro.

Con esto, hemos recorrido todos los tipos de micropruebas y macropruebas, y deberías conocer sus cuatro rasgos esenciales. En la siguiente sección se analiza una estrategia de pruebas funcionales automatizadas ampliamente adoptada por los equipos de software. Puedes utilizarla como base para definir una estrategia que se adapte a las necesidades específicas de tu proyecto.

Estrategia de pruebas funcionales automatizadas

Una estrategia de una sola línea que puede aplicarse a las pruebas automatizadas es: *añade pruebas para validar el alcance correcto de la funcionalidad en las capas correctas de la aplicación, de forma que proporcionen la información más rápida al equipo!* Mike Cohn lo cristalizó muy bien con una pista visual en su libro de 2009 *Succeeding with Agile* (Addison-Wesley Professional) como el concepto de *pirámide de pruebas*. La pirámide de pruebas recomienda tener un amplio cubo de pruebas de micronivel e ir reduciendo gradualmente el número de pruebas de macronivel a medida que aumenta su alcance. Por ejemplo, si tienes 10x pruebas unitarias y de integración, deberías tener 5x pruebas de servicio y

sólo x pruebas orientadas a la interfaz de usuario. Si las superpones, con las pruebas unitarias en la base, forman una pirámide. La razón obvia de esta recomendación es que, a medida que aumenta el alcance de las pruebas, tardan más en ejecutarse y cuesta más escribirlas y mantenerlas.

NOTA

Hay otras formas de pruebas de automatización aparte de la pirámide, como [el panal y el trofeo de pruebas](#). Esencialmente, todas ellas enfatizan el mismo principio: que las pruebas de micronivel son más fáciles de escribir y ejecutar que las pruebas de macronivel. Si exploras esas formas de prueba, presta atención a lo que definen como alcance de cada uno de los tipos de prueba. Las formas cambian a medida que cambia el alcance de las pruebas.

Una pirámide de pruebas típica para una aplicación web orientada a servicios , como la de nuestro ejemplo de comercio electrónico, podría parecerse a [la Figura 3-2](#).

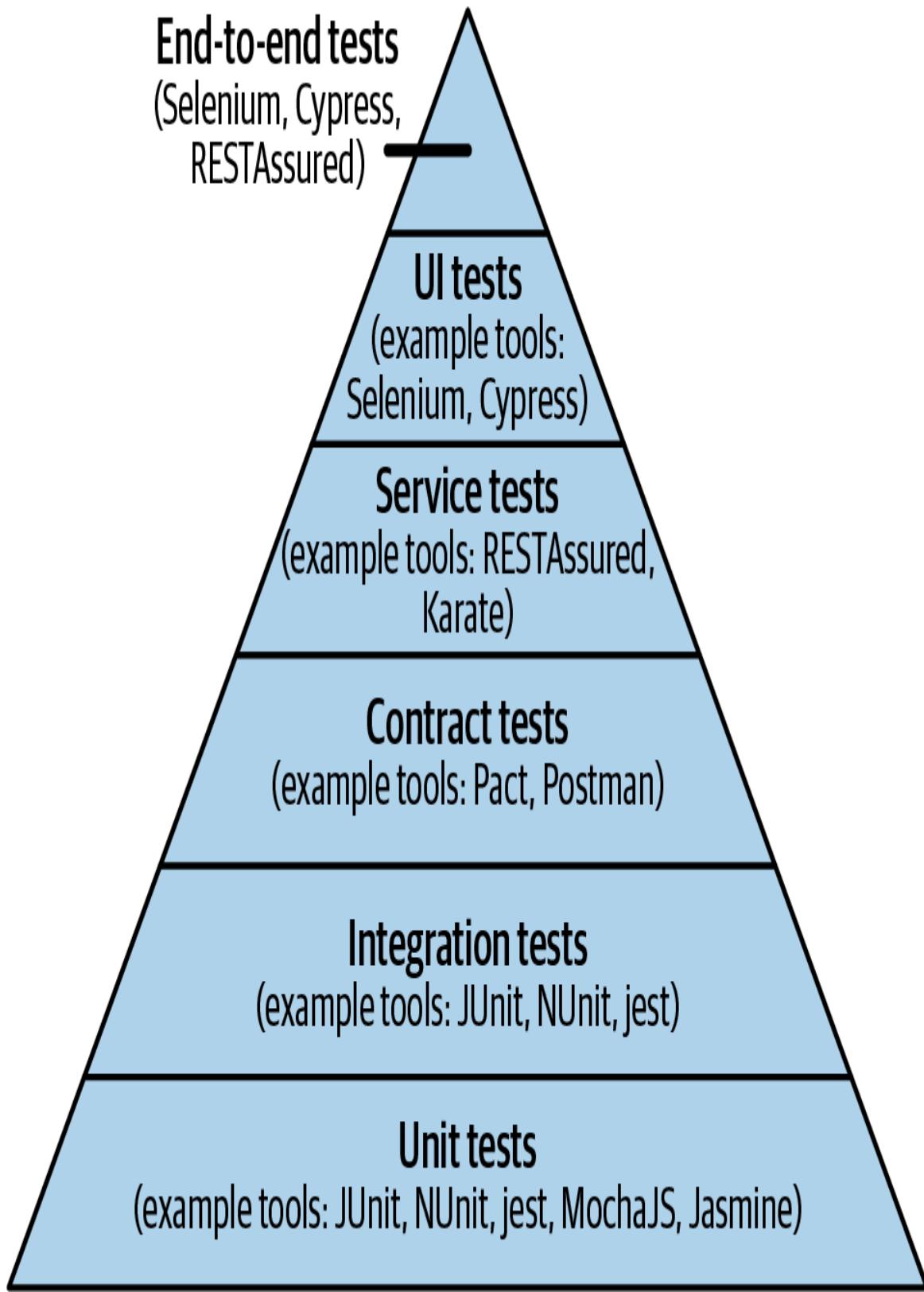


Figura 3-2. Pirámide de pruebas de una aplicación web orientada a servicios

He visto funcionar la pirámide de pruebas en la práctica, al igual que muchos otros profesionales de las pruebas. Un ejemplo digno de mención que puedo citar es que, después de que transformáramos el proyecto mencionado anteriormente, que tenía más de 200 pruebas de extremo a extremo basadas en la interfaz de usuario, para que se adhiriera a la pirámide de pruebas, el equipo pudo obtener información en 35 minutos desde la confirmación del código con ~470 pruebas!

NOTA

Aunque la pirámide de pruebas puede considerarse el ideal a perseguir, puede haber situaciones en las que no sea posible lograr una verdadera forma piramidal. Esto puede deberse a dificultades prácticas, como la falta de un entorno de pruebas totalmente desplegado para soportar las pruebas de extremo a extremo, o la falta de herramientas para automatizar determinados tipos de funcionalidades (como el escaneado de códigos de barras), o, directamente, la falta de habilidades. En tales casos, el equipo debe ser consciente de las concesiones que está haciendo y elegir la cantidad y los tipos de pruebas de tal manera que puedan seguir logrando el objetivo de obtener una respuesta rápida, a pesar de estas limitaciones.

Otra parte de la estrategia de automatización debe ser disponer de una forma de realizar un seguimiento de la cobertura de la automatización para garantizar que no haya retrasos. Para ello se pueden adoptar herramientas de gestión de pruebas como TestRail, herramientas de gestión de proyectos como Jira, o algo tan sencillo como una hoja de Excel. El seguimiento de la cobertura de automatización es esencial. Por diversas razones, muchos equipos omiten (o dejan de lado) los esfuerzos de automatización del alcance de la historia de usuario, lo que conduce a una retroalimentación retrasada e incompleta. Esto puede hacer que pierdan la confianza en la propia suite de automatización. Realizar un seguimiento de todos los casos de prueba y asegurarse de que están automatizados puede

ayudar a evitarlo. Una práctica ideal, que muchos equipos ágiles siguen, es dar por "terminada" una historia de usuario sólo si todas sus pruebas de nivel micro y macro están automatizadas.

Ejercicios

Una vez explorados todos los tipos de pruebas, es hora de flexionar los músculos de la codificación. Los ejercicios de esta sección te ayudarán a empezar a configurar un marco de pruebas funcionales en tres de las capas de la aplicación: Te mostraré cómo implementar pruebas funcionales basadas en la interfaz de usuario utilizando Selenium y Cypress, pruebas de servicios utilizando REST Assured y pruebas unitarias con JUnit. ¡Vamos a empezar!

LA PILA TECNOLÓGICA DE AUTOMATIZACIÓN DE PRUEBAS

He aquí algunos consejos que debes tener en cuenta al definir tu pila tecnológica de automatización:

- Lo mejor es que sea similar a la pila tecnológica de desarrollo, para que los miembros de tu equipo no tengan que aprender un conjunto de herramientas totalmente nuevo. He observado que cuando los equipos tienen pilas tecnológicas diferentes para el desarrollo y las pruebas, existe una resistencia natural entre los desarrolladores a apropiarse de las pruebas, lo que obstaculiza los objetivos de las pruebas por turnos y de obtener una respuesta más rápida.
- Evita el impulso de arrastrar las pruebas de todas las capas a una base de código común. Mantén las pruebas de cada capa dentro de sus respectivos componentes, de modo que las pruebas se envíen cuando se reutilicen los componentes. Esto implica que las opciones de pila tecnológica en cada capa dependerán de la pila tecnológica de desarrollo del componente respectivo.

Pruebas funcionales de interfaz de usuario

Selenium WebDriver y Cypress son dos herramientas populares de que ayudan a crear pruebas automatizadas funcionales basadas en la interfaz de usuario. Puedes escribir pruebas Selenium WebDriver con muchos lenguajes de programación, como Java, C#, Python, JavaScript, etc. Aunque las pruebas de Cypress sólo se pueden escribir en JavaScript, esta herramienta está repleta de muchas ventajas, como verás. Te mostraré cómo utilizar ambas, para que, en función de las preferencias de lenguaje de programación de tu

equipo y de la pila tecnológica de desarrollo, puedes elegir la herramienta adecuada.

Marco Java-Selenium WebDriver

Empecemos con un ejercicio para crear un marco de automatización utilizando Java y [Selenium WebDriver](#).

Requisitos previos

Como requisitos previos, necesitarás instalar en las siguientes herramientas esenciales:

- La última versión de [Java](#)
- El entorno de desarrollo integrado (IDE) de tu [elección-IntelliJ](#) es un IDE de uso común para Java
- El [navegador Chrome](#)

También tendrás que instalar algunas otras herramientas, como se describe en los siguientes apartados.

Maven

Apache Maven es una herramienta de automatización de la construcción. Construye Las herramientas de automatización ayudan esencialmente a estandarizar los procesos de gestión de dependencias y los pasos de construcción del proyecto. Para profundizar más, en muchos proyectos es necesario utilizar algunas bibliotecas y plug-ins de terceros para crear nuevas funcionalidades. Es crucial que todos los miembros del equipo utilicen las mismas versiones de las bibliotecas y plug-ins, y que sigan la misma secuencia de pasos para crear artefactos de aplicación (es decir, construir el proyecto, ejecutar las pruebas, etc.). Las herramientas de automatización de la compilación ayudan a conseguir estos objetivos. Maven y Gradle son dos opciones populares para construir aplicaciones Java. Para este ejercicio utilizaremos Maven; [descárgalo ahora](#) y sigue las instrucciones de instalación del sitio web.

Para comprender rápidamente cómo funciona Maven, echemos un vistazo en al archivo XML del Modelo de Objetos del Proyecto (POM), *pom.xml*. Aquí es donde defines todas las bibliotecas dependientes, los plug-ins y sus versiones, como se ve en el [Ejemplo 3-1](#), y Maven lo lleva adelante a partir de ahí.

Ejemplo 3-1. Un archivo pom.xml de ejemplo

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                               http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>SeleniumJavaExample</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>15</maven.compiler.source>
        <maven.compiler.target>15</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>4.0.0</version>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>7.4.0</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Éstas son las cosas importantes que hay que observar en este archivo:

- Los atributos `groupId`, `artifactId`, y `version` definen las coordenadas del proyecto que permiten a Maven realizar un seguimiento del proyecto a lo largo del tiempo. Al crear un nuevo proyecto en IntelliJ, puedes añadir estos detalles.
- La sección de propiedades declara la versión de Java que se utilizará para la compilación. También puedes incluir aquí variables específicas del proyecto que quieras reutilizar en otras secciones del archivo `pom.xml` o en el código de la aplicación.
- La sección de dependencias es donde se enumeran las bibliotecas dependientes y sus versiones. Observa que el archivo hace referencia a Selenium 4.0 como dependencia. Maven mantiene un repositorio central de todas las bibliotecas y sus distintas versiones, desde el que las extrae a tu máquina basándose en la configuración de esta sección del archivo `pom.xml`. Esta forma centralizada de gestionar las bibliotecas garantiza que todos los miembros del equipo tengan exactamente los mismos binarios de todas las bibliotecas. Para añadir una dependencia para una biblioteca en tu archivo `pom.xml`, busca la biblioteca en el **Repositorio Maven** y copia sus valores de coordenadas en la sección de dependencias .

Del mismo modo, puedes configurar plug-ins, entornos, etc. declarándolos como atributos adecuadamente en tu archivo `pom.xml`, tal y como se describe en [la documentación](#) de Maven. Utilizarás el archivo del [Ejemplo 3-1](#) para escribir tus pruebas Selenium WebDriver.

Maven también proporciona comandos del ciclo de vida de construcción que son necesarios para crear artefactos de aplicación. Entre los comandos que utilizarás con frecuencia se incluyen:

`mvn compile`

Compila el código del proyecto

```
mvn clean
```

Limpia (es decir, elimina) los artefactos creados anteriormente

```
mvn test
```

Ejecuta las pruebas escritas utilizando marcos de pruebas (a continuación configuraremos uno de ellos)

También hay otros comandos de Maven para instalar, implementar, etc., para completar todo el ciclo de vida de la aplicación creación de artefactos.

TestNG

TestNG es un marco de pruebas, también conocido como *corredor de pruebas*. JUnit es otro popular marco de pruebas en Java. Los marcos de pruebas, en general, ofrecen la posibilidad de crear pruebas, añadir aserciones, añadir tareas de configuración y desmontaje, organizar pruebas en grupos, ejecutar pruebas y presentar un resumen de la ejecución de la prueba. TestNG puede utilizarse para todo tipo de pruebas: unitarias, de integración y de extremo a extremo. Para instalarlo, sólo tienes que añadirlo como dependencia en tu archivo *pom.xml*, como se ve en el [Ejemplo 3-1](#).

Algunas de las funciones destacadas de TestNG que puedes utilizar con regularidad son las siguientes:

- **@Test**: Una anotación para indicar el método de una clase como método de prueba para que TestNG lo ejecute. Por tanto, toda prueba debe ir precedida de esta anotación.
- **@BeforeClass, @AfterClass, @BeforeMethod, @AfterMethod, @BeforeSuite, @AfterSuite**: Como su nombre indica, se ejecutan antes o después de las clases de prueba, de los métodos de prueba o de todo el conjunto. Los métodos de

instalación y desinstalación de tus pruebas se pueden anotar con estas etiquetas.

- `assertEquals()`, `assertTrue()`, y otros métodos de aserción para realizar validaciones dentro de los testsIntelliJ te guía en la sintaxis de estos métodos mientras creas tests.

Selenium WebDriver

Jason Huggins inventó originalmente Selenium, una popular herramienta de automatización de pruebas de código abierto , en 2004, y ha pasado por muchas encarnaciones desde entonces.

Puedes leer sobre la fascinante [historia y evolución de la herramienta](#) en su sitio web, y una comunidad de código abierto extremadamente vibrante sigue apoyándola hoy en día.

¿POR QUÉ SE LLAMA SELENIO?

El selenio, elemento químico, se utiliza como antídoto contra el envenenamiento por mercurio. Antes de Selenio, la herramienta de pruebas automatizadas, la herramienta más popular utilizada para las pruebas automatizadas se llamaba Mercurio. ¿Entiendes el chiste?

Selenium WebDriver facilita principalmente la interacción con la aplicación web renderizada en el navegador. No sirve para ningún otro propósito, como aserciones, generación de informes, etc., por lo que necesitamos otras herramientas como TestNG y Maven para completar el marco de automatización.

Selenium WebDriver tiene tres componentes básicos:

APIs

Estos son los métodos que permiten interactuar con los elementos de la aplicación en el navegador (hacer clic, escribir en los campos, etc.).

Biblioteca de clientes

La biblioteca cliente de Selenium WebDriver agrupa las API para que las utilicemos en nuestro conjunto de pruebas. Las bibliotecas cliente están disponibles en muchos lenguajes de programación.

Conductor

Es el componente que indica al navegador que realice las acciones dictadas por la API. Los controladores suelen ser creados y mantenidos por los propios navegadores respectivos y no forman parte del paquete de distribución de Selenium. Por ejemplo, si quieres ejecutar las pruebas contra Chrome, tienes que descargar el ChromeDriver por separado e incluirlo en tus scripts de automatización.

Conozcamos primero las distintas API que proporciona Selenium WebDriver. En [el Ejemplo 3-2](#) se enumeran algunos de los métodos WebDriver más utilizados para encontrar distintos elementos en la aplicación. Selenium identifica los elementos de una página web basándose en los valores de sus atributos HTML, como `id`, `className`, `cssSelector`, etc. Puedes seleccionar la opción Inspeccionar del menú contextual para obtener estos valores en Chrome. Prueba a inspeccionar el cuadro de texto de búsqueda de Amazon, y verás que el ID es "twotabsearchtextbox".

Ejemplo 3-2. Algunos métodos WebDriver de uso común para encontrar elementos

```
// find element by ID  
driver.findElement(By.id("login"))  
  
// find element by CSS selector  
driver.findElement(By.cssSelector("#login"));  
  
// find element by class name  
driver.findElement(By.className("login-card"));
```

```
// find element by XPath  
driver.findElement(By.XPath("//@login"));  
  
// find multiple elements  
driver.findElements(By.cssSelector("#username li"));
```

CONSEJO

El `id` es único para cada elemento de una página. Por lo tanto, es el tipo de localizador preferido para mantener estables tus pruebas. Los selectores CSS y los localizadores XPath tienden a romperse cuando la aplicación sufre cambios frecuentes.

Selenium WebDriver también proporciona formas avanzadas de encontrar elementos en la página utilizando **localizadores relativos**, como especificar que son `above`, `below` o `toLeftOf` otro elemento.

Una vez que encontramos los elementos, necesitamos interactuar con ellos. En [el Ejemplo 3-3](#) se enumeran algunos métodos de Selenium WebDriver de uso frecuente para realizar distintas acciones sobre los elementos.

Ejemplo 3-3. Algunos métodos WebDriver de uso común para interactuar con elementos web

```
// click an element  
driver.findElement(By.id("submit")).click();  
  
// type text into an input box  
driver.findElement(By.cssSelector("#username")).sendKeys(username);
```

También puedes utilizar la **clase Acciones** en WebDriver para interacciones más avanzadas como `keyDown`, `contextClick` y `dragAndDrop`.

Aparte de los métodos para interactuar con los elementos de la aplicación, WebDriver también proporciona métodos para gestionar el comportamiento del navegador, como abrir una URL, volver atrás,

cerrar el navegador, establecer el tamaño de la ventana del navegador, establecer cookies en el navegador, cambiar entre varias pestañas, etc. **El Ejemplo 3-4** muestra algunos de estos métodos de manipulación del navegador de uso común.

Ejemplo 3-4. Algunos métodos WebDriver de uso común para manipular el comportamiento del navegador

```
// open a URL  
driver.get("https://example.com");  
  
// browser back, forward, and refresh  
driver.navigate().back();  
driver.navigate().forward();  
driver.navigate().refresh();  
  
// open browser in iPad size  
driver.manage().window().setSize(new Dimension(768, 1024));  
  
// close browser  
driver.close();  
  
// quit the driver session  
driver.quit();
```

Al navegar por las páginas, tienes que hacer que la prueba espere a que la página se cargue o a que un elemento sea visible después de que la página se cargue. Algunos equipos utilizan sentencias de espera codificadas para hacer que la prueba espere, pero hacen que las pruebas sean frágiles, ya que los tiempos de carga de la página pueden ser diferentes en distintos entornos. WebDriver ofrece algunas estrategias de espera incorporadas para superar este problema:

- La estrategia de espera *implícita* hace que WebDriver sondee el Modelo de Objetos del Documento (DOM), que representa el contenido completo de un documento HTML, durante x segundos, esperando a que aparezca el elemento. El comportamiento predeterminado de WebDriver es esperar 0

segundos; puedes cambiar esto y utilizar la espera implícita durante la etapa de inicialización del controlador para establecer un tiempo de espera estándar.

- La estrategia de espera *explícita* hace que WebDriver espere hasta x segundos a que se cumpla una condición esperada.
- La opción *de* espera *fluida* da más flexibilidad a la hora de definir una estrategia de espera. Hace que WebDriver espere un máximo de x segundos a que se cumpla una condición esperada, comprobando si la condición se cumple cada y segundos.

El Ejemplo 3-5 muestra estos diferentes métodos de espera.

Ejemplo 3-5. Estrategias de espera de WebDriver

```
// Implicit wait for 10 seconds before timeout exception
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

// Explicit wait for 10 seconds until the submit button becomes clickable
WebElement submitButton = new WebDriverWait(driver,
Duration.ofSeconds(10)).
until(ExpectedConditions.elementToBeClickable(By.id("submit")));

// Fluent wait, which polls every 1 second up to a maximum of 3 seconds
// and waits
// for the spinner to disappear
FluentWait<WebElement> wait = new FluentWait<WebElement>(driver)
    .withTimeout(Duration.ofSeconds(3))
    .pollingEvery(Duration.ofSeconds(1))
    .ignoring(NoSuchElementException.class);
wait.until(ExpectedConditions.invisibilityOf(driver.findElement(By.id("inner"))));
```

Estos son los métodos WebDriver más utilizados para ayudar a escribir pruebas cotidianas. WebDriver también ofrece muchas interacciones más avanzadas, como escuchar eventos y realizar distintas acciones en función del tipo de evento, interactuar con ventanas modales y casi cualquier otra cosa que quieras probar en un navegador. Selenium 4 también permite burlarse de las

respuestas del servidor y depurar utilizando el [protocolo Chrome DevTools](#). Si quieres explotar estas capacidades avanzadas, consulta el [sitio web](#) para obtener más detalles.

Modelo de objetos de página

El Modelo de Objetos de Página es el patrón de diseño más comúnmente adoptado por para un marco de automatización basado en la interfaz de usuario. Implica recrear la estructura de la aplicación tal y como es en el marco de automatización; es decir, creas una clase de página para cada página de tu aplicación y defines los elementos y acciones de la página en esa clase. El patrón ha demostrado ser fructífero, ya que permite la abstracción y la encapsulación, y por tanto facilita la solución de problemas o la incorporación de nuevos cambios. Por ejemplo, cuando cambia el ID de un elemento, sabes dónde encontrarlo (en su clase de página) y arreglarlo. Si no tienes esa abstracción, tendrás que hacer el cambio de ID en todas las pruebas explícitamente.

[El Ejemplo 3-6](#) muestra un ejemplo de clase LoginPage con tres elementos: un campo de nombre de usuario, un campo de contraseña y un botón de inicio de sesión. También tiene un método login(email, password) para realizar la acción de inicio de sesión en la página. Nos referiremos a esta clase LoginPage más adelante cuando creemos una prueba.

Ejemplo 3-6. La clase LoginPage utilizando el Modelo de Objetos de Página

```
// LoginPage.java

package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class LoginPage {

    private WebDriver driver;
```

```

private By emailID = By.id("user_email");
private By passwordField = By.id("user_password");
private By signInButton =
    By.cssSelector("input.gr-button.gr-button--large");

public LoginPage(WebDriver driver) {
    this.driver = driver;
}

public HomePage login(String email, String password){
    driver.findElement(emailID).sendKeys(email);
    driver.findElement(passwordField).sendKeys(password);
    driver.findElement(signInButton).click();
    return new HomePage(driver);
}
}

```

Del mismo modo, tu marco de automatización debe clases de página que representen todas las páginas de tu aplicación.

Configuración y flujo de trabajo

Una vez explorados todos los componentes necesarios para un marco de automatización de la interfaz de usuario típico de Java-Selenium WebDriver, el siguiente paso de es reunirlos y escribir la primera prueba para un flujo de usuario sencillo: iniciar sesión en tu aplicación de comercio electrónico favorita (elige una en la que tengas una cuenta) y hacer una aserción en el título de la página de inicio. Sigue los pasos que se indican aquí para crear esta prueba:

1. Abre IntelliJ y crea un nuevo proyecto Maven seleccionando Archivo → Nuevo → Proyecto → Maven.
2. Selecciona la versión de Java que has descargado.
3. Pasa a la siguiente ventana para introducir el nombre de tu proyecto, la ubicación, groupId, y artifactID, como se ve en la **Figura 3-3**.

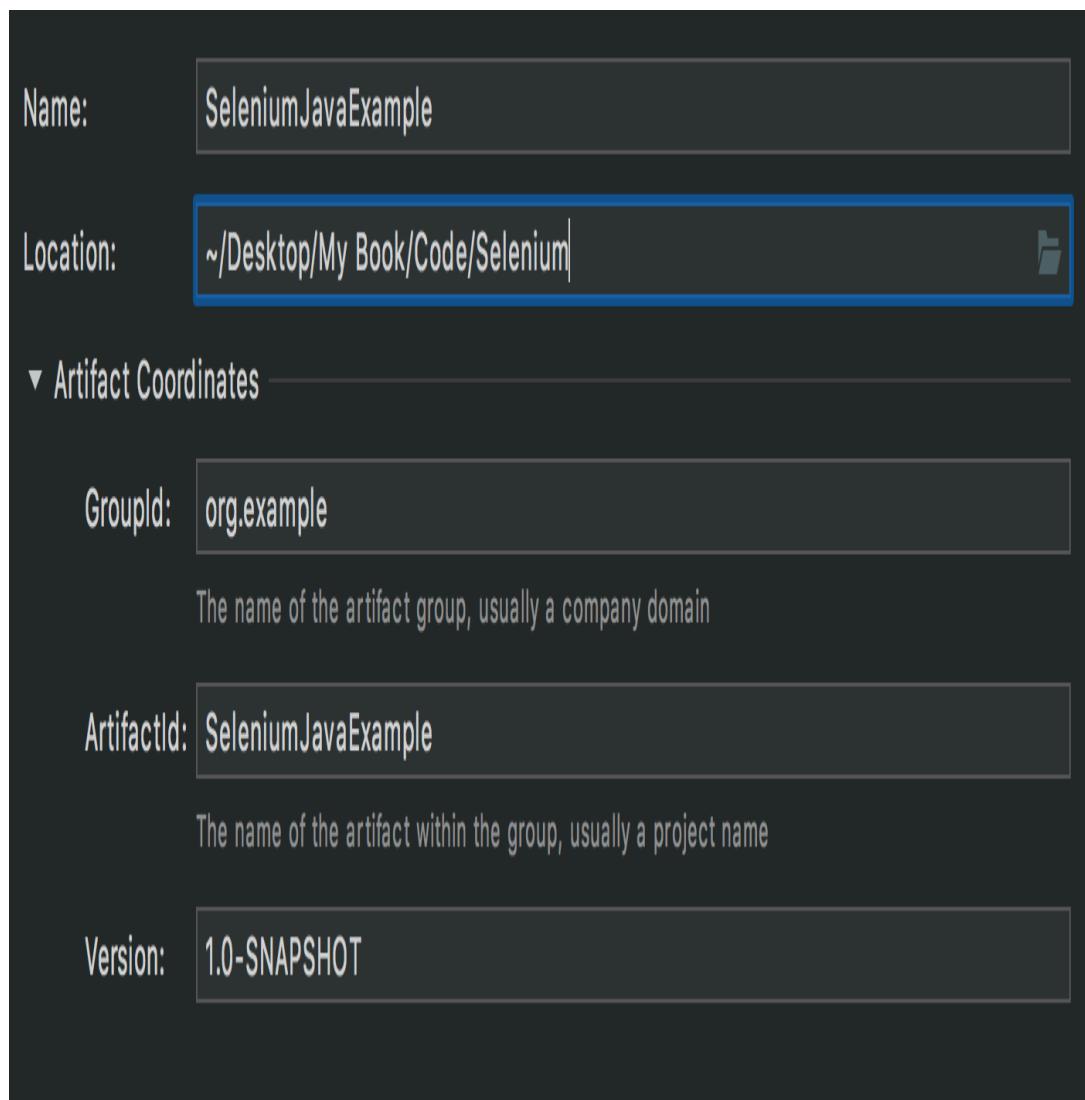


Figura 3-3. Crear un nuevo proyecto Maven en IntelliJ

Estos tres pasos configurarán una estructura inicial del proyecto, como se muestra en el [Ejemplo 3-7](#).

Ejemplo 3-7. Estructura inicial del proyecto Maven

```
└── SeleniumJavaExample.iml
└── pom.xml
└── src
    ├── main
    │   ├── Java
    │   └── resources
    └── test
        └── Java
```

4. Descarga el **ejecutable ChromeDriver** compatible con tu versión local del navegador Chrome (para ver tu versión de Chrome, selecciona Chrome → Acerca de Chrome).
5. Coloca el archivo ejecutable en la carpeta *src/main/resources* dentro de tu proyecto.
6. Añade las dependencias del proyecto, Selenium, Java y la biblioteca TestNG, como en el **Ejemplo 3-1**. En IntelliJ, el panel de Maven está a la derecha; puedes utilizarlo para actualizar y descargar las bibliotecas inmediatamente.
7. Crea un paquete llamado `base` en *src/test/java*.
8. Añade un nuevo archivo de clase llamado *BaseTests.java*, donde podrás definir la configuración del WebDriver como se ve en el **Ejemplo 3-8**.

Ejemplo 3-8. La clase BaseTests

```
// BaseTests.java

package base;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;

import java.time.Duration;

public class BaseTests {

    protected WebDriver driver;

    @BeforeMethod
    public void setUp(){
        System.setProperty("webdriver.chrome.driver",
                           "src/main/resources/chromedriver");
        driver = new ChromeDriver();

        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
    }
}
```

```

        driver.get("http://eCommerce.com/sign_in");
    }

    @AfterMethod
    public void teardown(){
        driver.quit();
    }
}

```

El método `setUp()` hace unas cuantas cosas: proporciona la ruta ejecutable de ChromeDriver, instancia el objeto `ChromeDriver`, define la espera implícita en 10 segundos y abre la URL de la aplicación utilizando el objeto `driver`. El método `tearDown()` realiza una tarea: cierra la sesión del navegador tras la ejecución de la prueba. Observa las anotaciones de TestNG `@BeforeMethod` y `@AfterMethod` utilizadas para crear y eliminar una nueva sesión de controlador, que se ejecutará para cada prueba.

9. A continuación, crea un nuevo paquete llamado `tests` en `src/test/java` y añade tu primera clase de prueba, por ejemplo, `LoginTest`, como en el [Ejemplo 3-9](#). Puedes ver la anotación `@Test` y el método `assertEquals()` proporcionados por TestNG.

Ejemplo 3-9. La clase `LoginTest` con la primera prueba

```

// LoginTest.java

package tests;

import base.BaseTests;
import org.testng.annotations.Test;
import pages.LoginPage;
import static org.testng.Assert.*;

public class LoginTest extends BaseTests {

    @Test
    public void verifySuccessfulLogin(){
        LoginPage loginPage = new LoginPage(driver);
        assertEquals(loginPage.login("example@gmail.com",

```

```

        "Admin123").getTitle(), "Home page");
    }
}

```

10. Despues de crear las pruebas, tendrás que crear tus clases de página. Crea un nuevo paquete llamado pages en *src/main/java* y añade allí tus clases de página. Consulta [el Ejemplo 3-6](#) para la clase LoginPage y [el Ejemplo 3-10](#) para la clase HomePage.

Ejemplo 3-10. La clase HomePage

```

// HomePage.java

package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import java.time.Duration;

public class HomePage {

    private WebDriver driver;
    private By searchField = By.cssSelector("input.searchBox");

    public HomePage(WebDriver driver) {
        this.driver = driver;
    }

    public String getTitle(){
        WebDriverWait wait = new WebDriverWait(driver,
            Duration.ofSeconds(10));
        wait.until(ExpectedConditions.
            presenceOfElementLocated(searchField));
        return driver.getTitle();
    }
}

```

Las clases de página tendrán los métodos de Selenium WebDriver que hemos comentado antes para encontrar e interactuar con los elementos de estas clases. Además, observa

cómo se encadenan las clases de página para devolver los objetos de las otras páginas. Por ejemplo, el método `login()` de la clase `LoginPage` devuelve un objeto `HomePage` junto con el objeto `driver`. Recuerda también que las aserciones no pertenecen a las clases de página.

11. Ahora puedes ejecutar la prueba desde el propio IDE haciendo clic con el botón derecho del ratón en el triángulo verde situado junto a la etiqueta `@Prueba`, o ejecutarla desde la línea de comandos utilizando Maven como se indica a continuación:

```
$ mvn clean test
```

Este comando abrirá el navegador Chrome y ejecutará tu prueba. También crea un informe HTML en `target/surefire-reports/index.html`, como se ve en la [Figura 3-4](#).

1 suite

All suites



✓ tests.LoginTest

Command line suite

verifySuccessfulLogin

Info

- [unset file name]
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results

- 1 method, 1 passed
- Passed methods (hide)
 - ✓ verifySuccessfulLogin

Figura 3-4. Informe HTML generado por el complemento Maven Surefire

Enhorabuena, ¡has creado y ejecutado con éxito tu primera prueba!

Esta prueba pasó, pero cuando haya fallos en CI, tener capturas de pantalla de esos fallos será muy útil para la depuración. Para hacer capturas de pantalla en los fallos, modifica tu método `teardown()` como se muestra en el [Ejemplo 3-11](#). Crea una carpeta llamada *capturas de pantalla* bajo `src/main/resources`, y las capturas de pantalla de los fallos se colocarán allí.

Ejemplo 3-11. Hacer capturas de pantalla en caso de fallo

```
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.testng.ITestResult;
import java.io.File;
import java.io.IOException;
import com.google.common.io.Files;

@AfterMethod
public void teardown(ITestResult result){
    if(ITestResult.FAILURE == result.getStatus()) {
        var camera = (TakesScreenshot) driver;
        File screenshot = camera.getScreenshotAs(OutputType.FILE);
        try {
            Files.move(screenshot,
                new File("src/main/resources/screenshots/" +
                    result.getName() + ".png"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    driver.quit();
}
```

Puedes añadir más capacidades a tu marco de automatización en función de las necesidades específicas del proyecto. Por ejemplo, puedes ejecutar tus pruebas en paralelo utilizando las capacidades de [TestNG](#) o [Selenium Grid](#), agrupar las pruebas y ejecutarlas en varios navegadores utilizando las capacidades de [TestNG](#), incluir un

marco de desarrollo basado en el comportamiento (BDD) como **Cucumber**, etc. Sin embargo, recuerda siempre reducir al mínimo tus pruebas de interfaz de usuario .

DESARROLLO ORIENTADO AL COMPORTAMIENTO

BDD es una práctica de desarrollo de software que pretende acercar el negocio a los miembros del equipo técnico. Por ejemplo, los marcos BDD como Cucumber proporcionan facilidades para escribir pruebas en lenguaje natural, parecidas a una típica historia de usuario con la **estructura Dado, Cuándo, Entonces**. Este permite a la gente del negocio pasar requisitos como pruebas fallidas y a la gente técnica empezar a construir características corrigiendo las pruebas fallidas.

Framework JavaScript-Cypress

Cypress se lanzó en 2014, 10 años después de que se introdujera por primera vez Selenium, y ha conseguido una amplia adopción como herramienta de automatización de la interfaz de usuario de extremo a extremo. Las pruebas de Cypress sólo pueden escribirse en JavaScript, a diferencia de las pruebas de Selenium. A pesar de esa limitación, se ha hecho popular gracias a algunas de las siguientes características destacadas:

- La arquitectura de Cypress es tal que no ejecuta los comandos a través de la red como hace Selenium, sino que los ejecuta dentro del mismo bucle de ejecución que la aplicación. Esto lo hace mucho más rápido.
- Cypress incluye todas las herramientas necesarias para escribir pruebas de automatización de la interfaz de usuario de extremo a extremo, por lo que no necesitas configurar herramientas adicionales como TestNG, Cucumber, etc. Incorpora herramientas existentes y probadas para realizar sus respectivas

tareas. Por ejemplo, Cypress utiliza por defecto Mocha como marco de pruebas y Chai para las aserciones.

- Como Cypress está integrado en la aplicación, permite crear casos de prueba variados, como funciones de aplicación stubbing, simular escenarios de caída del servidor alterando las peticiones, establecer estados predefinidos de la aplicación, etc.
- Cypress aborda los fallos en las pruebas debidos a la adopción incorrecta de estrategias de espera, esperando automáticamente a que la página se cargue y los elementos sean visibles o se pueda hacer clic en ellos.
- Cypress hace que depurar los fallos de las pruebas sea mucho más sencillo, ya que proporciona capturas de pantalla, registros y vídeos de cada comando que se ha ejecutado en la prueba. También te permite inspeccionar los errores en la página de la aplicación en su estado predefinido como parte del flujo de la prueba utilizando Chrome DevTools.

Cypress cuenta con un buen apoyo de la comunidad de código abierto, y con frecuencia se añaden nuevos plug-ins para satisfacer requisitos avanzados. Así pues, veamos cómo configurar un marco de automatización de la interfaz de usuario utilizando Cypress y el Modelo de Objetos de Página.

NOTA

La comunidad Cypress aboga por utilizar el Modelo de Acciones de Aplicación en lugar del Modelo de Objetos de Página. Si quieres explorar esto, consulta la [entrada del blog](#) de Gleb Bahmutov.

Requisitos previos

Las siguientes herramientas son necesarias para configurar un marco de automatización en JavaScript:

- Node.js 12 o superior
- El IDE que elijas: VisualStudio Code es popular para proyectos de JavaScript
- Un navegador: Cypress funciona con Chrome, Chromium, Edge, Electron y Firefox

Ciprés

Una vez que tengas instalados los requisitos previos , sigue estos cinco pasos para familiarizarte rápidamente con el funcionamiento de Cypress:

1. Crea un directorio de proyecto. Instala Cypress ejecutando el siguiente comando desde la carpeta del proyecto en tu terminal:

```
$ npm install cypress --save-dev
```

2. Crea un archivo *package.json* en esta carpeta con el contenido que se muestra en el **Ejemplo 3-12.**

Ejemplo 3-12. El archivo package.json

```
{
  "name": "functional-tests",
  "version": "1.0.0",
  "description": "UI Driven End-to-End Tests",
  "main": "index.js",
  "devDependencies": {
    "cypress": "^9.2.0"
  },
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

3. Ejecuta el siguiente comando, que abrirá la aplicación Cypress como se ve en la **Figura 3-5**. También configurará una

estructura de marco de automatización de arranque con pruebas Cypress de ejemplo para una **aplicación web Todo** de muestra.

```
$ node_modules/.bin/cypress open
```

[Tests](#)[Runs](#)[Settings](#)

Chrome 91 ▾

Welcome to Cypress!



We've created some sample test files that demonstrate key Cypress concepts to help you get started.

[How to write your first test ↗](#) | No thanks, delete example files

Q Press Cmd + F to search...

+ New Spec File

▼ INTEGRATION TESTS [COLLAPSE ALL](#) | [EXPAND ALL](#)

▶ Run 20 integration specs

▼ □ 1-getting-started

[todo.spec.js](#)

▼ □ 2-advanced-examples

[actions.spec.js](#)

[aliasing.spec.js](#)

[assertions.spec.js](#)

[connectors.spec.js](#)

[cookies.spec.js](#)

Figura 3-5. Aplicación Cypress con archivos de prueba

4. Una vez realizada la configuración, puedes probar a ejecutar las pruebas existentes para hacerte una idea de lo fácil que es trabajar con Cypress antes de configurar el marco de objetos de página específico de tu aplicación. Selecciona tu navegador preferido en el desplegable de la esquina superior derecha de la aplicación Cypress, y haz clic en cualquier archivo de pruebas (*.spec.js*). Cypress abrirá el navegador, ejecutará las pruebas dentro del archivo y te mostrará un informe como el de la **Figura 3-6.**

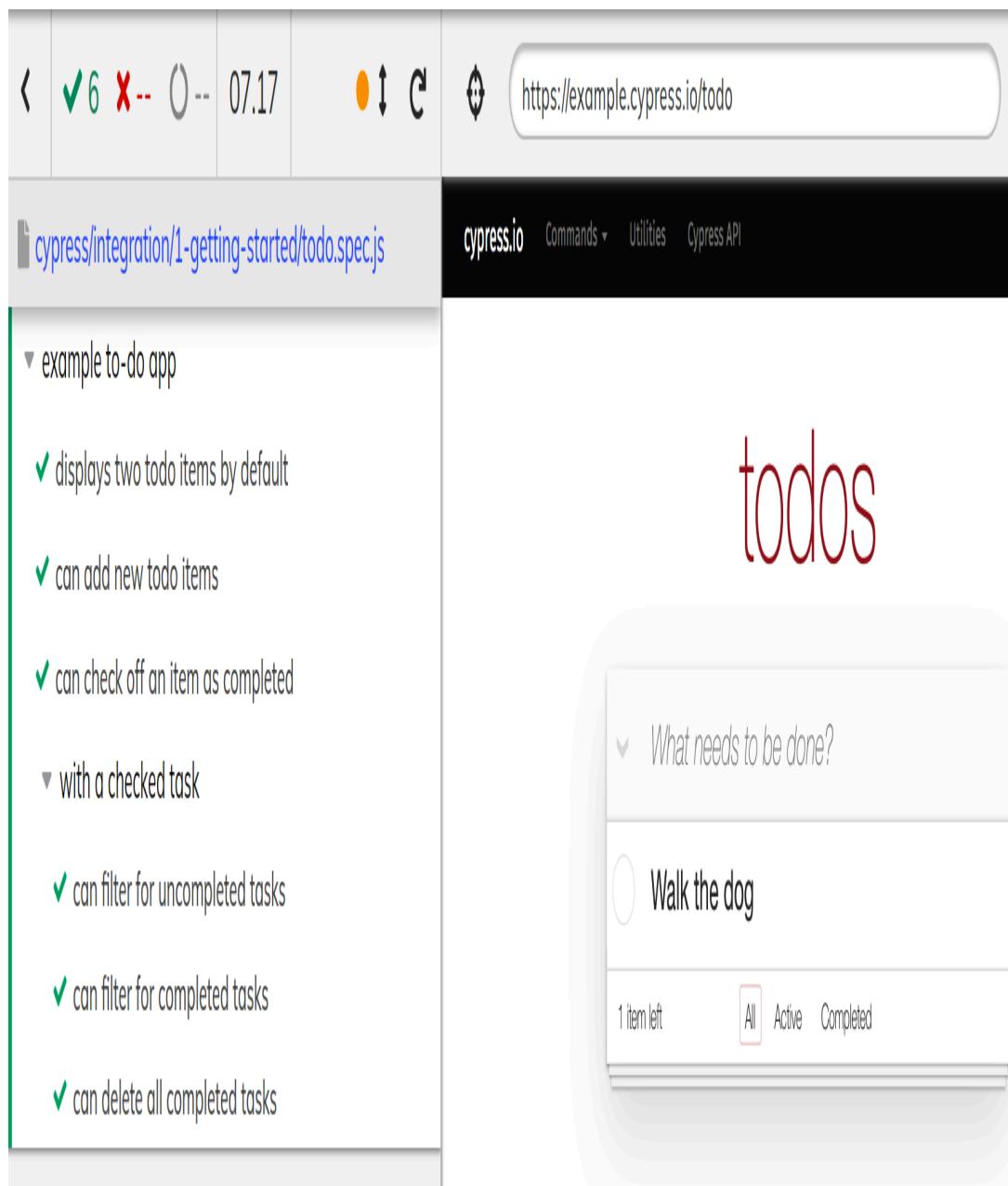


Figura 3-6. Informe de ejecución de la prueba Cypress

Para seguir explorando, haz clic en una de las pruebas. Verás una lista de los comandos ejecutados en esa prueba, y cuando pases el ratón por encima de cada comando se mostrará a la derecha el estado de la aplicación tal y como estaba cuando ejecutó el comando, como se ve en la [Figura 3-7](#). ¿Qué más ayuda podemos pedir para la depuración?

✓ 6 ✗ - 0 08.52 ⚡ C

cypress/integration/1-getting-started/todo.spec.js

▼ example to-do app

✓ displays two todo items by default

▼ BEFORE EACH

```
1 visit https://example.cypress.i...
```

▼ TEST BODY

```

1 get .todo-list li
2 -assert expected [ <li>, 1 more...
] to have a length of 2
3 get .todo-list li
  -first
5 -assert expected <li> to have text
  Pay electric bill
6 get .todo-list li
  -last
8 -assert expected <li> to have text
  Walk the dog

```

https://example.cypress.io/todo

cypress.io Commands Utilities Cypress API

What needs to be done?

- Pay electric bill
- Walk the dog

2 items left

All Active Completed

Double-click to edit a todo

Forked from TodoMVC

Figura 3-7. Depuración con Cypress

5. Para ejecutar pruebas desde la línea de comandos, añade el siguiente código a tu archivo *package.json* e inicia la ejecución de la prueba utilizando el comando `npm test`. Observarás que las pruebas se ejecutan en modo headless y que se crea una carpeta de *vídeos* en el directorio de tu proyecto con grabaciones de las ejecuciones de tus pruebas:

```
"scripts": {  
  "test": "cypress run"  
}
```

Ahora que ya has tenido una introducción al funcionamiento de Cypress, veremos brevemente los métodos que proporciona para interactuar con tu aplicación web y navegar por ella. Abre cualquier archivo de prueba, y observarás algunos métodos de uso común como los siguientes:

- `get(element_locator)` obtiene el elemento web del DOM tras esperar automáticamente a que esté disponible. La aplicación Cypress tiene una herramienta para inspeccionar el elemento y obtener el localizador, como se ve en la [Figura 3-8](#). Puedes utilizarla cuando crees tus módulos de página.

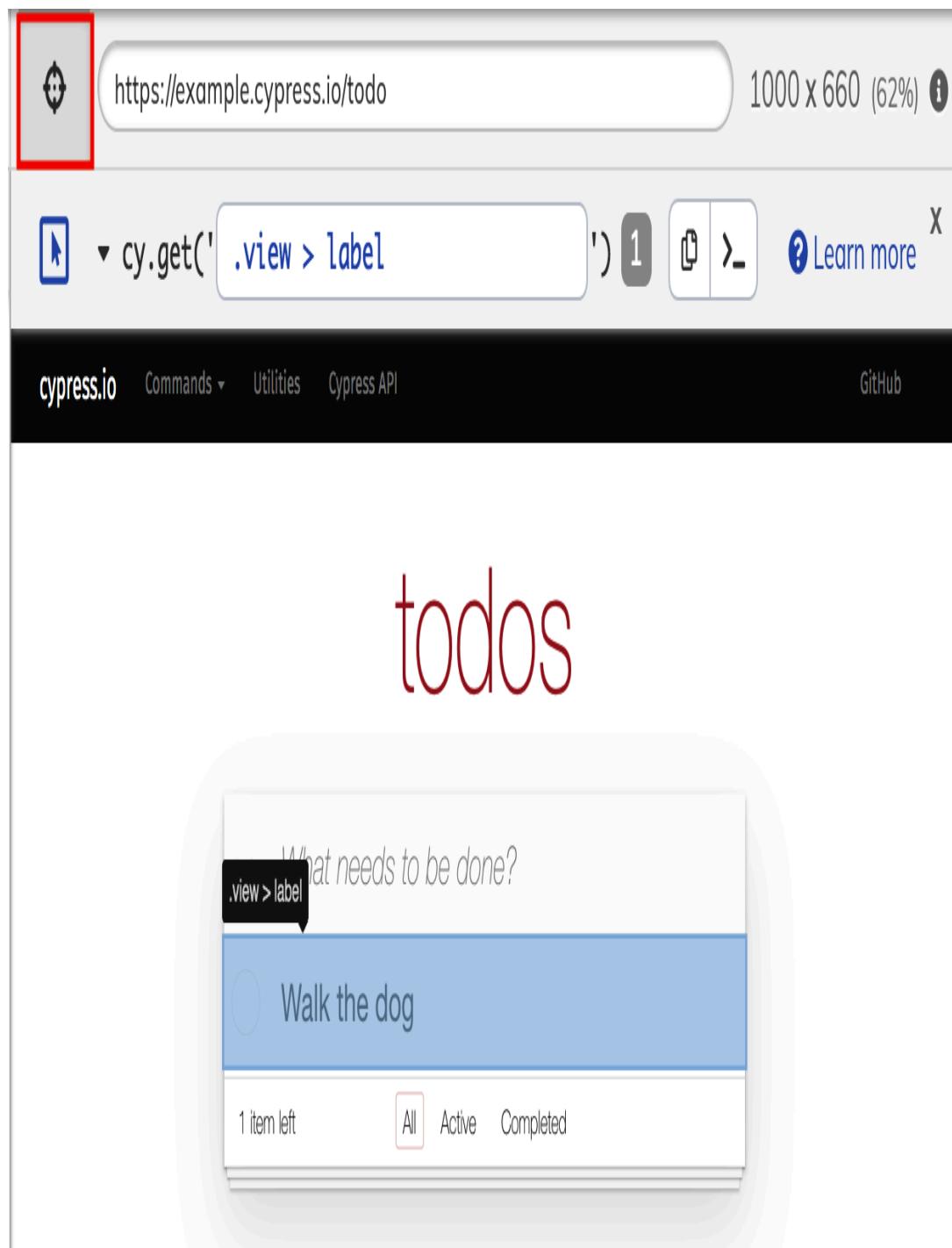


Figura 3-8. Encontrar localizadores de elementos utilizando Cypress

- `get(element_locator).click()` hace clic en el elemento elegido.
- `title()` devuelve el título de la página.

- `get(select_locator).select(option)` selecciona opciones de un desplegable.
- `get(element_locator).rightclick()` realiza un clic con el botón derecho del ratón sobre el elemento elegido .

Puedes encontrar otros métodos avanzados en la [documentación](#) detallada de la herramienta .

Configuración y flujo de trabajo

Sólo se necesitan unos pocos pasos para crear un marco de automatización utilizando Cypress y el Modelo de Objetos de Página. Puedes crear la misma prueba que en la sección Selenium para abrir una aplicación de comercio electrónico, iniciar sesión y hacer una afirmación en el título de la página de inicio siguiendo estos pasos:

1. Crea una nueva carpeta de pruebas en *cypress/integration*, digamos *ecommerce-e2e-tests*, y crea un archivo de pruebas, digamos *login_tests.spec.js*, en ella.
2. Ahora crea una nueva carpeta */page-objects* fuera de la carpeta */integration* y crea dentro de ella tus módulos de página: *login-page.js* y *home-page.js*, como se ve en el [Ejemplo 3-13](#).
3. Puedes ejecutar la prueba directamente desde la aplicación Cypress o utilizando el comando `npm test`.

Ejemplo 3-13. Estructura de objetos de página Cypress

```
// page-objects/login-page.js

/// <reference types="cypress" />

export class LoginPage {

  login(email, password){
    cy.get('[id=user_email]').type(email)
    cy.get('[id=user_password]').type(password)
    cy.get('.submitPara > .gr-button').click()
  }
}
```

```

}

// page-objects/home-page.js

/// <reference types="cypress" />

export class HomePage {

    getTitle(){
        return cy.title()
    }
}

// integration/eCommerce-e2e-tests/login_tests.spec.js

/// <reference types="cypress" />

import {LoginPage} from '../../page-objects/login-page'
import {HomePage} from '../../page-objects/home-page'

describe('example to-do app', () => {
    const loginPage = new LoginPage()
    const HomePage = new HomePage()

    beforeEach(() => {
        cy.visit('https://example.com')
    })

    it('should log in and land on home page', () => {
        loginPage.login('example@gmail.com', 'Admin123')
        HomePage.getTitle().should('have.string', 'Home Page')
    })
})

```

Fíjate en el uso del método `beforeEach()` proporcionado por el marco de pruebas Mocha (similar a `@beforeMethod` en TestNG) para abrir la URL de la aplicación antes de cada ejecución de la prueba y el método de aserción `should('have.string', string)` del marco Chai, que vienen incluidos con Cypress por defecto.

Cypress ejecuta tus pruebas automáticamente cada vez que guardas nuevos cambios en ellas. Esto facilita la creación de pruebas, ya que puedes verificar rápidamente que tu nuevo código funciona como esperabas. También verás cómo hacer pruebas visuales con Cypress en [el Capítulo 7](#). En resumen, si puedes superar el obstáculo de aprender JavaScript (que no es tan difícil), te beneficiarás enormemente de Cypress .

Pruebas de servicio

Pasemos ahora a las pruebas del servicio . En esta sección configuraremos un marco de automatización de pruebas utilizando la biblioteca REST Assured Java para validar una API REST de muestra. Si no conoces las API, consulta "[Pruebas de API](#)" para obtener una introducción.

Requisitos previos

En primer lugar, asegúrate de que tienes instalados los siguientes requisitos previos:

- La última versión de [Java](#).
- El IDE de tu [elección-IntelliJ](#) es una opción común para Java
- [Maven](#)

Marco Asegurado Java-REST

[REST Assured](#) es la biblioteca Java de referencia para realizar pruebas automatizadas de API REST. Ofrece a un lenguaje específico del dominio (DSL) con sintaxis Gherkin (Given, When, Then) para crear pruebas de API legibles y mantenibles, y utiliza matchers hamcrest para las aserciones. REST Assured puede funcionar con cualquier marco de pruebas, como JUnit o TestNG.

Supongamos que tenemos la siguiente API GET /items en nuestro hipotético servicio de pedidos, que devuelve una lista de artículos y

sus detalles:

GET: <https://eCommerce.com/items>

Response:

```
Status Code: 200
[
  {
    "SKU": "984058981",
    "Color": "Green",
    "Size": "M"
  }
]
```

Entonces el DSL REST Asegurado para llamar a la API GET y afirmar el código de estado tendrá el siguiente aspecto:

```
given().
when().
get("https://eCommerce.com/items").
then().
assertThat().statusCode(200);
```

¿No es sencillo? Del mismo modo, tienes DSL para POST, PUT y todos los demás métodos relacionados con las pruebas de la API.

Configuremos ahora un marco de pruebas de automatización de la API y escribamos una prueba para validar el mismo punto final GET /items. Puedes crear ese punto final como un stub en tu máquina siguiendo los pasos del [Capítulo 2](#).

CONSEJO

Si necesitas API de muestra con las que practicar, el sitio [Any API](#) tiene una lista consolidada de 1.400 API REST alojadas públicamente entre las que elegir.

Configuración y flujo de trabajo

Como vimos en la configuración del marco de automatización de la interfaz de usuario, los tres componentes básicos de un marco de pruebas automatizadas son el gestor de dependencias (Maven, en nuestro caso), una biblioteca para realizar el tipo de pruebas requerido (REST Assured para APIs), y un marco de pruebas para crear y ejecutar las pruebas (utilizaremos TestNG). Crea tu framework reuniendo esos tres componentes de la siguiente manera:

1. Crea un nuevo proyecto Maven con IntelliJ (o el IDE que prefieras). Consulta "**Marco Java-Selenium WebDriver**" para más detalles.
2. Añade las dependencias TestNG y REST Assured en tu archivo *pom.xml*. Puedes encontrar los parámetros de dependencia necesarios en el repositorio de Maven Central, como ya hemos comentado.
3. Crea un nuevo paquete llamado *tests* en la carpeta */src/test/java* y una nueva clase de prueba llamada *ItemsTest*.
4. **El Ejemplo 3-14** muestra una prueba de ejemplo para verificar el punto final GET */items*.

Ejemplo 3-14. La clase ItemsTest con una prueba API para el punto final GET /items

```
// ItemsTest.java

package apitests;

import org.testng.annotations.Test;

import static io.restassured.RestAssured.given;

public class ItemsTest {

    @Test
```

```

public void verifyGetItemsEndpointReturnsSuccessStatusCode(){
    given().
        when().
        get("http://localhost:1000/items").
        then().
        assertThat().statusCode(200);
}

```

Puedes ejecutar la prueba desde el IDE o ejecutando el comando `mvn clean test` desde tu terminal.

Una vez que tengas la configuración básica funcionando, puedes añadir una prueba para verificar un punto final POST /items. Digamos que el punto final POST toma los mismos detalles del artículo como JSON en el cuerpo de la solicitud y devuelve una respuesta HTTP 201 al añadir con éxito el artículo a un nuevo pedido. Crea un stub en tu máquina, siguiendo los pasos del [Capítulo 2](#).

Para pasar un cuerpo JSON a las peticiones POST, un método más limpio es crear una clase `dataObject` y serializarla utilizando una biblioteca de serialización JSON; por ejemplo, la biblioteca `jackson-databind`. Añadámoslo a nuestro framework:

1. Añade la biblioteca `jackson-databind` en tu archivo `pom.xml`.
2. Ahora crea un nuevo paquete `dataObjects` en `/src/main/java` y añade una nueva clase `dataObject`, digamos `ItemDetails.java`. **El ejemplo 3-15** muestra la clase `ItemDetails` que representa el cuerpo JSON de la solicitud POST.

Ejemplo 3-15. La clase `ItemDetails` como `dataObject`

```

// ItemDetails.java

package dataobjects;

import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.annotation.JsonPropertyOrder;

```

```

@JsonPropertyOrder({"sku", "color", "size"})
public class ItemDetails {

    private String sku;
    private String color;
    private String size;

    public ItemDetails(String sku, String color, String size){
        this.sku = sku;
        this.color = color;
        this.size = size;
    }

    @JsonProperty("sku")
    public String getSKU(){
        return sku;
    }

    @JsonProperty("color")
    public String getColor(){
        return color;
    }

    @JsonProperty("size")
    public String getSize(){
        return size;
    }
}

```

Observa cómo la biblioteca jackson-databind permite definir la estructura JSON esperada al principio con la anotación `@JsonPropertyOrder`.

3. En tu clase de prueba, puedes utilizar el objeto `ItemDetails` como cuerpo de la petición POST. [El Ejemplo 3-16](#) muestra la prueba del punto final POST `/items`.

Ejemplo 3-16. Prueba de la API para el punto final POST /items

```

@Test
public void verifyPostItemsEndpointReturnsSuccessStatusCode(){

```

```
ItemDetails greenShirt = new ItemDetails("98765490", "Green",
"M");

given().
    contentType(MediaType.JSON).
    body(greenShirt).
    log().body().
when().
    post("http://localhost:1000/items").
then().
    assertThat().
    statusCode(200);
}
```

Cuando ejecutes la prueba, el método `log().body()` registrará el cuerpo de la solicitud para que compruebes la serialización. Aquí sólo hemos afirmado sobre la `statusCode` de la respuesta. REST Assured ofrece más flexibilidad para encontrar los campos requeridos en el cuerpo de la respuesta, como se detalla en la [documentación oficial](#), y aseverarlos adecuadamente.

Pruebas unitarias

Dado que las pruebas unitarias están estrechamente integradas con el código de la aplicación, el marco de pruebas que utilices debe ser compatible con el lenguaje de programación de la aplicación, como JUnit o TestNG para Java, NUnit para .NET, Jest o Mocha para JavaScript, RSpec para Ruby, etc. Aquí veremos la configuración de JUnit. Los requisitos previos para JUnit son los mismos que los mencionados para las pruebas de API.

NOTA

Aunque las pruebas unitarias sólo las escriben los desarrolladores, es importante que los probadores comprendan su estructura básica para que puedan planificar sabiamente la estrategia de pruebas de la aplicación. La intención de incluir esto como ejercicio es dar esa experiencia a los probadores, y de ahí que esta sección se mantenga sencilla.

JUnit

JUnit es un marco de pruebas unitarias muy popular creado por Kent Beck y Erich Gamma en 1997. Desde entonces ha atendido todo el espectro de necesidades de pruebas unitarias y sigue siendo el marco de pruebas unitarias de facto para Java en la actualidad. JUnit ofrece funciones de creación de pruebas, aserción, organización, ejecución e informes. TestNG, otro marco popular, se creó para subsanar algunas de las carencias de JUnit, pero JUnit ha mejorado sus características en sus últimas ediciones para colmar esas lagunas.

Algunas de las características básicas de JUnit son las siguientes:

- Anotaciones de prueba y ciclo de vida como `@Test` para marcar los métodos de prueba, y `@BeforeEach`, `@BeforeAll`, `@AfterEach`, y `@AfterAll` para las actividades de preparación y desmontaje
- La anotación `@DisplayName` para mostrar un nombre legible para cada prueba
- Anotaciones de etiquetado personalizadas, como `@Tag("smoke")`, que pueden utilizarse para ejecutar sólo un subconjunto de pruebas cuando sea necesario
- APIs de aserción como `assertTrue()`, `assertEquals()`, `assertAll()`, etc.

Configuración y flujo de trabajo

Vamos a escribir un par de pruebas unitarias sencillas para el servicio de atención al cliente de nuestra aplicación de comercio electrónico. Crea un nuevo proyecto Java y añade una clase `CustomerManagement`, como se ve en el [Ejemplo 3-17](#). Esta clase de ejemplo tiene dos métodos que añaden y devuelven detalles del cliente. A continuación añadiremos pruebas unitarias para estos dos métodos.

Ejemplo 3-17. La clase CustomerManagement

```
// CustomerManagement.java

package Customers;

import java.util.ArrayList;
import java.util.List;

public class CustomerManagement {

    private String firstName;
    private String lastName;
    private String age;

    private List<List<String>> customers = new ArrayList<List<String>>()
();

    public List<List<String>> getCustomers(){
        return customers;
    }

    // if the customer name is empty, throw an exception; else add the
    // customer
    public void addCustomer(List<String> customerDetails){
        if (customerDetails.get(0).isEmpty())
            throw new IllegalArgumentException();
        customers.add(customerDetails);
    }
}
```

Para añadir las pruebas unitarias:

1. Añade las siguientes dependencias en tu archivo *pom.xml*:

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.7.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.7.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

2. Crea una nueva clase de prueba en el archivo *CustomerManagementTests.java*, dentro de la carpeta */src/main/test*.

3. Utiliza las anotaciones y aserciones de JUnit para crear tus pruebas, como se ve en el [Ejemplo 3-18](#).

Ejemplo 3-18. CustomerManagementTests.java con pruebas JUnit

```
package customersUnitTests;

import Customers.CustomerManagement;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

import java.util.ArrayList;
import java.util.List;

@DisplayName("When managing new customers")
```

```

public class CustomerManagementTests {

    @Test
    @DisplayName("should return empty when there are no customers")
    public void shouldReturnEmptyWhenThereAreNoCustomers(){
        CustomerManagement customer = new CustomerManagement();
        List<List<String>> customers = customer.getCustomers();

        assertTrue(customers.isEmpty(), "Error: Customers exists");
    }

    @Test
    @DisplayName("should throw exception when customer name is
invalid")
    public void shouldThrowExceptionForInvalidInput(){
        List<String> newCustomer = new ArrayList<>();
        newCustomer.add("");
        newCustomer.add("Jackson");
        newCustomer.add("20");

        CustomerManagement customer = new CustomerManagement();
        IllegalArgumentException err =
            assertThrows(IllegalArgumentException.class, () ->
customer.addCustomers(newCustomer));

    }
}

```

Puedes ver que la etiqueta `@DisplayName` incluye descripciones de pruebas legibles. Como éstas explican, la primera prueba comprueba si el método `getCustomers()` devuelve un valor vacío cuando no hay clientes existentes, y la segunda prueba afirma que el método `addCustomers()` devuelve un `IllegalArgumentException` cuando se añade un cliente con un nombre de pila vacío. Observa también los distintos métodos de aseveración para aseverar excepciones y valores de retorno.

Puedes ejecutar estas pruebas desde el IDE o desde la línea de comandos, utilizando el comando `mvn clean test`. Verás los

resultados de la ejecución de las pruebas con sus nombres en pantalla, como se muestra en la [Figura 3-9](#).

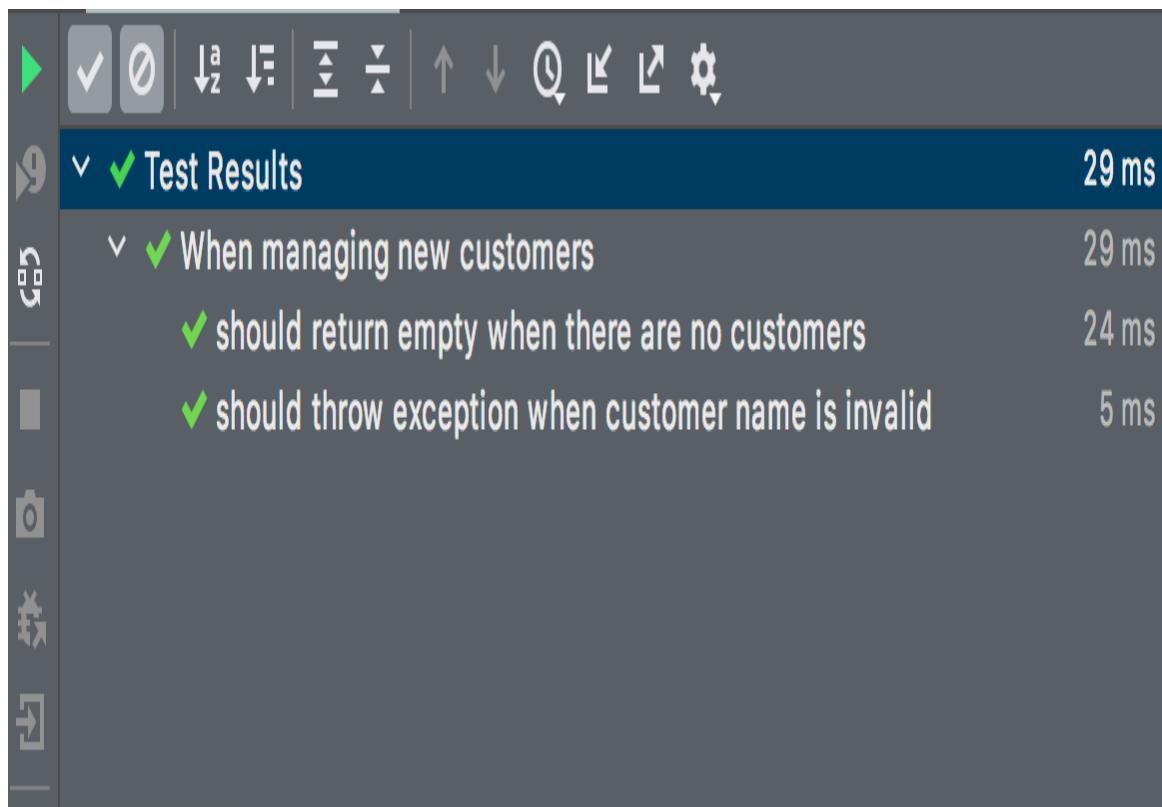


Figura 3-9. Resultados de la ejecución de pruebas JUnit en IntelliJ, que muestra sus nombres de visualización legibles

Aparte de JUnit, dependiendo del caso de prueba unitario que estés probando, es posible que necesites capacidades adicionales del marco de desarrollo de la aplicación (por ejemplo, Spring Boot) y bibliotecas externas como Mockito para simular la llamada al servicio, jackson-databinder para la vinculación de datos, etc. Cuando se utilizan estas capacidades adicionales para acceder a un sistema externo, como una base de datos, la prueba unitaria se convierte en una prueba de integración.

CARACTERÍSTICAS DE LOS BUENOS TESTS

Las características enumeradas aquí se aplican a todos los tipos de pruebas que hemos discutido hasta ahora. Las pruebas sin estas características pueden convertirse fácilmente en un peligro para el mantenimiento:

- Las pruebas deben ser legibles, con nombres apropiados de métodos y variables que expresen adecuadamente la intención. Sigue el patrón Organizar, Actuar y Afirmar (AAA), que sugiere que primero organices los requisitos previos del caso de prueba, luego realices las acciones necesarias para el caso de prueba y, por último, afirmes el comportamiento esperado.
- Cada prueba debe verificar un solo comportamiento para que sea rápida y exprese las intenciones correctas cuando falle.
- Las pruebas deben ser independientes entre sí. Recuerda que encadenar pruebas también provocará errores encadenados. Tener una configuración y desmontaje adecuados para cada prueba ayudará a mantener tus pruebas independientes y facilitará la ejecución en paralelo.
- Las pruebas deben ser agnósticas al entorno. Por ejemplo, las pruebas no deben depender de datos estáticos en un entorno concreto.
- Automatiza los procesos de creación y ejecución de pruebas para que cualquier miembro del equipo pueda consultar el código y ejecutar un único comando para activar las pruebas sin tener que gestionar manualmente las dependencias .

Herramientas de comprobación adicionales

En esta sección exploraremos algunas herramientas más de automatización de pruebas: Pact, una herramienta de pruebas de contratos; Karate, una herramienta BDD para crear pruebas de servicios; y algunas de las herramientas de automatización de pruebas AI/ML que actualmente están en auge en el mercado. Esto te proporcionará una comprensión más amplia de las herramientas en el espacio de la automatización de pruebas funcionales y, por tanto, te ayudará a hacer elecciones acertadas cuando sea necesario.

Pacto

Pact es una popular herramienta para crear pruebas de contratos en Java. Las pruebas también se pueden escribir en Python, JavaScript, Go, Scala y otros lenguajes. Pact se utiliza específicamente para *pruebas de contratos orientadas al consumidor*.

Un *consumidor* es una aplicación (por ejemplo, un servicio o una interfaz web) que recibe información de otra aplicación (por ejemplo, un servicio o una cola de mensajes). Obviamente, la aplicación que proporciona la información requerida es el *proveedor*. Por poner un ejemplo, el servicio de pedidos de la aplicación de comercio electrónico recibe los detalles del artículo del vendedor del servicio PIM, por lo que el servicio de pedidos es un consumidor y el servicio PIM es el proveedor. Ten en cuenta que muchos otros consumidores podrían consumir el servicio PIM aparte del servicio de pedido. Además, cada consumidor puede necesitar información distinta del servicio PIM. Por ejemplo, el servicio de pedidos puede requerir la SKU de cada artículo como parte de los detalles del mismo, pero puede no utilizar la dirección de su fabricante, que puede ser un requisito para otro consumidor.

Dado que los requisitos están impulsados por el consumidor, el servicio PIM puede verse empujado a una situación en la que tenga

que cambiar sus contratos para atender a un nuevo consumidor o a un nuevo requisito, lo que supondrá un riesgo para el servicio de pedidos y otros equipos de consumidores. Necesitan un mecanismo para verificar continuamente que los contratos del servicio PIM - especialmente los atributos relevantes para ellos- están intactos para evitar problemas de integración más adelante. Los probadores o desarrolladores pueden escribir pruebas de servicio o de integración para mitigar el riesgo, pero estas pruebas pueden ser frágiles y lentas debido a las dependencias de ambas aplicaciones, además de costosas de configurar y mantener. Tangencialmente, a veces el proveedor y el consumidor pueden estar experimentando un desarrollo paralelo, lo que significa que ni siquiera puedes escribir pruebas de integración o de servicio de extremo a extremo. Las pruebas de contrato orientadas al consumidor se convierten entonces en la clave para resolver estos enredos.

Como se ve en [la Figura 3-10](#), con las pruebas de contratos dirigidas por el consumidor, cada equipo de consumidores escribe pruebas contra la versión stubbed de los contratos acordados con el proveedor. Las pruebas afirman específicamente los atributos esperados por ese consumidor y no todo el contrato. A continuación, estas pruebas se transmiten al equipo del proveedor, que las ejecuta todas contra las API reales del proveedor, asegurándose de que satisface las necesidades de sus consumidores según lo esperado. Cuando se detectan desviaciones, el equipo del proveedor puede, al menos, advertir al consumidor respectivo que espere el cambio.

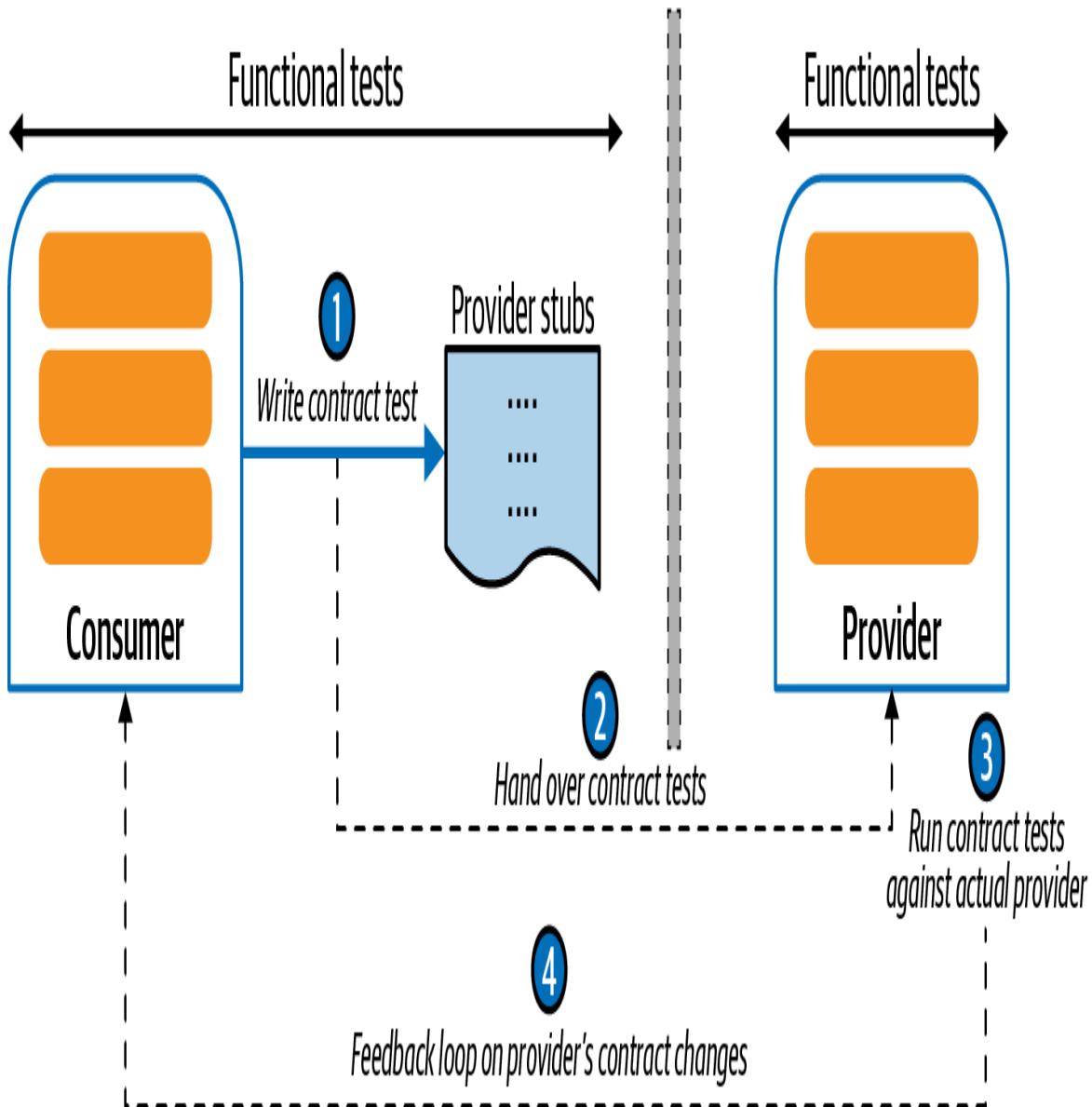


Figura 3-10. Flujo de pruebas de contratos impulsados por el consumidor

Básicamente, este tipo de pruebas por contrato divide las pruebas de integración de extremo a extremo en partes, como se indica a continuación:

- Cada consumidor escribe pruebas a nivel micro y macro para validar su comportamiento funcional haciendo un stubbing del proveedor, como se ve en la Figura 3-10.
- Cada consumidor también escribe pruebas de contrato contra el stub del proveedor, y éste las ejecuta continuamente.

- El proveedor escribe pruebas a nivel micro y macro para validar su comportamiento funcional .

Esto alivia algunos de los puntos dolorosos de escribir pruebas de integración/servicio de extremo a extremo, ya que el alcance de las pruebas de contrato es menor, y elimina las dependencias.

Pact permite automatizar totalmente el proceso de comprobación de contratos. Para hacernos una idea de su flujo de trabajo, utilizaremos el mismo ejemplo de pedido y servicio PIM. Digamos que el servicio de pedidos se está integrando con el punto final GET /items del servicio PIM externo para recuperar los detalles del artículo, concretamente las tallas, SKU y colores disponibles. El flujo de trabajo Pact de los dos equipos será el siguiente:

1. Como primer paso, el equipo del servicio de pedidos recopila todos los casos de prueba de integración. Por ejemplo, algunos de los casos de prueba de integración serán que el punto final /items devuelva los detalles del artículo como se espera cuando el artículo existe, devuelva una matriz vacía cuando el artículo no existe y devuelva los códigos de error apropiados (404, 500, etc.) en las solicitudes no válidas.
2. El equipo del servicio de pedidos crea stubs para estos casos de prueba utilizando Pact.
3. El equipo de servicio de pedidos escribe pruebas de contrato de consumidor utilizando Pact contra estos stubs, afirmando sobre los atributos específicos: códigos de estado, SKU, tallas disponibles y colores. Estas pruebas, cuando se ejecutan, producen automáticamente un *archivo pact*. Este archivo captura las diferentes solicitudes al punto final /items y las afirmaciones sobre los atributos esperados en las respuestas.
4. El archivo del pacto se transmite automáticamente al equipo PIM a través de una disposición de código abierto llamada Pact Broker, que debe ser configurada y mantenida tanto por el

equipo de consumidores como por el de proveedores. El equipo Pact también ofrece un servicio de pago llamado Pactflow, que elimina la necesidad de configurar y mantener el Pact Broker. Para hacerlo más sencillo, los archivos también pueden compartirse mediante carpetas.

5. En el lado del servicio PIM, el equipo escribe una prueba de contrato de proveedor para recibir el archivo pact del agente Pact y configurar los datos de prueba en diferentes estados según los requisitos de las pruebas de consumidor. Cuando se ejecute la prueba del proveedor, Pact realizará las solicitudes adecuadas descritas en el archivo pact contra el servicio PIM real y verificará las respuestas reales.
6. Los resultados de la prueba del proveedor están a disposición del consumidor a través del Agente del Pacto, con lo que se completa todo el circuito de información sin intervención.
7. Tanto las pruebas Pact del consumidor como las del proveedor están integradas en la canalización CI para que los equipos puedan recibir información continuamente.

El ejemplo 3-19 muestra un ejemplo de prueba de consumidor Pact con un `pactMethod`. En primer lugar, el `pactMethod` establece el estado del punto final `/items` tal y como espera la prueba de consumidor Pact. Como puedes ver, el método `given()` describe este estado y será consultado por la prueba de proveedor para iniciar la configuración de los datos de prueba adecuados. A continuación, la prueba de consumidor Pact abre el stub `/items` tal y como se describe en `pactMethod` y se afirma en la respuesta de detalles del elemento.

Ejemplo 3-19. Ejemplo de prueba de consumo con Pact

```
@ExtendWith(PactConsumerTestExt.class)
public class ItemsPactConsumerTest {

    @Pact(consumer = "Order service", provider = "PIMService")
```

```

    RequestResponsePact getAvailableItemDetails(PactDslWithProvider
builder) {
    return builder.given("items are available")
        .uponReceiving("get item details")
        .method("GET")
        .path("/items")
        .willRespondWith()
        .status(200)
        .headers(Map.of("Content-Type", "application/json;
charset=utf-8"))
        .body(newJsonArrayMinLike(2, array ->
            array.object(object -> {
                object.stringType("SKU", "A091897654");
                object.stringType("Color", "Green");
                object.stringType("Size", "S");
            })
        ).build())
        .toPact();
}

@Test
@PactTestFor(pactMethod = "getAvailableItemDetails")
void getItemDetailsWhenItemsAreAvailable(MockServer mockServer) {

    // brings up the PIM /items endpoint stub as described by
    // the pact method above
    RestTemplate restTemplate = new RestTemplateBuilder()
        .rootUri(mockServer.getUrl())
        .build();

    List<Item> items = new
    PIMService(restTemplate).getAvailableItemDetails();

    Item item1 = new Item("A091897654", "Green", "S");
    Item item2 = new Item("A091897654", "Green", "S");
    List<Item> expectedItems = List.of(item1, item2);
    assertEquals(expectedItems, items);
}

```

Esta prueba generará un archivo pact y se comparte con el proveedor a través de una carpeta. La prueba del proveedor pact del **Ejemplo 3-20** recibe el archivo pact, configura los datos de prueba

según el método anotado @State, accede al punto final real /items siguiendo las instrucciones del archivo pact, y comprueba que la respuesta real tiene el mismo formato de detalles del elemento que en el [Ejemplo 3-19](#).

Ejemplo 3-20. Ejemplo de prueba de proveedor con Pact

```
@Provider("PIMService")
@PactFolder("pacts")
@ExtendWith(SpringExtension.class)
@SpringBootTest(webEnvironment =
    SpringBootTest.WebEnvironment.RANDOM_PORT)

public class ItemsPactProviderTest {

    @LocalServerPort
    int port;

    @MockBean
    private ItemRepository itemRepository;

    @BeforeEach
    void setUp(PactVerificationContext context) {
        context.setTarget(new HttpTestTarget("localhost", port));
    }

    @TestTemplate
    @ExtendWith(PactVerificationInvocationContextProvider.class)
    void verifyPact(PactVerificationContext context, HttpRequest request)
    {
        context.verifyInteraction();
    }

    @State("items are available")
    void setItemsAvailableState() {
        when(itemRepository.getItems()).thenReturn(
            List.of(new Item("A091897654", "Green", "S"),
                   new Item("A091897654", "Green", "S")));
    }
}
```

Pact genera informes HTML que pueden integrarse con CI. Por lo general, las pruebas de Pact están estrechamente vinculadas al

código de la aplicación, y para crearlas y depurarlas puede ser necesario conocer los marcos de desarrollo de aplicaciones que se utilizan (por ejemplo, Spring Boot) .

Karate

Karate llama la atención principalmente por su forma única de ayudar a la creación de pruebas de servicio. Ofrece sentencias Gherkin predefinidas (similares a Cucumber) para escribir pruebas, eliminando la necesidad de codificar. La herramienta no se limita a las pruebas de la API -pretende dar soporte a pruebas de interfaz de usuario automatizadas de extremo a extremo, pruebas de contratos, configuraciones de servidores simulados, etc.-, pero hace que escribir pruebas de servicios sea mucho más sencillo de lo que puedas imaginar. **El Ejemplo 3-21** muestra la misma prueba que utilizamos para hacer una afirmación en el punto final GET /items con REST Assured escrito utilizando Karate.

Ejemplo 3-21. Una prueba para el punto final GET /items utilizando el DSL Karate

Feature: Order service should return item details

```
Scenario: verify GET items endpoint
  Given url 'http://localhost:1000/items'
  When method get
  Then status 200
```

Eso es todo: tres líneas de sentencias Gherkin predefinidas. Puedes encontrar una lista completa de estas sentencias en la [página GitHub](#) de Karate. Instalar la herramienta es tan sencillo como importar un arquetipo de Maven durante la creación del proyecto en IntelliJ.

Herramientas AI/ML en Pruebas Funcionales Automatizadas

Hemos hablado de bastantes herramientas en este capítulo, y son suficientes para tus necesidades de automatización de pruebas funcionales en todas las capas de la aplicación. Sin embargo, la inteligencia artificial y las tecnologías de aprendizaje automático han dado lugar a nuevas herramientas que proporcionan una mejor asistencia en algunas tareas cotidianas de automatización de pruebas, como la autoría de pruebas, el mantenimiento de pruebas, el análisis de informes de pruebas y el gobierno de pruebas. En esta sección, ofreceré una rápida visión general de las herramientas disponibles en la actualidad.

Creación de pruebas

El salto de la IA/ML para ayudar en la autoría de pruebas es un hito importante en el espacio de las pruebas, ya que permite a personas sin conocimientos de codificación autorizar fácilmente pruebas funcionales basadas en la interfaz de usuario. Test.ai, Functionize, Appvance, Testim y TestCraft son algunas de las herramientas de pago que ofrecen esta funcionalidad.

Para crear pruebas con estas herramientas, tendrás que navegar manualmente por el flujo del usuario en el sitio web, mientras que el grabador "respaldado por ML" de la herramienta identifica los elementos y las acciones realizadas en cada paso y crea las pruebas en segundo plano. Una ventaja del grabador respaldado por ML es que identifica los elementos no sólo por sus localizadores, sino también por sus aspectos estructurales y visuales. Algunas de estas herramientas también ayudan en el mantenimiento de las pruebas y el análisis de la causa raíz, lo que aligera significativamente la carga en el espacio de la automatización de pruebas. También pueden conectarse a CI, para obtener retroalimentación continua.

Mantenimiento de las pruebas

¿Alguna vez te has enfrentado a una situación en la que hubiera un gran número de fallos en las pruebas funcionales de la interfaz de usuario debido al cambio de ID de un solo elemento? La mayoría de las veces, sólo habrá cambiado el ID del elemento, y su funcionalidad, aspecto y sensación habrán permanecido inalterados. Aun así, las pruebas funcionales de la interfaz de usuario fallarán, ya que dependen principalmente de los valores del localizador del elemento. A mí me ha pasado, y me preguntaba si habría herramientas que pudieran autocorregir esos pequeños cambios y ahorrarme tiempo.

Esta funcionalidad de autocorrección, denominada *autocuración*, ya está disponible como parte de herramientas de automatización de pruebas impulsadas por IA/ML, como [test.ai](#) y [Functionize](#). Como ya se ha dicho, la grabadora respaldada por ML captura los aspectos estructurales y visuales de los elementos de la interfaz de usuario junto con sus localizadores; cuando cambia el valor del localizador de un elemento, éste sigue identificándose como el mismo, y las herramientas sólo necesitan nuestra aprobación para actualizar el valor del localizador en los guiones de prueba.

Análisis de informes de pruebas

Como he mencionado antes, he visto grandes proyectos empresariales que tenían cientos de pruebas automatizadas basadas en la interfaz de usuario que se ejecutaban toda la noche, con un equipo de automatización dedicado a analizar los resultados de las pruebas por la mañana. El equipo pasaba horas intentando averiguar las causas de los fallos de las pruebas. La mayoría de las veces, eran una de estas tres cosas: defectos, cambios en las nuevas funciones o problemas del entorno. Una vez encontradas las causas profundas, el equipo elaboraba informes de errores para los defectos, corregía los scripts de prueba para los cambios en las nuevas funciones y hacía un seguimiento con el equipo de infraestructura para resolver

los problemas del entorno. Todos los días estaban ocupados con estas tareas repetitivas. [ReportPortal](#), una herramienta de análisis de informes de pruebas de código abierto, podría haberles sido muy útil.

ReportPortal dispone de una función de autoanálisis basada en ML que lee los registros de fallos de pruebas y los clasifica en defectos, problemas del script de prueba y problemas del entorno. El algoritmo ML aprende de los datos de registro de los fallos de prueba analizados previamente. Esto requiere cierto esfuerzo manual preliminar para analizar los fallos de prueba anteriores y etiquetarlos adecuadamente. Una vez que hay suficientes datos de análisis de fallos de prueba, el autoanalizador aprende de ellos y empieza a identificar los fallos de prueba con precisión, ahorrando mucho tiempo al equipo.

Gobernanza de las pruebas

Una cobertura de pruebas adecuada en las capas apropiadas de la aplicación es una cuestión pertinente para todos los equipos. Un equipo puede alegrarse de un gran porcentaje de cobertura de pruebas funcionales o unitarias, pero ignorar que un módulo no tiene ninguna prueba. La gobernanza de las pruebas consiste en garantizar que las pruebas adecuadas estén en las capas adecuadas e injectar puertas de calidad en cada capa. Esto requiere datos de todas las capas, incluidas las funcionalidades que no se han probado. [SeaLights](#) es una herramienta de gobernanza de pruebas impulsada por IA/ML precisamente para este propósito: presenta métricas sobre la cobertura de las pruebas en todas las capas, identifica las áreas de código con una cobertura de pruebas deficiente, identifica los riesgos de calidad correlacionando los datos de ejecución de las pruebas y la cobertura de las pruebas, y proporciona muchas otras funciones relacionadas con la gobernanza de la calidad.

Tales son las mejoras que aportan las tecnologías IA/ML en el espacio de la automatización de pruebas. En resumen, su ayuda está empezando a ser significativa, y además siguen evolucionando. Siempre que sea posible, los equipos deben aprovechar estas herramientas para descargar las tareas repetitivas, de modo que puedan centrar su capacidad intelectual en tareas de orden superior, como la planificación, la innovación, la seguridad, el rendimiento, etc.

Perspectivas

Hemos profundizado y ampliado el espacio de la automatización de pruebas funcionales, pero antes de cerrar el capítulo, me gustaría llamar tu atención sobre algunos temas clave más: los antipatrones en las pruebas funcionales automatizadas, la cobertura de las pruebas de automatización y, concretamente, lo que significa tener una cobertura de automatización del 100%.

Antipatrones a superar

Incluso después de haber dedicado mucho tiempo y esfuerzo a elaborar la estrategia adecuada de pruebas funcionales automatizadas y a implantar los marcos de pruebas en las capas adecuadas, es esencial darse cuenta de que tus esfuerzos de pruebas funcionales automatizadas no han hecho más que empezar. A lo largo del plazo de entrega, debes seguir vigilando los antipatrones en las pruebas funcionales automatizadas a medida que el equipo avanza en el desarrollo de más y más pruebas. En mi opinión, es fácil caer presa de estos antipatrones con el ajetreo de la entrega, por lo que resulta crucial estar atento a los primeros síntomas. En esta sección hablaremos de algunos antipatrones comunes -el cono de helado y la magdalena, como se ve en [la Figura 3-11-](#), junto con sus síntomas y consejos para superarlos.

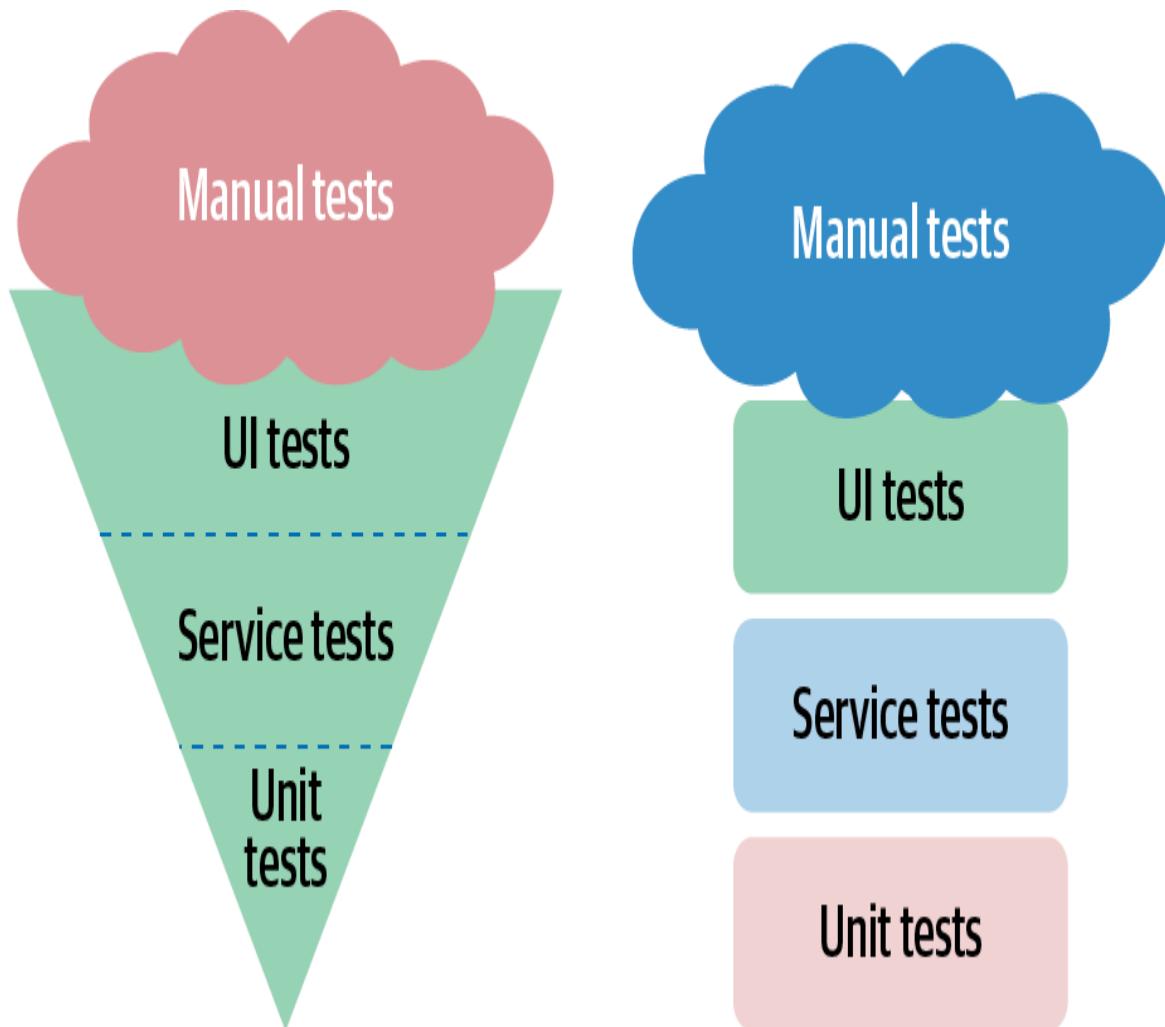


Figura 3-11. Antipatrones en las pruebas funcionales automatizadas

El cucuricho de helado

Cuando inviertes la pirámide de pruebas, parece como un cono. Es lo que se conoce como el **antipatrón del cono de helado**, en el que hay más pruebas de nivel macro orientadas a la interfaz de usuario y muy pocas pruebas de nivel micro. Puedes percibir el antipatrón del cono de helado cuando observes algunos de estos síntomas en el proyecto:

- Esperar un largo periodo para obtener información de las pruebas realizadas

- Detectar los defectos más tarde en el ciclo, a veces sólo durante la fase de pruebas de lanzamiento
- Se necesitan pruebas manuales elaboradas para dar retroalimentación a pesar de tener pruebas automatizadas
- Frustración en el equipo con las pruebas automatizadas, ya que los esfuerzos diligentes en la automatización de los flujos de la interfaz de usuario no han dado los resultados adecuados

CONSEJO

La señal más temprana en la que puedes evitar que tu equipo se desvíe bruscamente hacia este antipatrón es cuando encuentres defectos de regresión durante las pruebas manuales de historias. Haz inmediatamente un análisis de la causa raíz y corrige pronto las prácticas de tu equipo.

La magdalena

Cuando duplicas las pruebas en varias capas de , en lugar de una pirámide de pruebas acabas teniendo una capa inferior ancha, una intermedia ancha y una superior aún más ancha; en conjunto, parece **una magdalena**. Este tipo de desorganización suele producirse cuando tienes equipos de desarrolladores y probadores automatizados divididos en silos. Por ejemplo, los desarrolladores habrán añadido pruebas unitarias para verificar todas las entradas de inicio de sesión no válidas, y los probadores añadirán las mismas pruebas en la capa de la interfaz de usuario.

Puedes percibir este antipatrón cuando tu equipo tarda mucho tiempo en publicar incluso una característica minúscula. También puedes notar juegos de culpas, ya que un rol esperará que el otro haya añadido las pruebas apropiadas siempre que haya un fallo.

CONSEJO

Una forma sencilla de evitar este antipatrón es mantener un breve debate entre las funciones relevantes de un equipo para determinar qué pruebas se espera que se escriban en cada capa. La vía adecuada para tal discusión podría ser la reunión de inicio de la historia de usuario, a la que debería seguir la documentación de los resultados de la discusión en las tarjetas de historia de usuario.

100% de cobertura de automatización

Los equipos suelen hacer un seguimiento del porcentaje de cobertura de automatización como métrica, y un porcentaje alto suele considerarse una validación de sus buenas prácticas de desarrollo de software. El porcentaje de cobertura de automatización se calcula capturando todos los casos de prueba de la aplicación, marcándolos como automatizados o no, y utilizando matemáticas sencillas para obtener un porcentaje. Los equipos suelen fijarse el objetivo de alcanzar el 100% de cobertura de automatización, con buenas intenciones, pero al hacerlo, es importante tener en cuenta algunas indicaciones.

COBERTURA DEL CÓDIGO Y PRUEBAS DE MUTACIÓN

La métrica tradicional de cobertura del código es diferente de la métrica de cobertura de las pruebas de automatización. La cobertura de código indica si hay líneas de código que no serán ejecutadas por las pruebas unitarias existentes. En otras palabras, identifica las líneas de código no probadas. Las herramientas de cobertura de código, como JaCoCo y Cobertura, pueden integrarse en el proceso de compilación CI y hacer que la compilación falle cuando el porcentaje de cobertura de código esté por debajo de un determinado umbral, para evitar que el código no probado llegue más allá de la fase de compilación. Sin embargo, una cobertura de código alta no indica necesariamente que todos los casos de prueba estén automatizados.

Para encontrar los casos de prueba omitidos en las pruebas unitarias, se emplea una técnica llamada *prueba de mutación*. Las pruebas de mutación cambian el código de la aplicación y comprueban si fallan las pruebas. Por ejemplo, cuando hay llamadas a métodos nulos, elimina las llamadas en el código y vuelve a ejecutar las pruebas unitarias. Se dice que la mutación ha "muerto" si las pruebas fallan y que ha "sobrevivido" en caso contrario. **PIT** es una popular herramienta de pruebas de mutaciones que puede añadirse como dependencia de Maven y ejecutarse desde la línea de comandos. Enumera los casos de prueba que han sobrevivido junto con una puntuación global de mutación de la aplicación. Las pruebas de mutación, aunque son muy eficaces, requieren mucho tiempo, por lo que deben utilizarse con prudencia.

El primer punto que me gustaría destacar sobre el porcentaje de cobertura de automatización es que, aunque tengas una cobertura del 100%, eso no garantiza una aplicación libre de errores! El porcentaje es simplemente una medida de cuántos casos de prueba

conocidos se automatizan: probablemente descubrirás casos desconocidos más adelante. Es importante señalar esto a las partes interesadas de la empresa y a tu equipo, ya que de lo contrario puede llevarles a cuestionar la fiabilidad del conjunto de pruebas de automatización y el valor del esfuerzo invertido en él cuando se encuentre un fallo crítico. También es crucial hacerles comprender que el resultado esperado del seguimiento de esta métrica es revelar el backlog de automatización (lo ideal es que no haya ninguno) y planificar la capacidad en las próximas iteraciones para completar estas tareas. También puedes utilizar sabiamente el seguimiento para observar si tu equipo está derivando hacia uno de los antipatrones mencionados en la sección anterior.

El segundo punto es que, cuando realices el seguimiento de la cobertura de automatización, debes observar si todas las áreas de la aplicación tienen cobertura de automatización. Especialmente cuando desarrollas aplicaciones a gran escala con distintos equipos trabajando en varios componentes, tu porcentaje de cobertura puede seguir siendo alto (digamos, >80%) aunque un módulo tenga cero pruebas, siempre que los demás módulos tengan altos porcentajes de cobertura de pruebas .

La penúltima indicación sobre la cobertura de automatización del 100% es que debes incluir tanto los casos de prueba funcionales como los interfuncionales al calcular esta métrica. La mayoría de las veces, los casos de prueba interfuncionales no contribuyen al porcentaje, lo que da lugar a errores más adelante (aprenderás más sobre la automatización de casos de prueba interfuncionales en los próximos capítulos).

Y, por último, aunque tu objetivo debe ser automatizar todos los casos de prueba, dependiendo de la naturaleza de la aplicación, los entornos, los costes de automatización, etc., puede ser imposible lograr una cobertura de automatización del 100%. En tales casos, debes hacer un seguimiento adecuado de los casos de prueba no automatizados y añadirlos a tu lista de pruebas manuales. Dicho

esto, no deberías tener una lista alta de casos de prueba manuales: iquieres evitar los 1.200 minutos de pruebas de lanzamiento sobre los que advertí al principio del capítulo!

Los mayores beneficios de todo este seguimiento meticuloso y de garantizar una cobertura de automatización adecuada empezarán a verse a medida que el proyecto crezca, sobre todo cuando se extienda a lo largo de varios años. Como suele decirse, el código sobrevive a las personas, y a menudo las pruebas automatizadas acaban siendo la única documentación viva fiable de las funcionalidades de la aplicación. Por tanto, tus esfuerzos en escribir buenas pruebas automatizadas resultarán ser una inversión que merece la pena, no sólo para el proyecto, sino para ti y tus futuros compañeros de equipo.

Puntos clave

Éstos son los puntos clave de este capítulo:

- Las pruebas automatizadas son la práctica de utilizar herramientas para verificar el comportamiento esperado de la aplicación con el fin de recibir información rápida durante el desarrollo del software.
- Una forma inteligente de equilibrar la capacidad de pruebas en un proyecto es realizar pruebas exploratorias manuales para encontrar nuevos casos de prueba y automatizarlas para ayudar en las pruebas de regresión.
- Cuando se trata de pruebas funcionales automatizadas, su alcance se amplía más allá de las pruebas funcionales de interfaz de usuario comúnmente adoptadas. Las pruebas unitarias, de integración, de contrato, de servicio, funcionales de interfaz de usuario y de extremo a extremo son los distintos tipos de pruebas a nivel micro y macro que, cuando se

entretejen adecuadamente, proporcionan una rápida retroalimentación.

- La pirámide de pruebas es el objetivo ideal al elaborar tu estrategia de pruebas funcionales automatizadas. Añadir una amplia base de pruebas de micronivel y disminuir gradualmente el número de pruebas de macronivel a medida que se amplía su alcance es la mejor forma de reducir el tiempo de creación y ejecución de las pruebas.
- Varias herramientas, incluidas las de IA/ML, han evolucionado para facilitar los esfuerzos de creación, mantenimiento y análisis de pruebas funcionales automatizadas.
- Aunque te hayas esforzado mucho en crear tus marcos de automatización en distintas capas, el trabajo no acaba ahí. Tienes que seguir atento a las señales de antipatrones, como el cono de helado y la magdalena.
- Es vital realizar un seguimiento de la cobertura de automatización para garantizar que los esfuerzos de automatización no se dejan de lado en medio del ajetreo de la entrega. Además, ten en cuenta que un alto porcentaje de cobertura de automatización puede llevar a tu equipo a una falsa sensación de seguridad; es importante mirar más allá del número para asegurarte de que tienes cobertura en todas las áreas de la aplicación .

Capítulo 4. Pruebas continuas

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

Tus esfuerzos de retroalimentación rápida están en el limbo sin una retroalimentación continua!

En el capítulo anterior, hablamos de cómo añadir pruebas en las capas adecuadas de la aplicación acelera los ciclos de retroalimentación. Es imprescindible recibir esa retroalimentación rápida *de forma continua* y no sólo en ráfagas aleatorias para regular sin fisuras la calidad de la aplicación a lo largo del ciclo de desarrollo. Este capítulo está dedicado a la elaboración de esa práctica de pruebas continuas.

Las pruebas continuas (CT) son el proceso de validar la calidad de la aplicación utilizando métodos de prueba tanto manuales como automatizados después de cada cambio incremental, y de alertar al equipo cuando el cambio provoca una desviación de los resultados de calidad previstos. Por ejemplo, cuando una parte de la funcionalidad se desvía de las cifras previstas de rendimiento de la aplicación, el proceso de TC avisa inmediatamente al equipo mediante pruebas de rendimiento fallidas. Esto da al equipo la oportunidad de solucionar los problemas lo antes posible, cuando aún son relativamente pequeños y manejables. La falta de un bucle de retroalimentación continuo de este tipo podría hacer que los problemas pasaran desapercibidos durante un período prolongado, permitiendo que con el tiempo se propagaran en cascada a niveles más profundos del código y aumentando el esfuerzo necesario para solucionarlos.

El proceso de TC se basa en gran medida en la práctica de la *integración continua* (IC) para realizar pruebas automatizadas contra cada cambio. La adopción de la IC junto con la TC permite al equipo realizar *entregas continuas* (EC). En última instancia, el trío formado por la IC, la DC y la TC convierte al equipo en un equipo de alto rendimiento, medido por las cuatro métricas clave: *tiempo de espera, frecuencia de implementación, tiempo medio de restauración y porcentaje de cambios fallidos*. Estas métricas, que veremos al final de este capítulo, proporcionan información sobre la calidad de las prácticas de entrega del equipo.

Este capítulo te dotará de las habilidades necesarias para establecer un proceso de CT para tu equipo. Aprenderás sobre los procesos CI/CD/CT y las estrategias para conseguir múltiples bucles de retroalimentación en varias dimensiones de la calidad. También se incluye un ejercicio guiado para configurar un servidor CI e integrar las pruebas automatizadas.

Bloques de construcción

Como base para la destreza en las pruebas continuas, esta sección te introducirá en la terminología y el proceso general CI/CD/CT. También aprenderás los principios fundamentales y la etiqueta que deben impregnarse cuidadosamente en el equipo para que el proceso tenga éxito. Empecemos con una introducción a la CI.

Introducción a la integración continua

[Martin Fowler](#), autor de media docena de libros, entre ellos *Refactoring: Improving the Design of Existing Code* (Addison Wesley) y científico jefe de Thoughtworks, describe la integración continua como "una práctica de desarrollo de software en la que los miembros de un equipo integran su trabajo con frecuencia, normalmente cada persona lo integra al menos a diario, lo que da

lugar a múltiples integraciones al día". Veamos un ejemplo para ilustrar las ventajas de seguir esta práctica.

Dos compañeros de equipo, Allie y Bob, empezaron a desarrollar de forma independiente un inicio de sesión y una página de inicio. Empezaron a trabajar por la mañana, y al mediodía Allie había terminado un flujo básico de inicio de sesión y Bob una estructura básica de página de inicio. Ambos probaron sus respectivas funcionalidades en sus máquinas locales y siguieron trabajando. Al final del día, Allie había completado la funcionalidad de inicio de sesión haciendo que la aplicación aterrizará en una página de inicio vacía después de iniciar sesión con éxito, ya que la página de inicio aún no estaba disponible para ella. Del mismo modo, Bob completó la funcionalidad de la página de inicio codificando el nombre de usuario en el mensaje de bienvenida, ya que no disponía de la información de usuario del inicio de sesión.

Al día siguiente, ambos informaron de que sus funcionalidades estaban "terminadas". Pero, ¿están realmente terminadas? ¿Cuál de los dos desarrolladores es responsable de integrar las páginas? ¿Deberían crear una historia de usuario de integración distinta para cada escenario de integración en toda la aplicación? Si es así, ¿estarán preparados para el gasto de los esfuerzos de prueba duplicados que supone probar la historia de integración? ¿O deberían retrasar las pruebas hasta que la integración esté hecha? Éste es el tipo de preguntas que se abordan implícitamente con la integración continua.

Cuando se siga el CI, Allie y Bob compartirán el progreso de su trabajo a lo largo del día (después de todo, ambos tenían listo un esqueleto básico de sus funcionalidades al mediodía). Bob podrá añadir el código de integración necesario para abstraer el nombre de usuario tras el inicio de sesión (por ejemplo, a partir de un token JSON o JWT), y Allie podrá hacer que la aplicación aterrice en la página de inicio real tras un inicio de sesión satisfactorio. Entonces, la aplicación será realmente utilizable y comprobable!

En este ejemplo, puede parecer un pequeño coste adicional integrar las dos páginas al día siguiente. Sin embargo, cuando el código se acumula e integra más tarde en el ciclo de desarrollo, las pruebas de integración resultan costosas y llevan mucho tiempo. Además, cuanto más se retrasen las pruebas, más probabilidades habrá de encontrar problemas enredados difíciles de solucionar, que a veces incluso justifican la reescritura de una parte importante del software. Posteriormente, esto creará un miedo generalizado a la integración entre los miembros del equipo, ia menudo un acompañamiento tácito del retraso de la integración!

La práctica de la integración continua trata esencialmente de reducir esos riesgos de integración y ahorrar al equipo reescrituras y parches ad hoc. No elimina por completo los defectos de integración, pero facilita encontrarlos y solucionarlos pronto, cuando apenas están brotando.

El proceso CI/CT/CD

Empecemos por examinar en detalle los procesos de integración y prueba continuas. Más adelante, veremos cómo se conectan para formar el proceso de entrega continua.

El proceso CI/CT se basa en cuatro componentes individuales:

- El *sistema de control de versiones* (VCS), que contiene toda la base de código de la aplicación y sirve como repositorio central del que todos los miembros del equipo pueden extraer la última versión del código y donde pueden integrar su trabajo continuamente
- Las pruebas funcionales y multifuncionales automatizadas que validan la aplicación
- El servidor CI, que ejecuta automáticamente las pruebas automatizadas contra la última versión del código de la aplicación por cada cambio adicional

- La infraestructura que aloja el servidor CI y la aplicación

El flujo de trabajo de integración y pruebas continuas comienza con el desarrollador, que, en cuanto termina una pequeña parte de la funcionalidad, introduce sus cambios en un sistema de control de versiones común (por ejemplo, Git, SVN). El VCS realiza un seguimiento de cada cambio que se le envía. A continuación, los cambios se envían a través del proceso de pruebas continuas, donde el código de la aplicación se construye completamente y se ejecutan pruebas automatizadas contra él mediante un servidor CI (por ejemplo, Jenkins, GoCD). Cuando se superan todas las pruebas, los nuevos cambios se consideran totalmente integrados. Cuando hay fallos, el propietario del código correspondiente soluciona los problemas lo antes posible. A veces, los cambios se revierten desde el VCS hasta que se resuelven los problemas. Esto se hace principalmente para evitar que otros saquen el código con problemas e integren su trabajo sobre él.

VENTAJAS DE LAS VCS

¿Has pensado alguna vez cómo compartían los equipos su código antes de los VCS? Algunos equipos utilizaban unidades de disco compartidas, y otros enviaban directamente su código a un servidor central que albergaba toda la base de código! Tal fue el dolor que llevó al desarrollo del primer VCS de la historia en los años 60, llamado the Source Code Control System (SCCS). Desde entonces, los VCS se han enriquecido, con nuevas funciones que han eliminado muchos puntos dolorosos y han ofrecido enormes ventajas para la integración laboral.

Algunas ventajas significativas son las siguientes:

- Un VCS lleva un registro de cada versión de código que se introduce en él, ya sea una adición, eliminación o modificación de código, en una base de datos independiente. Esto sirve como historial a largo plazo de los cambios y, por tanto, facilita significativamente el análisis de la causa raíz de los problemas.
- Dado que las versiones se mantienen de forma independiente, un VCS permite a los equipos volver a una versión de la aplicación que funcionaba anteriormente cuando surgen problemas.
- Los cambios en el VCS pueden vincularse a una historia de usuario o a una tarjeta de defecto. Esto da al equipo la posibilidad de rastrear los cambios hasta una historia de usuario y comprender el contexto detrás del código escrito y la evolución de una característica a lo largo del tiempo.
- A veces, los equipos pueden tener que trabajar en un área común de código para construir sus características. Un VCS permite a los miembros del equipo crear **ramas** de la base de código principal, construir sobre ellas y fusionarlas con la

base de código principal un poco más tarde. Sin embargo, una rama de características de larga vida es un antipatrón.

Como muestra [la Figura 4-1](#), Allie introduce su código para la funcionalidad básica de inicio de sesión junto con las pruebas de inicio de sesión en el sistema común de control de versiones antes del mediodía, como parte del commit C_n .

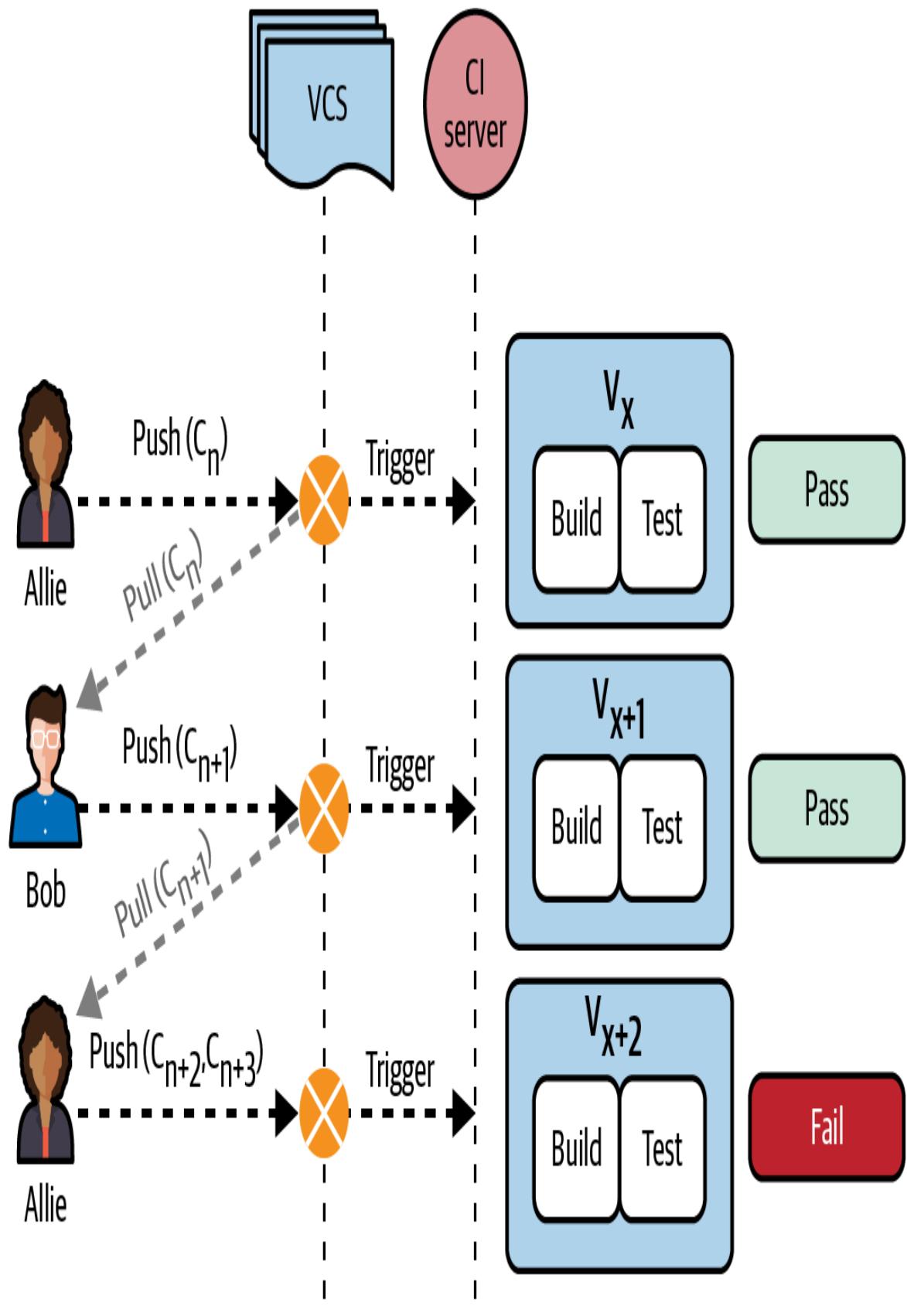


Figura 4-1. Componentes de un proceso continuo de integración y pruebas

NOTA

En el VCS Git, un *commit* es una instantánea de toda la base de código en un momento dado. Cuando se practica la integración continua, se recomienda guardar los pequeños cambios incrementales como commits independientes en la máquina local. Cuando la funcionalidad alcance un estado lógico, como completar una funcionalidad básica de inicio de sesión, los commits deben empujarse al repositorio VCS común. Sólo cuando se envían los cambios al VCS comienzan los procesos de CI y de prueba.

El nuevo cambio, C_n , activa una canalización independiente en el servidor CI. Cada canal se compone de varias etapas secuenciales. La primera es la etapa *de construcción y pruebas*, que construye la aplicación y ejecuta pruebas automatizadas contra ella. Éstas incluyen todas las pruebas a nivel micro y macro de las que se habló en [el Capítulo 3](#), y las pruebas que hacen valer las dimensiones de calidad de la aplicación (rendimiento, seguridad, etc.), de las que hablaremos en los próximos capítulos. Una vez completada esta etapa, se indican los resultados de las pruebas a Allie. En este caso, el código de Allie se ha integrado correctamente, y ella continúa con su funcionalidad de inicio de sesión.

Más tarde ese mismo día, Bob envía el commit C_{n+1} para la función de la página de inicio tras extraer los últimos cambios (C_n) del VCS común. C_{n+1} es, por tanto, una instantánea de la base de código de la aplicación que incluye los nuevos cambios de Allie y Bob. Esto desencadena la fase de compilación y prueba en el proceso CI. Cuando las pruebas se ejecutan contra C_{n+1} , se garantiza que los nuevos cambios de Bob no han roto ninguna de las funcionalidades anteriores, incluida la última confirmación de Allie, ya que ella también ha añadido las pruebas de inicio de sesión.

Afortunadamente, Bob no lo ha hecho. Sin embargo, vemos en [la](#)

Figura 4-1 que los cambios de Allie en los commits c_{n+2} y c_{n+3} han roto la integración, y las pruebas han fallado. Tiene que corregirlos antes de continuar con su trabajo, ya que ha introducido un error en el VCS común. Puede enviar su corrección como otra confirmación y el proceso continuará.

Imagina el mismo flujo de trabajo en un gran equipo distribuido, y comprenderás lo mucho que facilita la IC que todos los miembros del equipo compartan sus progresos e integren su trabajo sin problemas. Además, en las aplicaciones a gran escala, suele haber varios componentes interdependientes que justifican pruebas de integración exhaustivas, y el proceso de pruebas continuas proporciona la confianza tan necesaria en la delicadeza de su integración!

Con ese tipo de confianza obtenida de los procesos de integración y prueba totalmente automatizados, el equipo se sitúa en un lugar privilegiado para empujar su código a producción siempre que el negocio lo exija. En otras palabras, el equipo está equipado para realizar entregas continuas.

La entrega continua depende de que se sigan los procesos de integración y pruebas continuas de para que la aplicación esté lista para producción en todo momento. Además, exige disponer de un mecanismo de implementación automatizado que pueda activarse con un solo clic para desplegarse en cualquier entorno, ya sea de control de calidad o de producción. **La Figura 4-2** muestra el proceso de entrega continua.

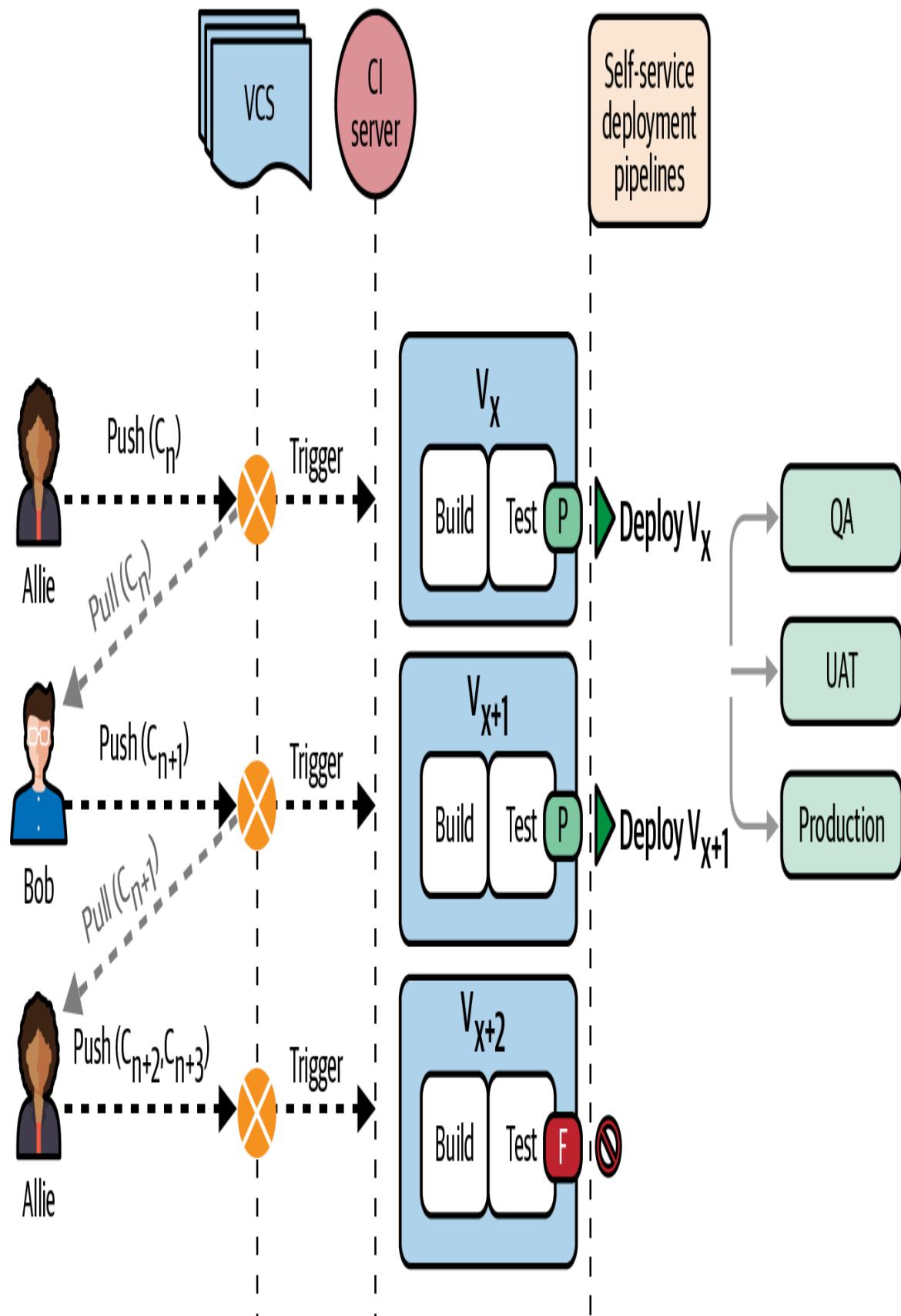


Figura 4-2. Proceso de entrega continua con CI, CT y conductos de implementación

Como puedes ver, el proceso de entrega continua engloba los procesos CI/CT junto con las canalizaciones de implementación de autoservicio. Estos conductos también son etapas configuradas en el servidor de CI; realizan la tarea de desplegar la versión "elegida" de los artefactos de la aplicación en el entorno requerido.

El servidor CI enumera todos los commits de con el estado de los resultados de sus pruebas. Sólo si se han superado todas las pruebas de una confirmación (o de un conjunto de confirmaciones), ofrece la opción de implementar esa versión concreta de la aplicación (V_x). Por ejemplo, supongamos que el equipo de Allie quiere recibir comentarios de la empresa sobre la funcionalidad básica de inicio de sesión introducida como parte de la confirmación C_n . Pueden pulsar el botón Implementación V_x , como se ve en [la Figura 4-2](#), y elegir el entorno de **pruebas de aceptación del usuario** (UAT). Esto desplegará sólo los cambios realizados hasta ese momento en el entorno UAT, es decir, no se desplegarán las confirmaciones C_{n+1} y posteriores de Bob. Como puedes ver, los commits C_{n+2} y C_{n+3} no están disponibles para su implementación, ya que las pruebas han fallado.

Este tipo de configuración de entrega continua resuelve muchos problemas críticos, pero una de las ventajas más importantes que proporciona es la capacidad de lanzar al mercado las características del producto en el momento adecuado. A menudo, los retrasos en los lanzamientos de características provocan pérdidas de ingresos y de clientes a favor de la competencia. Además, desde el punto de vista del equipo, el proceso de implementación se automatiza totalmente, reduciendo la dependencia de que determinadas personas hagan su magia el día de la implementación; cualquiera es libre de hacer una implementación sin problemas en cualquier entorno y en cualquier momento. La automatización de las Implementaciones también reduce el riesgo de bibliotecas

incompatibles, configuraciones ausentes o incorrectas y documentación insuficiente.

IMPLEMENTACIÓN CONTINUA FRENTE A ENTREGA CONTINUA

La implementación continua es diferente de la entrega continua. La implementación continua implica disponer de canalizaciones de implementación automatizadas en que envían cada confirmación a producción automáticamente tras el proceso de pruebas continuas. En otras palabras, la función que acabas de confirmar está disponible para los usuarios finales reales en producción inmediatamente. En cambio, practicar la entrega continua consiste en estar preparado en cualquier momento para pasar la aplicación a producción con una opción de implementación de autoservicio. La entrega continua es adecuada en los casos en que las empresas han fijado fechas de lanzamiento para las funciones. A veces, las empresas incluso hacen anuncios públicos sobre la inauguración de características.

Principios y etiqueta

Ahora que hemos hablado de los procesos CI/CD/CT, es importante señalar que estos procesos sólo pueden llegar a buen puerto si todos los miembros del equipo siguen un conjunto de principios bien definidos y una etiqueta . Al fin y al cabo, se trata de una forma automatizada de colaborar en su trabajo, ya sean pruebas automatizadas, código de aplicación o configuraciones de infraestructura. El equipo debe establecer estos principios al principio de su ciclo de entrega y seguir reforzándolos a lo largo del mismo. He aquí un conjunto mínimo de principios y etiqueta que un equipo deberá respetar para tener éxito:

Haz commits de código frecuentes

Los miembros del equipo deben hacer frecuentes commits de código en y enviarlos al VCS tan pronto como terminen cada pequeña pieza de funcionalidad, para que se pruebe y esté disponible para que otros construyan sobre ella.

Confirma siempre el código autocomprobado

Cada vez que se confirma un nuevo fragmento de código de , debe ir acompañado de pruebas automatizadas en la misma confirmación. Martin Fowler denomina a esta práctica **autocomprobación del código**. Por ejemplo, como hemos visto antes, Allie confirmó su funcionalidad de inicio de sesión junto con las pruebas de inicio de sesión. Esto garantizó que su confirmación no se rompiera cuando Bob confirmara su código a continuación.

Adherirse a la Prueba de Certificación de Integración Continua

Cada miembro del equipo debe asegurarse de que su commit supera el proceso de pruebas continuas antes de pasar al siguiente conjunto de tareas. Si las pruebas fallan, deben repararlas inmediatamente. Según la **Prueba de Certificación de Integración Continua** de Martin Fowler, una etapa de compilación y prueba rota debe repararse en 10 minutos. Si esto no es posible, debe revertirse el commit roto, dejando el código estable (o verde).

No ignores/comentes las pruebas que fallan

Con las prisas por hacer pasar la fase de compilación y la fase de pruebas, los miembros del equipo no deben comentar e ignorar las pruebas que fallan. Por evidentes que sean las razones por las que esto no debe hacerse, es una práctica habitual.

No empujes a una construcción rota

El equipo no debe empujar su código cuando la fase de construcción y pruebas esté rota (o en rojo). Empujar trabajo sobre una base de código ya rota hará que las pruebas vuelvan a fallar. Esto sobrecargaría aún más al equipo con la tarea adicional de encontrar qué cambios rompieron originalmente la compilación.

Asume la responsabilidad de todos los fracasos

Cuando las pruebas fallan en un área de código en la que alguien no ha trabajado, pero falla debido a sus cambios, la responsabilidad de arreglar la construcción sigue recayendo sobre ellos. Si es necesario, pueden emparejarse con alguien que tenga los conocimientos necesarios para arreglarlo, pero en última instancia arreglarlo antes de pasar a su siguiente tarea es un requisito fundamental. Esta práctica es esencial porque, a menudo, la responsabilidad de arreglar las pruebas fallidas se reparte entre unos y otros, lo que provoca un retraso en la resolución de los problemas. A veces las pruebas dejan de ejecutarse en el CI durante días mientras no se soluciona el problema. Esto hace que el proceso de pruebas continuas proporcione información incompleta o falsa sobre los cambios introducidos durante esa ventana abierta.

Muchos equipos también adoptan prácticas más estrictas en su propio beneficio, como exigir que todas las pruebas de nivel micro y macro se superen en las máquinas locales antes de enviar la confirmación al VCS, suspender la fase de creación y prueba si una confirmación no alcanza el umbral de cobertura del código, publicar el estado de la confirmación (aprobado o suspenso) con el nombre de la persona que la ha realizado para todo el mundo en un canal de comunicación como Slack, poner música a todo volumen en el área del equipo cada vez que se rompe una creación desde un monitor de CI dedicado, etcétera. Además, como probador del equipo, vigilo el estado de las pruebas en el CI y me ocupo de que se corrijan a

tiempo. Fundamentalmente, todas estas medidas se toman para racionalizar las prácticas del equipo en torno a los procesos de CI/CT y obtener así los beneficios adecuados, aunque la medida principal que siempre parece funcionar mejor es dotar al equipo de conocimientos sobre no sólo el "cómo", sino también el "por qué" del proceso.

Estrategia de pruebas continuas

Ahora que conoces los procesos y principios, el siguiente paso es crear y aplicar estrategias personalizadas a las necesidades de tu proyecto.

En la sección anterior, se demostró el proceso de pruebas continuas de con una única etapa de compilación y pruebas que ejecuta todas las pruebas y proporciona retroalimentación en un único bucle.

También puedes acelerar el ciclo de retroalimentación con dos bucles de retroalimentación independientes: uno que ejecute las pruebas contra el código estático de la aplicación (por ejemplo, todas las pruebas de micronivel), y otro que ejecute las pruebas de macronivel contra la aplicación implementada. Esto, en cierto modo, es un ligero desplazamiento a la izquierda en el que aprovechamos la capacidad de las pruebas de micronivel (unidad, integración, contrato) para ejecutarse más rápidamente que las pruebas de macronivel (API, interfaz de usuario, extremo a extremo) para obtener una retroalimentación más rápida.

La Figura 4-3 muestra un proceso de TC con dos etapas. Como puedes ver aquí, una práctica habitual es combinar la compilación de la aplicación con las pruebas de micronivel como una única etapa en CI. Esto se denomina tradicionalmente etapa *de compilación y pruebas*. Cuando el equipo se adhiere a la pirámide de pruebas, como comentamos en [el Capítulo 3](#), las pruebas de micronivel acabarán validando una amplia gama de funcionalidades de la aplicación. Como resultado, esta etapa les ayuda a obtener

rápidamente una amplia retroalimentación sobre el commit. La etapa de construcción y pruebas debe ser lo suficientemente rápida como para terminar su ejecución en pocos minutos, de modo que, según los principios y la etiqueta recomendados, el equipo espere a que finalice antes de pasar a la siguiente tarea. Si tarda más, el equipo debe encontrar formas de mejorarla, por ejemplo, paralelizando la fase de construcción y prueba para cada componente en lugar de tener una única fase para toda la base de código.¹

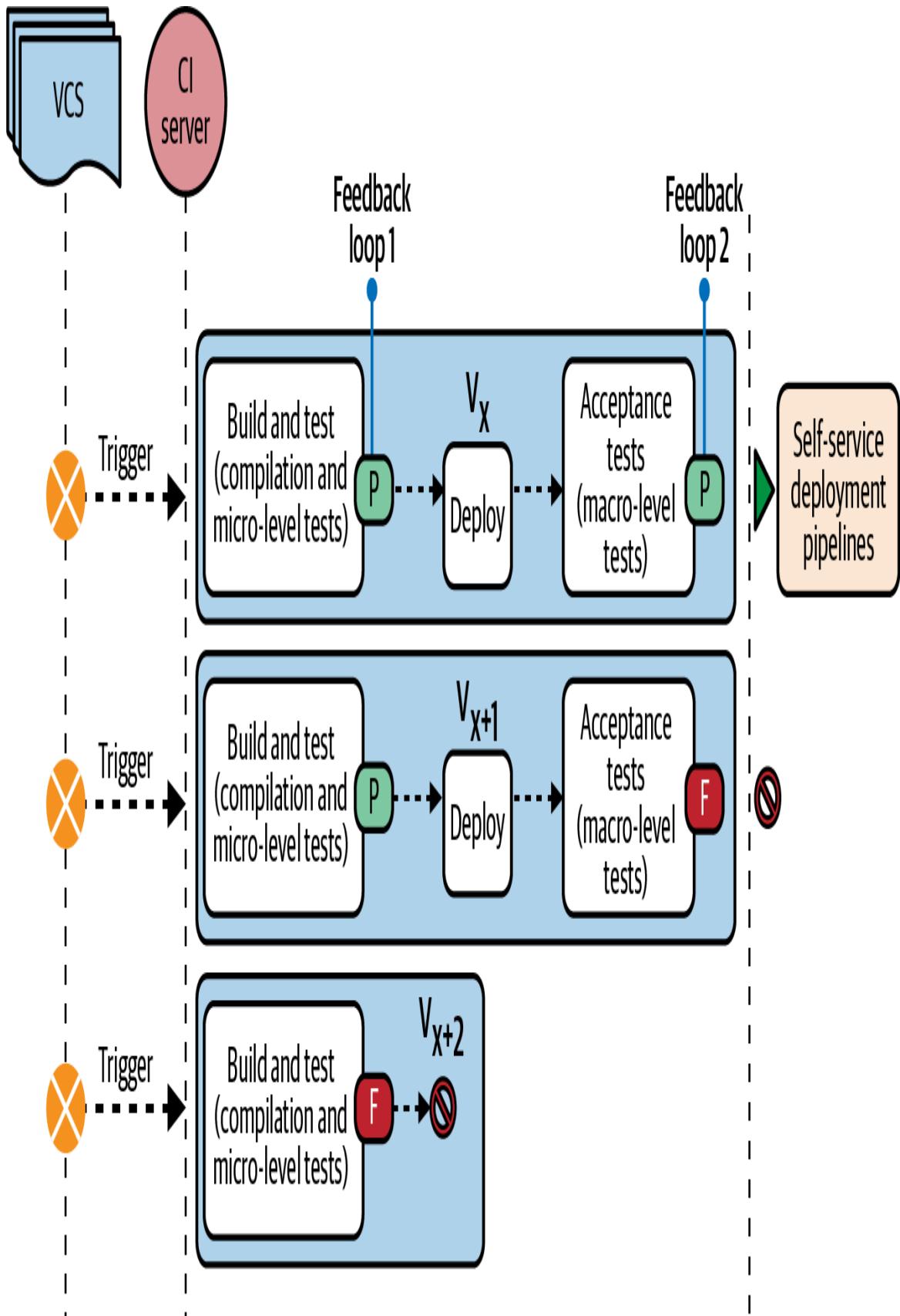


Figura 4-3. El proceso de prueba continua con dos bucles de retroalimentación

NOTA

En su libro *Entrega Continua* (Addison-Wesley Professional), Jez Humble y David Farley sugieren que la fase de construcción y prueba debe ser lo suficientemente corta como para que ocupe "más o menos el tiempo de que puedes dedicar a prepararte una taza de té, una charla rápida, consultar tu correo electrónico o estirar los músculos".

En cuanto se supera la etapa de construcción y prueba , la etapa de *implementación* envía los artefactos de la aplicación a un entorno CI (a veces llamado entorno de desarrollo). La siguiente etapa, denominada *etapa de pruebas funcionales* o *etapa de pruebas de aceptación*, ejecuta las pruebas de macrónivel contra la aplicación desplegada en el entorno CI. Sólo cuando se supera esta etapa, la aplicación está lista para la implementación de autoservicio en otros entornos de nivel superior, como QA, UAT y producción.

La retroalimentación de esta etapa puede llevar más tiempo, ya que las pruebas de aceptación tardan más en ejecutarse, y la etapa se activa después de la implementación de la aplicación, lo que también lleva tiempo. Pero cuando los equipos aplican correctamente la pirámide de pruebas, los dos bucles de retroalimentación deberían tardar menos de una hora en completarse. El ejemplo que di en [el Capítulo 3](#) lo corrobora: cuando el equipo tenía ~200 pruebas de macrónivel, tardaban 8 horas en obtener retroalimentación, pero cuando reimplimentaron su estructura de pruebas para ajustarse a la pirámide de pruebas, tardaron sólo unos 35 minutos desde el commit hasta estar listos para la implementación de autoservicio con ~470 pruebas de micronivel y macrónivel.

Otra consideración es que, cuando el bucle de retroalimentación es corto, los miembros del equipo pueden seguir dando prioridad a la solución de los problemas detectados en el proceso de pruebas

continuas, aunque hayan retomado una nueva tarea poco después de la fase de construcción y pruebas. Si tardan varias horas, pueden tener la tentación de ignorar las pruebas que fallan y seguir las como tarjetas de defectos para arreglarlas más tarde. Esto es perjudicial, ya que significa que están integrando su nuevo código sobre los defectos, y el nuevo código tampoco se prueba a fondo al ignorarse las pruebas que fallan. Por tanto, el equipo debe seguir monitorizando y adoptando formas de acelerar los dos bucles de retroalimentación mediante técnicas como paralelizar la ejecución de las pruebas, implementar la pirámide de pruebas, eliminar las pruebas duplicadas y refactorizar las pruebas para eliminar las esperas y abstraer las funcionalidades comunes.²

Este proceso de pruebas continuas puede ampliarse aún más para recibir retroalimentación interfuncional, como se muestra en la **Figura 4-4**. Los equipos pueden ejecutar pruebas automatizadas de rendimiento, seguridad y accesibilidad como parte de los dos bucles de retroalimentación existentes o configurar etapas separadas posteriores a la etapa de pruebas de aceptación en el servidor CI, logrando el objetivo de recibir retroalimentación rápida y continua sobre la calidad de la aplicación de forma holística. En los próximos capítulos aprenderás estrategias de cambio para las pruebas interfuncionales.

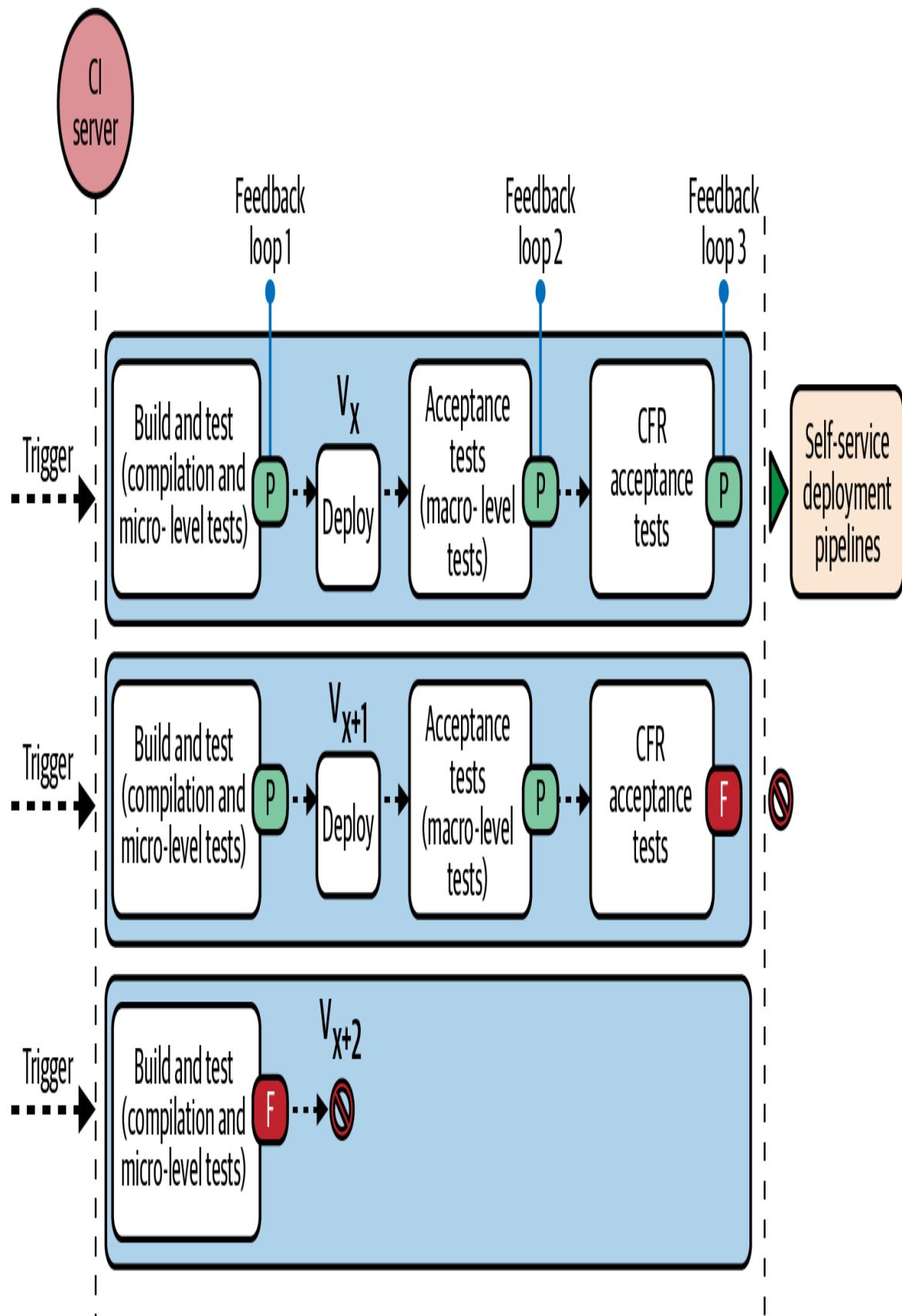


Figura 4-4. El proceso de prueba continua con tres bucles de retroalimentación

INTEGRACIÓN CONTINUA FREnte A PRUEBAS CONTINUAS

Como su nombre indica, el proceso de integración continua termina con la etapa de construcción y prueba. Es decir, un commit sólo se considera integrado cuando supera las pruebas de micronivel (al menos las pruebas unitarias).³

El proceso de pruebas continuas abarca la validación del comportamiento holístico de la aplicación, incluidos sus aspectos funcionales y multifuncionales, *para cada commit*, con el objetivo de garantizar que está lista para la entrega continua. De hecho, las pruebas continuas no terminan con la ejecución de pruebas automatizadas, sino que incluyen los esfuerzos de pruebas exploratorias manuales para cada commit después de la implementación de autoservicio. El proceso de TC también requiere que el equipo automatice los escenarios encontrados durante las pruebas exploratorias para poder llamar a la funcionalidad o commit "hecho".

Llegados a este punto, si ejecutas todas las pruebas de forma encadenada, puede que necesites mucho tiempo y recursos para terminar todas las etapas. Una forma de planificar estratégicamente el proceso de TC en este caso es dividir las pruebas en *pruebas de humo* y pruebas de *regresión nocturnas*, como se ve en [la Figura 4-5](#).

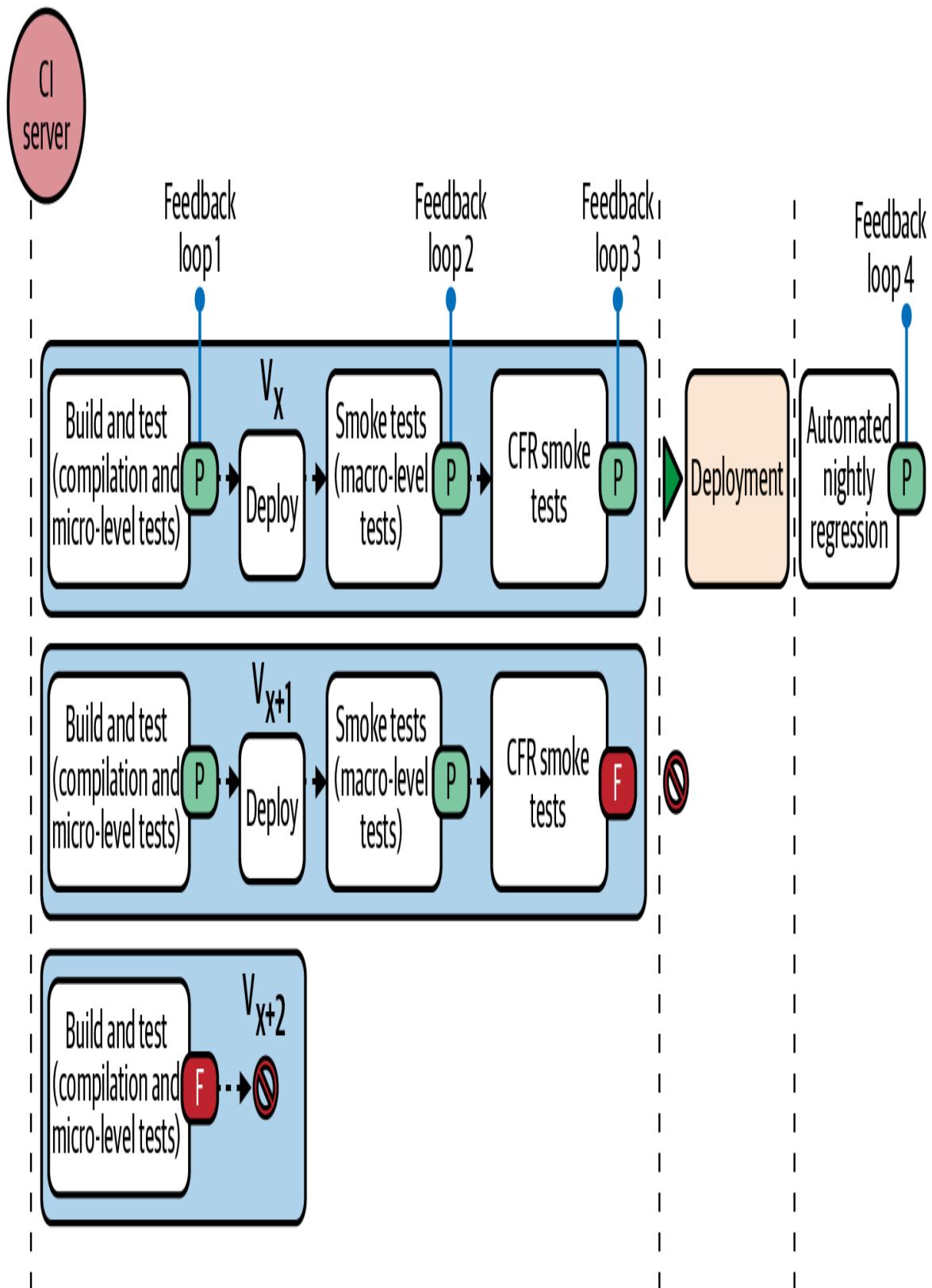


Figura 4-5. El proceso de prueba continua con cuatro bucles de retroalimentación

La prueba de humo es un término tomado de el mundo de la ingeniería eléctrica, en el que se hace pasar electricidad una vez completado el circuito para evaluar el flujo de extremo a extremo. Cuando haya problemas en el circuito, habrá humo (de ahí el nombre). Del mismo modo, puedes elegir las pruebas que cubren el flujo de extremo a extremo de cada función de la aplicación para formar el paquete de pruebas de humo y ejecutarlas sólo como parte de la fase de pruebas de aceptación. De este modo, puedes obtener rápidamente una señal de alto nivel sobre el estado de cada commit. Como se ve en [la Figura 4-5](#), la confirmación está lista para la implementación de autoservicio tras la fase de pruebas de humo.

Cuando decides realizar pruebas de humo, tienes que complementarlas con la regresión nocturna. La etapa de regresión nocturna se configura en el servidor de CI para que ejecute todo el conjunto de pruebas una vez al día, cuando el equipo no trabaja (por ejemplo, puede programarse para que se ejecute todos los días a las 7 de la tarde). Las pruebas se ejecutan con el último código base con todos los commits del día. El equipo debe acostumbrarse a analizar los resultados de la regresión nocturna a primera hora del día siguiente y priorizar la corrección de defectos y fallos del entorno. A veces esto puede requerir cambios en los scripts de prueba, y eso también debe priorizarse para el día, de modo que el proceso de pruebas continuas proporcione la información adecuada para los próximos commits.

Puedes aplicar estas dos estrategias para dividir las pruebas funcionales y multifuncionales. Por ejemplo, puedes optar por ejecutar la prueba de carga de rendimiento para un único punto final crítico como parte de cada commit y ejecutar el resto de pruebas de rendimiento como parte de la regresión nocturna (las pruebas de rendimiento se tratan en el [Capítulo 8](#)). Del mismo modo, puedes ejecutar las pruebas de exploración de seguridad del código estático como parte de la etapa de compilación y prueba, y ejecutar las pruebas de exploración de seguridad funcional (tratadas en el

Capítulo 7) como parte de la etapa de regresión nocturna. Aunque resulte obvio, la advertencia de este enfoque es que la retroalimentación se retrasa un día. En consecuencia, también se retrasa la corrección de la respuesta; los problemas se rastrean como defectos y se corrigen más tarde. En consecuencia, debes tener cuidado al elegir los tipos de pruebas que ejecutas como parte de las etapas de prueba de humo y regresión nocturna. Además, ten en cuenta que sólo las pruebas de nivel macro y multifuncionales deben clasificarse como pruebas de humo; todas las pruebas de nivel micro deben seguir ejecutándose como parte de la fase de construcción y prueba.

La mayoría de las veces, cuando la aplicación es joven, puedes renunciar a estas estrategias y disfrutar del privilegio de ejecutar todas las pruebas para cada commit. Luego, cuando la aplicación empiece a crecer (junto con el número de pruebas), puedes aplicar los distintos métodos de optimización de tiempo de ejecución de CI, y finalmente seguir el camino de las pruebas de humo y la regresión nocturna.

Beneficios

Si te estás preguntando si todo ese esfuerzo por emprender un proceso de pruebas continuas dará frutos que merezcan la pena, **la Figura 4-6** muestra algunas ventajas para que tú y tu equipo os motivéis.

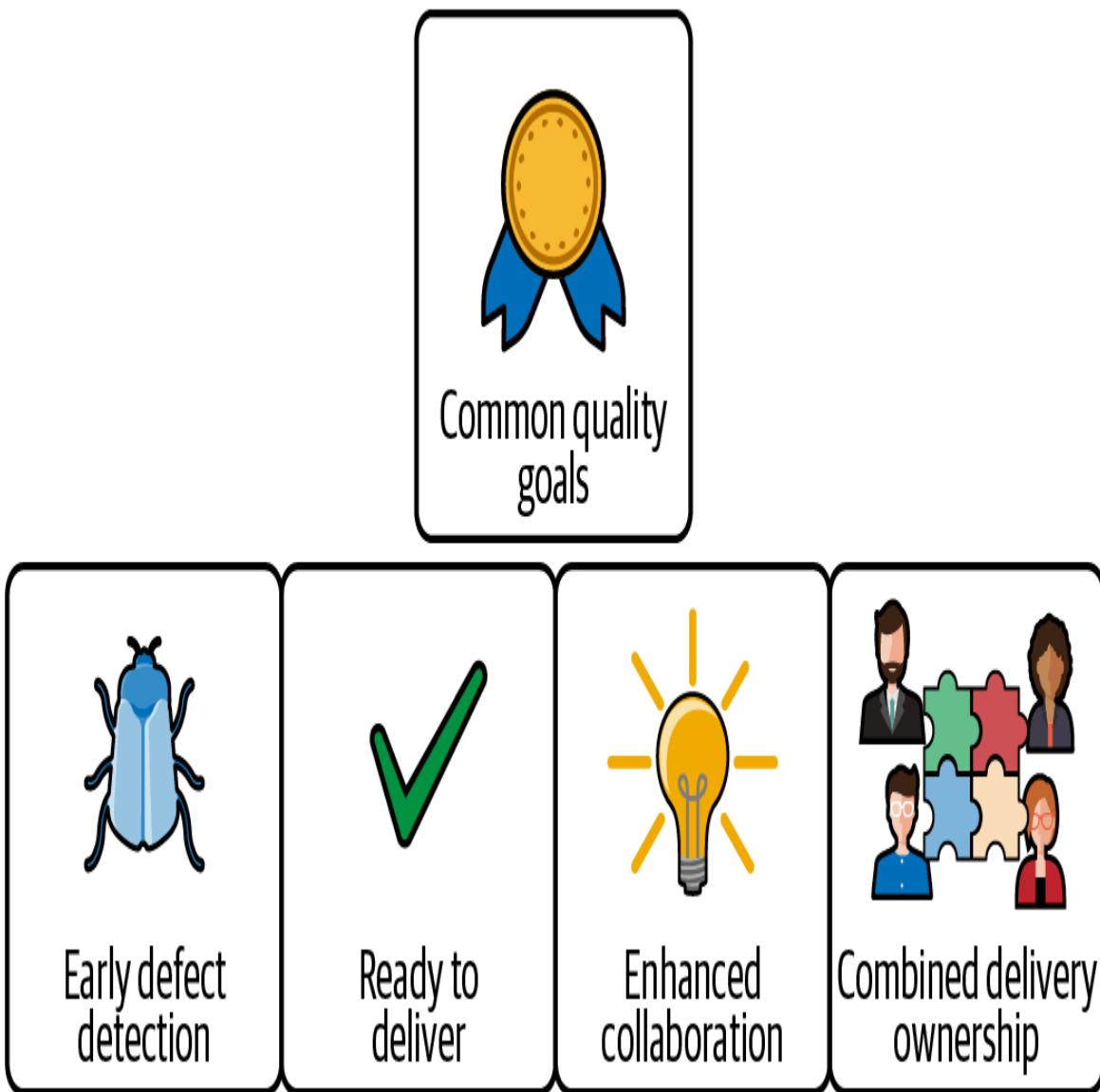


Figura 4-6. Ventajas del proceso de prueba continua

Veamos cada una de ellas por separado:

Objetivos comunes de calidad

Seguir el proceso de pruebas continuas garantiza que todos los miembros del equipo sean conscientes de un objetivo de calidad común y trabajen para alcanzarlo -en términos de aspectos de calidad tanto funcionales como interfuncionales-, ya que su trabajo se evalúa continuamente en relación con ese objetivo. Es una forma concreta de incorporar la calidad.

Detección precoz de defectos

Todos los miembros del equipo reciben comentarios inmediatos de sobre sus commits, tanto en lo que se refiere a los aspectos funcionales como a los interfuncionales. Esto les da la oportunidad de solucionar los problemas mientras disponen del contexto pertinente, en lugar de volver al código unos días o semanas más tarde.

Listo para entregar

Como el código se prueba continuamente en , la aplicación siempre está lista para su implementación en cualquier entorno.

Mayor colaboración

Es más fácil colaborar con miembros del equipo distribuidos en que comparten su trabajo y hacer un seguimiento de qué commit causó qué problemas, evitando así acusaciones y limitando la animosidad.

Propiedad de entrega combinada

La responsabilidad de la entrega se distribuye en entre todos los miembros del equipo, en lugar de sólo entre el equipo de pruebas o los desarrolladores principales, ya que todos son responsables de garantizar que sus commits están listos para la implementación.

Si llevas un tiempo trabajando en la industria del software , iseguro que sabes lo difícil que es conseguir algunos de estos beneficios de otra forma!

Ejercicio

Es hora de ponerse manos a la obra. El ejercicio guiado te mostrará cómo transferir las pruebas automatizadas que creaste en el [Capítulo 3](#) a un VCS, configurar un servidor CI e integrar las pruebas automatizadas con el servidor CI de forma que, cada vez que envíes una confirmación al VCS, se ejecuten las pruebas automatizadas. Aprenderás a utilizar Git y Jenkins como parte de este ejercicio.

Git

Desarrollado originalmente en 2005 por Linus Torvalds, creador del núcleo del sistema operativo Linux, Git es el sistema de control de versiones de código abierto más utilizado . Según la [encuesta de Stack Overflow](#) de 2021, el 90% de los encuestados utilizan Git. Es un sistema de control de versiones distribuido, lo que significa que cada miembro del equipo obtiene una copia de todo el código base junto con el historial de cambios. Esto da a los equipos mucha flexibilidad en términos de depuración y trabajo independiente.

Configurar

Para empezar, necesitarás un lugar donde alojar tu base de código en . GitHub y Bitbucket son empresas que proporcionan ofertas basadas en la nube para alojar repositorios Git (un repositorio, en términos sencillos, es una ubicación de almacenamiento para tu base de código). GitHub permite alojar repositorios públicos de forma gratuita, lo que lo hace popular, especialmente entre la comunidad de código abierto. Así que para este ejercicio, si aún no tienes una cuenta en GitHub, [crea una ahora](#).

En tu cuenta de GitHub, navega hasta Tus repositorios → Nuevo para crear un nuevo repositorio para tus pruebas automatizadas de Selenium. Dale un nombre al repositorio, por ejemplo *PruebasFuncionales*, y conviértelo en un repositorio público. Si lo creas correctamente, accederás a la página de configuración del

repositorio. Anota la URL de tu repositorio (<https://github.com/<tunombredeusuario>/PruebasFuncionales.git>). La página también te dará una serie de instrucciones para enviar tu código al repositorio utilizando comandos Git. Tendrás que instalar y configurar Git en tu máquina para ejecutarlos.

Para ello, sigue estos pasos:

1. Descarga e instala Git desde tu símbolo del sistema utilizando los siguientes comandos:

```
// macOS  
$ brew install git  
// Linux  
$ sudo apt-get install git
```

Si utilizas Windows, descarga el instalador desde el [sitio oficial de Git para Windows](#).

2. Verifica la instalación ejecutando el siguiente comando:

```
$ git --version
```

3. Siempre que hagas un commit, es necesario vincularlo a un nombre de usuario y a una dirección de correo electrónico con fines de seguimiento. Proporciona los tuyos a Git con estos comandos para que los adjunte automáticamente cuando hagas una confirmación:

```
$ git config --global user.name "yourUsername"  
$ git config --global user.email "yourEmail"
```

4. Verifica la configuración ejecutando este comando:

```
$ git config --global --list
```

Flujo de trabajo

El flujo de trabajo en Git tiene cuatro etapas por las que se moverá tu código, como se ve en la [Figura 4-7](#). Cada etapa tiene un propósito diferente, como aprenderás.

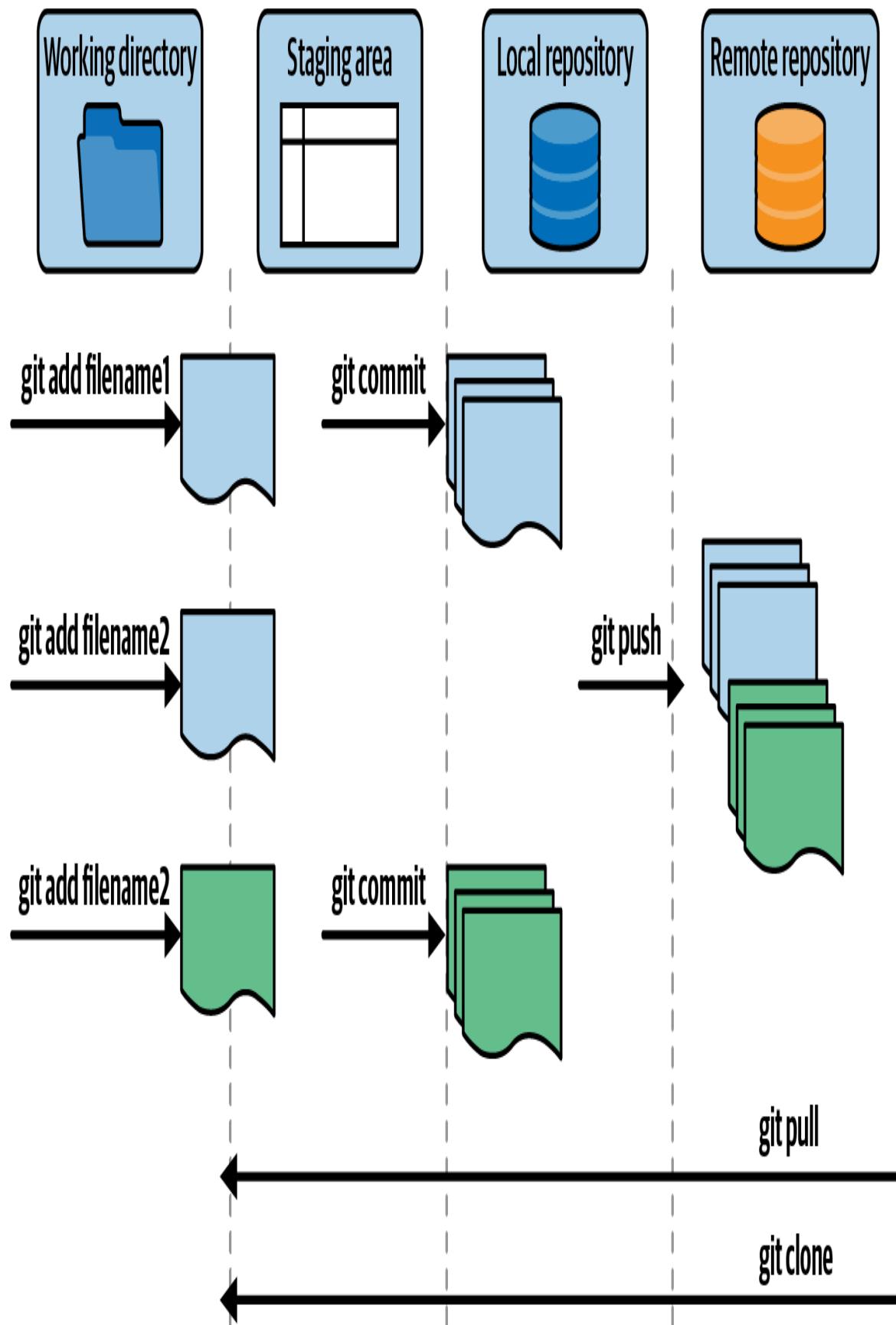


Figura 4-7. Flujo de trabajo Git con cuatro etapas

La primera etapa es tu *directorio de trabajo*, donde realizas cambios en tu código de prueba (añades nuevas pruebas, arreglas guiones de prueba, etc.). La segunda etapa es el *área de preparación* local a la que añades cada pequeño trozo de trabajo, como la creación de una clase de página, a medida que lo terminas. Esto te permite hacer un seguimiento de los cambios que vas haciendo para poder revisarlos y reutilizarlos más adelante. La tercera etapa es tu *repositorio local*. Como se ha mencionado antes, Git proporciona a todo el mundo una copia de todo el repositorio junto con el historial en su máquina local. Una vez que tengas una estructura de pruebas que funcione, puedes hacer un commit que moverá todo lo que hayas añadido a la zona de pruebas a tu repositorio local. Esto facilita la reversión de todo el código como un único trozo cuando se produzcan fallos. Una vez que hayas terminado con todos los cambios necesarios -en este caso, cuando hayas completado una prueba y quieras que se ejecute como parte de la tubería CI- puedes enviarla al repositorio remoto. La nueva prueba también estará disponible para todos los miembros de tu equipo.

Los comandos Git para mover el código a través de las distintas etapas se muestran en la [Figura 4-7](#). Puedes probarlos ahora paso a paso como se indica a continuación:

1. En tu terminal, ve a la carpeta donde creaste tus pruebas automatizadas de Selenium en [el Capítulo 3](#). Ejecuta el siguiente comando para inicializar el repositorio Git:

```
$ cd /path/to/project/  
$ git init
```

Este comando creará la carpeta *.git* en tu directorio de trabajo actual.

2. Añade todo tu conjunto de pruebas al área de preparación ejecutando el siguiente comando:

```
$ git add .
```

En su lugar, puedes añadir un archivo (o directorio) concreto con `git add filename`.

3. Consigna tus cambios en el repositorio local con un mensaje legible que explique el contexto de la consignación ejecutando el siguiente comando con el texto de mensaje adecuado:

```
$ git commit -m "Adding functional tests"
```

Puedes combinar los pasos 2 y 3 añadiendo al parámetro opcional `-a`; es decir, `git commit -am "message"`.

4. Para enviar tu código al repositorio público, primero debes proporcionar su ubicación a tu Git local. Para ello, ejecuta el siguiente comando:

```
$ git remote add origin
```

<https://github.com/<yourusername>/FunctionalTests.git>

5. El siguiente paso es enviarlo al repositorio público. Tienes que autenticarte proporcionando tu nombre de usuario de GitHub y tu código de acceso personal al enviar. Un token de acceso personal es una contraseña de corta duración que GitHub exige para todas las operaciones a partir de agosto de 2021, por motivos de seguridad. Para obtener tu token de acceso personal, ve a tu cuenta de GitHub, navega hasta Configuración → Configuración de desarrollador → "Tokens de acceso personal", haz clic en "Generar nuevo token" y rellena los

campos requeridos. Utiliza el token cuando se te solicite después de ejecutar el siguiente comando:

```
$ git push -u origin master
```

CONSEJO

Si no quieres autenticarte cada vez que interactúes con el repositorio público, puedes optar por [configurar el mecanismo de autenticación SSH](#).

6. Abre tu cuenta de GitHub y verifica el repositorio.

Cuando trabajes con miembros del equipo, tendrás que extraer su código a tu máquina desde el repositorio público. Puedes hacerlo ejecutando el comando `git pull`. Si ya tienes un repositorio de pruebas funcionales para tu equipo, puedes utilizar `git clone repoURL` para obtener tu copia del repositorio local en lugar de `git init`.

Otros comandos de Git, como `git merge`, `git fetch`, y `git reset`, nos hacen la vida más fácil. Explóralos en la [documentación oficial](#) cuando sea necesario.

Jenkins

El siguiente paso es configurar un servidor Jenkins CI en tu máquina local e integrar las pruebas automatizadas de tu repositorio Git.

NOTA

La intención de esta parte del ejercicio es que comprendas cómo se pueden implementar en la práctica las pruebas continuas utilizando herramientas de CI/CD, no enseñarte DevOps. Los equipos pueden contratar a desarrolladores con conocimientos especializados de DevOps o tener una función de DevOps para gestionar el trabajo de creación y mantenimiento de canalizaciones CI/CD/CT. Sin embargo, es esencial que tanto los desarrolladores como los probadores estén familiarizados con el proceso CI/CD/CT y su funcionamiento, ya que interactuarán con este proceso y depurarán los fallos de primera mano. Además, desde el punto de vista de las pruebas, es fundamental aprender a adaptar el proceso de CT a las necesidades específicas del proyecto y asegurarse de que las fases de prueba se encadenen correctamente, según la estrategia de CT del equipo.

Configurar

Jenkins es un servidor CI de código abierto. Para utilizarlo en , **descarga** el paquete de instalación para tu sistema operativo y sigue el procedimiento de instalación estándar. Una vez instalado, inicia el servicio Jenkins. En macOS, puedes instalar e iniciar el servicio Jenkins utilizando los comandos brew como se indica a continuación:

```
$ brew install jenkins-lts  
$ brew services start jenkins-lts
```

Cuando el servicio se haya iniciado correctamente, abre la interfaz web de Jenkins en <http://localhost:8080/>. El sitio te guiará a través de las siguientes actividades de configuración:

1. Desbloquea Jenkins con una contraseña de administrador única que se generó como parte del proceso de instalación. La página web te mostrará la ruta a la ubicación de esta contraseña en tu máquina local.
2. Descarga e instala los plug-ins de Jenkins más utilizados.

3. Crea una cuenta de administrador. Con esta cuenta entrarás siempre en Jenkins.

Después de la configuración inicial, se te llevará a la página del Panel de Jenkins, como se ve en la [Figura 4-8](#).



search



2



Gayathri Mohan

log out

Dashboard ➔



Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle



Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job



Set up a distributed build

Set up an agent



Configure a cloud



Learn more about distributed builds



Figura 4-8. Vista del panel de Jenkins

NOTA

Aunque para este ejercicio estás configurando un servidor de CI en tu máquina local, en la práctica el servidor de CI estará alojado en la nube o en una máquina virtual en la misma red para que todos los miembros del equipo de puedan acceder a él.

Flujo de trabajo

Ahora, sigue estos pasos para configurar una nueva canalización para tus pruebas automatizadas:

1. Desde el Panel de Control de Jenkins, ve a Gestiónar Jenkins → Configuración Global de la Herramienta para configurar las variables de entorno JAVA_HOME y MAVEN_HOME, como se ve en [4-9](#) y [4-10](#). Puedes escribir el comando mvn -v en tu terminal para obtener ambas ubicaciones.

JDK



Figura 4-9. Configuración de JAVA_HOME en Jenkins

Maven

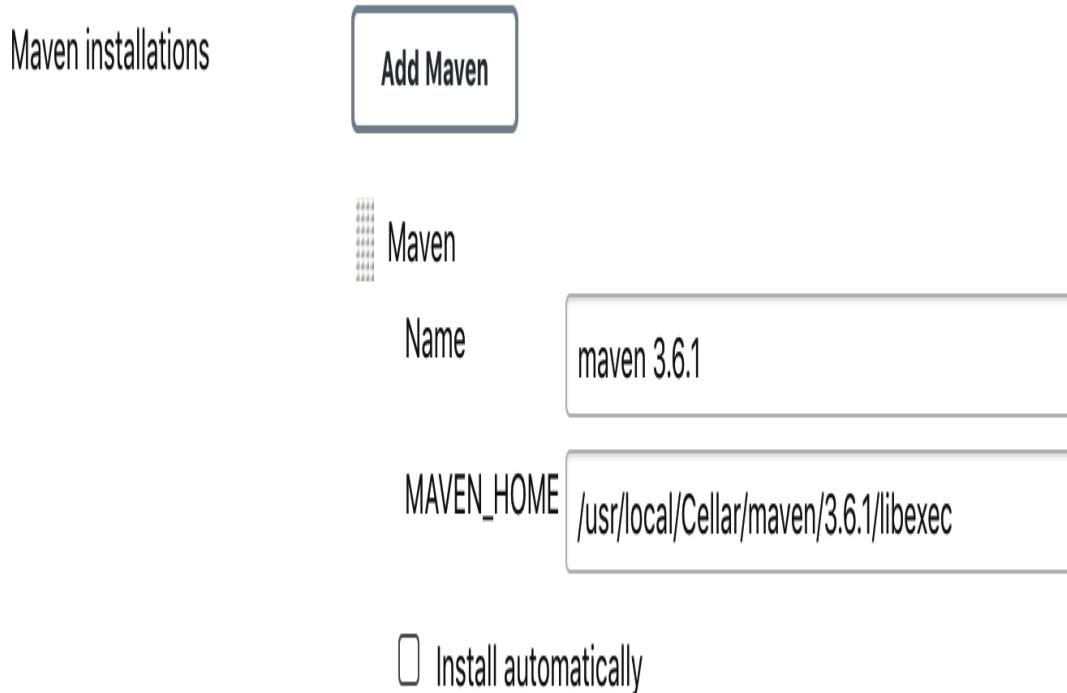


Figura 4-10. Configuración de MAVEN_HOME en Jenkins

2. Volviendo a la vista Panel de control, selecciona la opción Nuevo elemento en el panel izquierdo para crear una nueva canalización. Introduce un nombre para la canalización, por ejemplo "Pruebas funcionales", y elige la opción "Proyecto de estilo libre". Esto te llevará a la página de configuración de la canalización, como se ve en la Figura 4-11.

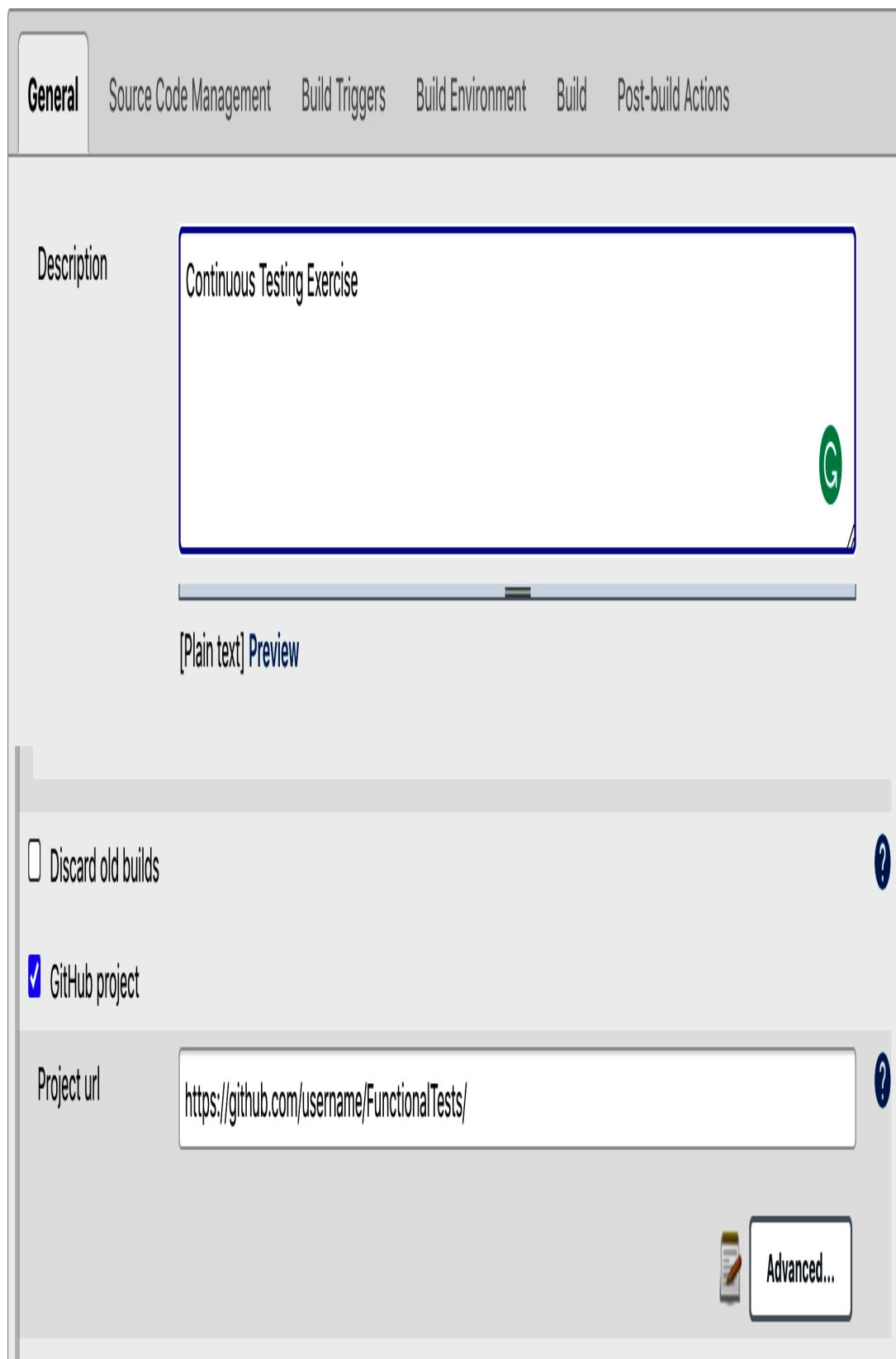


Figura 4-11. Página de configuración del canal de Jenkins

3. Introduce los siguientes datos para configurar tu canalización:

- En la pestaña General, añade una descripción de la canalización. Selecciona "Proyecto GitHub" e introduce la URL de tu repositorio (sin la extensión `.git`).
- En la pestaña Gestión del Código Fuente, selecciona Git e introduce la URL de tu repositorio (esta vez con la extensión `.git`). Jenkins lo utilizará para hacer `git clone`.
- La pestaña Desencadenantes de la compilación proporciona algunas opciones para configurar cuándo y cómo poner en marcha el canal de forma automatizada . Por ejemplo, la opción Sondear SCM puede utilizarse para sondear el repositorio Git cada dos minutos para comprobar si hay nuevos cambios y, si los hay, iniciar la ejecución de la prueba. La opción Construir periódicamente puede utilizarse para programar la ejecución de la prueba a intervalos fijos, aunque no haya nuevos cambios en el código. Esto puede utilizarse para configurar regresiones nocturnas. Del mismo modo, la opción "GitHub hook trigger for GITScm polling" configura un plug-in de GitHub para que envíe un trigger a Jenkins cada vez que haya nuevos cambios. Para simplificarlo, elige Sondear SCM e introduce este valor para sondear el repositorio de pruebas funcionales cada dos minutos: `H/2 * * * *`.
- Como tu marco de pruebas funcionales Selenium WebDriver utiliza Maven, selecciona la opción "Invocar objetivos Maven de nivel superior" en la pestaña Construir. Elige tu Maven local, que configuraste en el [Capítulo 3](#). En el campo Objetivos, introduce la fase del ciclo de vida de Maven que debe ejecutar la canalización: `test`. Esto ejecutará el comando `mvn test` desde el directorio del proyecto.

- La pestaña Acciones posteriores a la compilación es donde puedes encadenar varios pipelines, es decir, activar el pipeline de pruebas CFR después de que haya pasado el pipeline de pruebas funcionales y crear un pipeline CD completo.⁴
4. Guarda y navega hasta la vista Panel de Control. Verás la canalización creada, como se ve en la [Figura 4-12](#).

The screenshot shows the Jenkins Pipeline Control interface. At the top, there's a search bar with 'All' selected and a '+' button for creating new pipelines. To the right is a blue 'add description' button. Below the search bar is a header row with columns: S, W, Name ↓, Last Success, Last Failure, and Last Duration. The 'Functional Tests' pipeline is listed in the main table. It has a blue gear icon, a black gear icon with a lightning bolt, and a green circular progress bar icon. The details for the pipeline are: Name: Functional Tests, Last Success: 1 day 12 hr - #13, Last Failure: 1 day 12 hr - #12, and Last Duration: 22 sec. At the bottom left, there are 'Icon: SML' buttons. At the bottom right, there's a 'Legend' section with three items: 'Atom feed for all' (blue), 'Atom feed for failures' (red), and 'Atom feed for just latest builds' (green).

Figura 4-12. Tu canalización en el Panel de Jenkins

5. Haz clic en el nombre de la canalización en la vista Panel de control y, en la página de destino, selecciona la opción Construir ahora en el panel izquierdo. La canalización clonará el repositorio en tu máquina local y ejecutará el comando `mvn test`. Puedes ver que el navegador Chrome se abre y se cierra como parte de la ejecución de la prueba.
6. Localiza la carpeta *Espacio de trabajo* en la misma página. En esta carpeta encontrarás la copia local clonada del código del

repositorio y los informes generados tras la ejecución de las pruebas; puedes utilizarla para depurar.

7. En la parte inferior del panel izquierdo de la misma página, selecciona el recuento de construcciones de la ejecución actual del pipeline. Tendrás la opción de ver la salida de la consola en el panel izquierdo de la página de inicio. Esta vista mostrará las actividades de ejecución en directo para su depuración.

Enhorabuena, ieso completa tu configuración CI!

En la misma línea, tendrás que añadir las etapas de las pruebas respectivas (código estático, aceptación, humo, CFR) según tu estrategia de pruebas continuas para completar la configuración de CD de extremo a extremo del proyecto. Asegúrate de que las etapas se activan no sólo tras los cambios en el código de la aplicación , sino también tras los cambios en la configuración, la infraestructura y el código de prueba.

Las cuatro métricas clave

El resultado final de todo este esfuerzo de dedicado a establecer tus procesos de CI/CD/CT (y a cumplir los principios y la etiqueta expuestos anteriormente) es que el equipo se califique como equipo de élite o de alto rendimiento según las cuatro métricas clave (4KM) identificadas por el equipo de Investigación y Evaluación de DevOps (DORA) de Google. DORA formuló las 4KM basándose en **una amplia investigación**, y esbozó cómo utilizar estas métricas para cuantificar el nivel de rendimiento de un equipo de software como élite, alto, medio o bajo. El libro *Accelerate* de Jez Humble, Gene Kim y Nicole Forsgren es una lectura excelente para conocer los detalles de la investigación.

En resumen, las cuatro métricas clave nos permiten medir el ritmo de entrega de un equipo y la estabilidad de sus lanzamientos. Son las siguientes

Plazo de entrega

El tiempo transcurrido desde que se confirma el código hasta que está listo para su implementación en producción.

Frecuencia de Implementación

La frecuencia con la que el software se implementa en producción o en una tienda de aplicaciones.

Tiempo medio de restauración

El tiempo que se tarda en restablecer cualquier interrupción del servicio o en recuperarse de los fallos

Cambio porcentaje suspenso

El porcentaje de cambios liberados a producción que requieren una reparación posterior, como retrocesos a una versión anterior o correcciones en caliente, o que causan una degradación de la calidad del servicio.

Las dos primeras métricas, el plazo de entrega y la frecuencia de implementación, exponen el ritmo de entrega del equipo. Miden la rapidez con la que un equipo puede entregar valor a los usuarios finales y la frecuencia con la que añade valor a los usuarios finales. Sin embargo, en la prisa por ofrecer valor a los clientes, el equipo no debe comprometer la estabilidad del software. Las dos últimas métricas validan esto. El tiempo medio de restauración y el porcentaje de fallos de cambio proporcionan una indicación de la estabilidad del software que se está lanzando. En el mundo actual, los fallos de software son inevitables, y estas métricas miden lo fácil que es recuperarse de estos fallos y la frecuencia con que se producen debido a nuevas versiones. Como puedes ver, juntos, los 4KM dan una imagen clara del rendimiento de un equipo de

software, midiendo su velocidad, reactividad y capacidad de entregar con calidad y estabilidad.

Los objetivos de un equipo de élite, según la investigación DORA, están representados en [la Tabla 4-1](#).

Tabla 4-1. Las cuatro métricas clave de un equipo de élite

Métrica	Objetivo
Frecuencia de Implementación	Bajo demanda (múltiples implementaciones al día)
Plazo de entrega	Menos de un día
Tiempo medio de restauración	Menos de una hora
Cambio porcentaje suspenso	0-15%

Como ya hemos comentado, una de las principales ventajas de tener un proceso CI/CD/CT riguroso es que tu equipo podrá ofrecer valor a los clientes bajo demanda. Del mismo modo, como has visto, cuando colocas pruebas automatizadas en las capas adecuadas de la aplicación, puedes tener tu código probado como parte del proceso de pruebas continuas y fácilmente listo para su implementación en cuestión de horas (es decir, tu plazo de entrega será inferior a un día). Además, con tus pruebas de requisitos funcionales y multifuncionales automatizadas y ejecutadas como parte del proceso de TC, no debería ser ningún problema mantener tu porcentaje de cambios fallidos dentro del rango recomendado del 0-15%. Por tanto, el esfuerzo que realices en este frente permitirá a tu equipo obtener el estatus de "élite", según la definición de DORA. [La investigación de DORA](#) también demuestra que los equipos de élite contribuyen al éxito de una organización, en términos de beneficios, cotización, retención de clientes y otros criterios. Y cuando a la organización le va bien, cuida bien de sus empleados, ¿verdad?

Puntos clave

Éstos son los puntos clave de este capítulo:

- El proceso de pruebas continuas valida la calidad de la aplicación, tanto en los aspectos funcionales como en los interfuncionales, de forma automatizada para cada cambio incremental.
- Las pruebas continuas dependen en gran medida del proceso de integración continua. La integración y las pruebas continuas, a su vez, permiten la entrega continua de software a los clientes bajo demanda.
- Los procesos de integración y pruebas continuas exigen que los equipos sigan unos principios y una etiqueta estrictos para que sean fructíferos.
- Planifica tu proceso de pruebas continuas de forma que obtengas retroalimentación rápida en múltiples bucles continuamente.
- Las ventajas de las pruebas continuas son numerosas, y muchas de ellas -como el establecimiento de objetivos comunes de calidad entre funciones y equipos, la propiedad compartida de la entrega y la mejora de la colaboración entre equipos distribuidos- son difíciles de conseguir de otro modo.
- Aunque los ingenieros de DevOps puedan ser responsables de la configuración y el mantenimiento de CI/CD, es vital que los probadores del equipo diseñen la estrategia de pruebas continuas y se aseguren de que los bucles de retroalimentación se activan correctamente. Y lo que es más importante, deben vigilar de cerca las prácticas de TC del equipo para garantizar que el esfuerzo invertido en crear y mantener pruebas coseche los beneficios adecuados.

- Seguir procesos rigurosos de CI/CD/CT llevará a tu equipo a convertirse en un equipo de élite, según la definición de la investigación DORA. ¡Y un equipo de élite contribuye al éxito de toda la organización!

-
- 1 Para más información sobre éste y otros principios de la industria de CI/CD comúnmente prescritos, véase *The DevOps Handbook* (IT Revolution Press), de Gene Kim, Jez Humble, Patrick Debois y John Willis.
 - 2 Jez Humble y David Farley tratan más extensamente estas técnicas de optimización en *Entrega Continua*.
 - 3 Para más detalles, consulta el libro de Jez Humble, Gene Kim y Nicole Forsgren, *Accelerate* (IT Revolution Press).
 - 4 Para más información sobre cómo trabajar con canalizaciones, consulta [la documentación](#) de Jenkins.

Capítulo 5. Prueba de datos

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

iHaz o deshaz la confianza con los datos!

Tómate un momento para pensar en los servicios online que utilizas a diario. Verás que, en esencia, te ofrecen uno de estos dos tipos de servicios: te venden sus datos o recogen tus datos y los procesan en tu nombre. Por ejemplo, el comercio electrónico, los servicios de transporte, la entrega de comida a domicilio , la reserva/transmisión de películas y las aplicaciones de juegos en línea son ejemplos de la primera categoría, en la que su propuesta de valor principal proviene de la recopilación de datos, mientras que una aplicación de notas, las aplicaciones de redes sociales como Facebook, Twitter e Instagram, los sitios de blogs y similares prosperan acumulando tus datos. En ambos casos, los datos están en el centro de su galaxia, y sus funcionalidades únicas, el diseño de la experiencia del usuario, la marca y los aspectos de marketing giran en torno a ellos. Para profundizar en esto con un ejemplo, Amazon es un negocio de datos en su esencia. Su colección de información sobre productos constituye el epicentro del negocio, y las funcionalidades básicas, como la compra y entrega de productos, se construyen sobre ella. La marca y el marketing de la empresa llaman sutilmente la atención sobre su supremacía de datos: el logotipo de Amazon, con una flecha entre la A y la z, dice al mundo que tiene una enorme variedad de datos sobre productos que van de la a a la z.

Los datos tienen una importancia sin parangón en cualquier aplicación, y cuando su integridad no se mantiene con diligencia, la

confianza de los clientes en la aplicación puede ir rápidamente cuesta abajo, y junto con su confianza, se resentirán las ventas y el negocio. Por ejemplo, imagina que transfieres dinero entre dos de tus cuentas a través de tu aplicación de banca online y, aunque la transacción se considera correcta, los saldos de ambas cuentas no reflejan las cantidades correctas durante un periodo de tiempo. Probablemente entrarías en pánico, iy podrías empezar a cuestionar la integridad del banco! Este tipo de reacciones ocurren incluso cuando las aplicaciones no manejan ningún dato crítico. Por ejemplo, supongamos que las entradas que publicas en un sitio de blogs se muestran de forma incoherente a un conjunto de tus colegas, o que un sitio de redes sociales pierde tus fotos familiares. Estoy seguro de que todos nos sentiríamos decepcionados, ya que nuestros datos son importantes para nosotros, iindependientemente de su importancia relativa! Con el tiempo, estos fallos nos llevarán a buscar alternativas.

Lo que podemos deducir de estos ejemplos es que la integridad de los datos tiene un poder implacable de hacer o deshacer, y como resultado, probar cómo se almacenan, procesan y presentan los datos es fundamental para garantizar el éxito de la aplicación. En este capítulo trataremos los aspectos esenciales de dichas pruebas. En primer lugar, se te presentarán las distintas formas en que una aplicación almacena y procesa los datos: bases de datos, cachés, streaming y sistemas de procesamiento por lotes. La discusión en torno a estos sistemas te permitirá reconocer los nuevos casos de prueba introducidos por cada uno de ellos, y especialmente los casos de fallo provocados por problemas de concurrencia, procesamiento distribuido de datos y comunicación asíncrona. La última parte del capítulo contiene ejercicios que te capacitarán para realizar pruebas de datos automatizadas y manuales utilizando diversas herramientas.

PRUEBAS DE DATOS Y PRUEBAS FUNCIONALES

Se podría argumentar que las pruebas de datos se tratarán en como parte de las pruebas de las funcionalidades de la aplicación, y eso es parcialmente cierto. Sin embargo, cuando pienses en la misma funcionalidad desde el punto de vista del flujo de datos, descubrirás nuevos casos de prueba, que es el tema central de este capítulo.

Además, probar la funcionalidad a través de la interfaz de usuario y las API puede no ser suficiente todo el tiempo. Puede que tengas que probar la integridad de los datos en los sistemas de almacenamiento y procesamiento por separado para asegurarte de que la funcionalidad es completa. Y para ello, necesitas aprender un conjunto específico de herramientas y métodos. Además, verás en el transcurso de este capítulo que la naturaleza de estos sistemas de almacenamiento y procesamiento introduce nuevos casos de prueba, por lo que se requiere un conocimiento especializado de esos sistemas de datos. La habilidad de comprobación de datos cubre todo esto.

En otras palabras, para probar completamente una funcionalidad, necesitarás también conocimientos de prueba de datos.

Bloques de construcción

Empecemos con una introducción a un conjunto de sistemas de almacenamiento y procesamiento de datos que suelen utilizarse en aplicaciones web y móviles, para comprender los aspectos de prueba de cada uno de ellos. A efectos de este debate, volveremos a considerar la aplicación simplificada de comercio electrónico utilizada en [el Capítulo 3](#). La Figura 5-1 muestra la misma arquitectura de aplicación, esta vez resaltando los sistemas de datos.

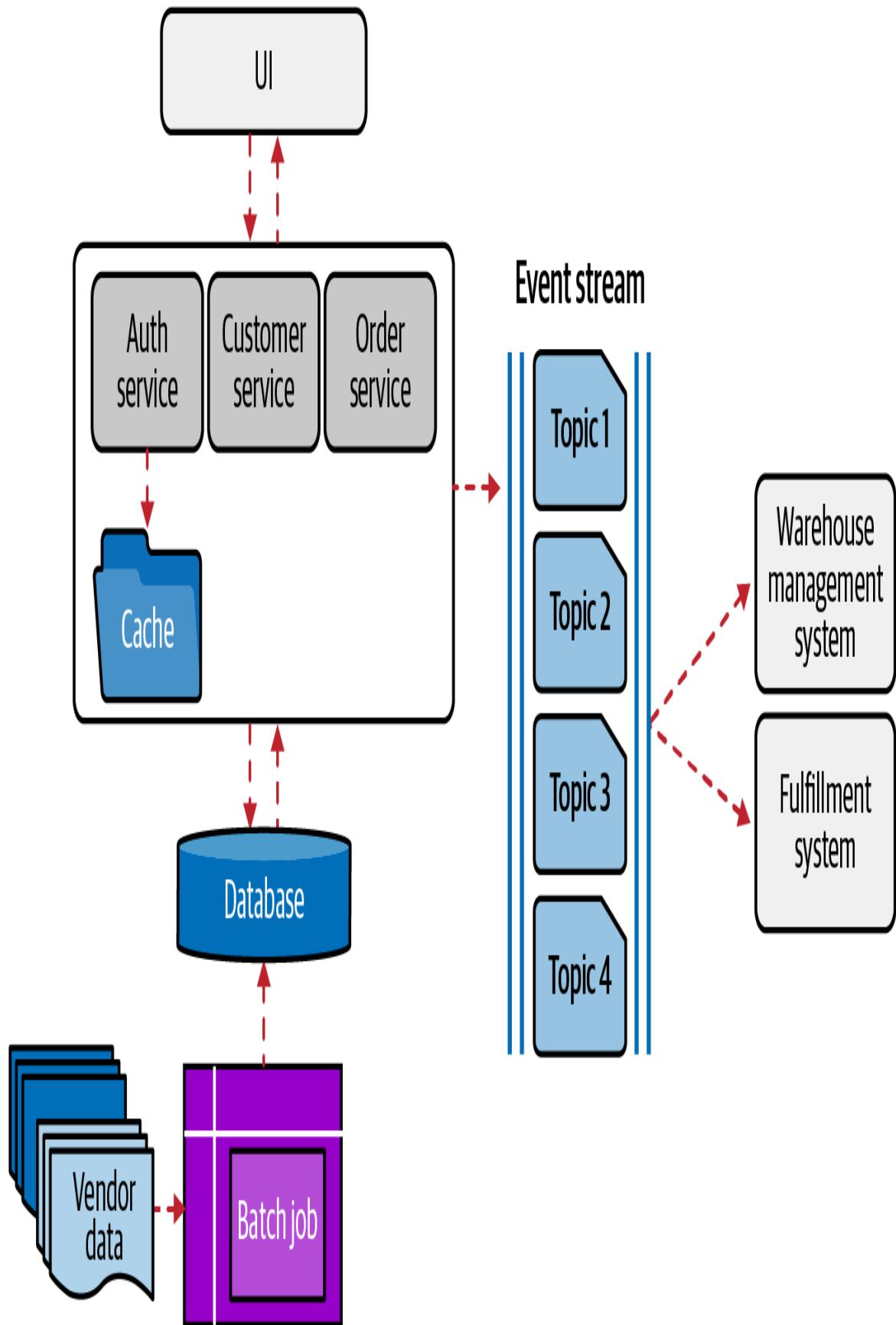


Figura 5-1. Una aplicación sencilla de comercio electrónico con cuatro sistemas de almacenamiento y procesamiento de datos

Como recordarás, la aplicación tiene una capa de interfaz de usuario, que se comunica con un conjunto de servicios para distintos tipos de procesamiento empresarial. Los servicios, a su vez, están conectados a una base de datos centralizada , donde se almacenan todos los datos de la aplicación. Además, verás otros tres sistemas de datos: un servidor de caché, un sistema de procesamiento por lotes y un flujo de eventos. Las flechas de la figura indican el flujo de datos entre estos sistemas. Vamos a trazar el flujo de datos empezando por la capa de interfaz de usuario para aclarar las distintas funciones que desempeñan estos sistemas de datos.

Para empezar, supongamos que un usuario final intenta iniciar sesión en la aplicación introduciendo sus credenciales en la interfaz de usuario. La capa UI comienza pasando los datos de las credenciales al servicio de autenticación (auth). Este servicio, a su vez, pasa los datos a la base de datos para comprobar si el registro coincide. Si las credenciales coinciden, el servicio de autenticación, suponiendo que siga el [protocolo OAuth 2.0](#), devuelve un token de acceso a la IU y también lo persiste en el servidor de caché. Se trata de un dato interno clave en la aplicación, ya que cualquier acción posterior realizada por el usuario requerirá este token de acceso para demostrar que un usuario final válido está solicitando la acción.

Vayamos un paso más allá y veamos un ejemplo. Supongamos que un usuario intenta hacer un pedido desde la interfaz de usuario. La capa de la interfaz de usuario construye una solicitud de pedido para enviarla al servicio de pedidos y añade el código de acceso en la cabecera de la solicitud. El servicio de pedidos comprueba con el servicio de autenticación para asegurarse de que el código de acceso es válido, y el servicio de autenticación, a su vez, consulta la caché. Si el token de acceso ya ha caducado, el servidor de caché lo habrá borrado automáticamente, y por tanto los servicios devolverán un código de estado 404. Al recibir esta respuesta 404, la IU redirigirá

diligentemente al usuario de vuelta a la página de acceso, en un esfuerzo por proteger su seguridad.

Por otra parte, cuando el token de acceso es válido (es decir, no ha caducado en la caché), el servicio de autenticación acusa recibo al servicio de pedidos, que permitirá que se cree el pedido en la base de datos. El servicio de pedidos también creará un evento con la información del nuevo pedido y lo colocará en el sistema de transmisión de eventos para que los sistemas posteriores, como los sistemas de gestión de almacenes y de gestión de cumplimiento, realicen sus respectivos trabajos. Un punto a tener en cuenta aquí es que la responsabilidad del servicio de pedidos termina con la colocación del evento en el sistema de transmisión de eventos, y realmente no le importa si los respectivos sistemas posteriores hacen su trabajo correctamente. Los sistemas descendentes escuchan continuamente al sistema de transmisión de eventos y sólo consumen aquellos eventos que son relevantes para ellos. Por ejemplo, si hay un evento de cambio de dirección de un cliente, el sistema de gestión del cumplimiento puede estar interesado, pero el sistema de gestión del almacén puede no procesarlo; sin embargo, si hay un evento de creación de un pedido, ambos sistemas pueden procesarlo.

Mientras tanto, existe un sistema de procesamiento por lotes para analizar los detalles de los productos de los distintos vendedores en la base de datos centralizada. Este procesador por lotes se activa automáticamente a la hora programada -digamos, una vez al día a medianoche- e importa todos los datos nuevos y actualizados a la base de datos, de modo que los nuevos productos y cualquier otro cambio sean visibles en la aplicación al día siguiente.

Como podemos ver en este ejemplo, cada uno de estos cuatro sistemas de datos desempeña un papel fundamental a la hora de satisfacer una serie de requisitos esenciales de la aplicación. Explorémoslos individualmente con más detalle, para descubrir las

propiedades únicas que los hacen adecuados para sus respectivos contextos y los nuevos casos de prueba que exponen.

Bases de datos

Las bases de datos no necesitan realmente una introducción, ya que son un componente prevalente en casi todas las aplicaciones y son, comparativamente hablando, un sistema de almacenamiento de datos establecido. Una de las razones de esta amplia adopción es su capacidad para proporcionar una gran durabilidad de los datos, ya que éstos se almacenan en un disco duro y sólo se pierden en caso de fallo del hardware.

Para los que no conocen las bases de datos, pueden compararse con los joyeros, en los que organizas los distintos accesorios en sus respectivos compartimentos y puedes acceder a ellos siempre que los necesites. Cada pieza de joyería se guarda a salvo en el joyero hasta que alguien la transfiere o la sustituye. Del mismo modo, los datos de una aplicación se organizan y almacenan de forma significativa en bases de datos y se pueden consultar siempre que sea necesario. La aplicación tiene la flexibilidad de *crear* nuevos datos y de *leer, actualizar* o *eliminar* los existentes según lo requiera la funcionalidad (estas cuatro operaciones básicas se denominan comúnmente en con el acrónimo CRUD).

Según la forma en que estén estructurados los datos, como en tablas, como documentos JSON o XML, o como gráficos, las bases de datos pueden clasificarse en bases de datos relacionales, de documentos y de gráficos, respectivamente.¹ Las bases de datos relacionales son la categoría dominante, y han servido a un amplio abanico de necesidades de almacenamiento de datos de las aplicaciones durante las últimas décadas. MySQL y PostgreSQL son ejemplos de bases de datos relacionales de código abierto; veremos cómo trabajar con una BD PostgreSQL en los ejercicios de este capítulo .

En una base de datos relacional, los datos se almacenan en estructuras de tablas, con filas y columnas. Cada fila de una tabla representa un conjunto de información relacionada separada en columnas, como en la tabla `Customers` de la [Tabla 5-1](#).

Tabla 5-1. Ejemplo de estructura de una tabla en una base de datos relacional

UUID (clave primaria)	Nombre del cliente (varchar 30)	Número de teléfono (int)	Dirección de correo electrónico (varchar 254)	Dirección envío (varchar 100)
019367	Alice	4567879	alice@xyz.com	13/8, Blc
045678	Bob D'arcy	0898678	bobdarcy@xyz.com	23-A, Pla Winscent

Las columnas pueden tener nombres predefinidos y propiedades, como su tipo de datos y longitud máxima. Además, cada fila recibe un identificador único universal (UUID) que sirve para relacionar los registros entre varias tablas. Por ejemplo, la lista de clientes de nuestra aplicación de comercio electrónico podría almacenarse en una tabla como ésta, en la que cada fila representa los datos de un único cliente, como su nombre, dirección de correo electrónico, número de teléfono y dirección de envío. En este caso, se creará un ID de usuario único como identificador único del registro y se almacenará junto con él, que luego podrá utilizarse para consultar la información del cliente. El mismo ID de usuario puede utilizarse en otras tablas, como la tabla de historial de cuentas, lo que permite recuperar un catálogo completo de información sobre un usuario determinado. Esta definición de tablas, filas, nombres de columnas, identificadores únicos, etc. se denomina *esquema de la base de datos*. El esquema de la base de datos se obtiene a partir del caso de uso empresarial de la aplicación, por los desarrolladores o un administrador de la base de datos. El esquema también puede

redefinirse en el transcurso de la entrega a medida que crecen los requisitos de la aplicación. Para realizar estas operaciones, en las bases de datos relacionales se utiliza un lenguaje específico del dominio llamado Lenguaje de Consulta Estructurado (SQL, pronunciado *see-quel*).

Dada esta información, algunos de los casos de prueba básicos que podría querer probar son:

- Verificar el caso de prueba positivo en el que la información obtenida del usuario a través de la interfaz de usuario debe almacenarse en la BD y relacionarse adecuadamente.
- Comprobación de los valores límite en función del tipo de datos de la columna y de la longitud de las entradas. Por ejemplo, cuando el campo nombre de cliente está restringido a menos de 20 caracteres en la BD, debe aplicarse la misma restricción en la IU, y debe mostrarse al usuario un mensaje de error adecuado si se supera la longitud.
- Pruebas con entradas que incluyan sintaxis SQL. Por ejemplo, ¿puede Bob D'arcy, cuyo nombre incluye un apóstrofo, almacenar su nombre correctamente en la BD? ¿O es necesaria alguna lógica de limpieza en el código?
- ¿Qué ocurre con una operación de escritura en curso si se produce un fallo repentino de la red? ¿Los datos se escriben parcialmente en algunas tablas, pero no en todas las tablas relacionadas? Este escenario se amplifica cuando la operación se divide entre varios servicios.
- ¿Cómo afecta a estos casos una operación de reintento?
- ¿Cuál es el tiempo de espera antes de que la aplicación reintente una operación de base de datos, y cómo será la experiencia del usuario?

Cuando incluimos el factor concurrencia -es decir, cuando varios usuarios y sistemas acceden simultáneamente a la base de datos para realizar lecturas y escrituras-, debemos pensar en algunos casos de prueba más, especialmente en torno a las condiciones de carrera. He aquí algunas consideraciones para desencadenar el proceso de reflexión:

- Es posible que las acciones de un usuario choquen con las de otro, dando lugar a actualizaciones perdidas. Por ejemplo, cuando dos usuarios compran el mismo artículo en el mismo instante, la cantidad del artículo puede disminuir sólo en una cuenta, y no en dos.
- Del mismo modo, los usuarios podrían ver datos no coincidentes si la aplicación opta por leer actualizaciones parciales. Por ejemplo, supongamos que se están reponiendo las existencias de algunos artículos no disponibles, y el proceso cambia primero el indicador de disponibilidad del artículo a verdadero en una tabla y luego actualiza el recuento de artículos disponibles en otra tabla. Entre estas dos operaciones, un usuario final podría ver que el artículo está disponible pero con una cantidad 0.
- De nuevo, la concurrencia podría tener un efecto sin precedentes sobre los recursos compartidos. Por ejemplo, si dos usuarios compran simultáneamente el último artículo disponible utilizando la opción de pago contra reembolso, es posible que el artículo se asigne a un usuario, pero la factura se genere para el otro.
- Además, la concurrencia impone un límite al rendimiento de la base de datos. Por tanto, las pruebas de rendimiento con volúmenes de datos previstos en tiempo real se convertirán en un caso de prueba crucial.

Una llamada de atención aquí es que los casos de prueba relacionados con la concurrencia son difíciles de simular, y

conocerlos es útil sobre todo en la fase de análisis para poder abordarlos preventivamente durante el desarrollo.

Más allá de los accesos concurrentes de una única instancia, las bases de datos atienden a la escalabilidad mediante *la replicación*. La replicación se refiere a la creación de instancias redundantes de los mismos datos. Las instancias suelen mantenerse separadas geográficamente, para mejorar el rendimiento de los usuarios de distintas ubicaciones (por ejemplo, la costa este y oeste de EE.UU., o Norteamérica y Europa). En estos casos, debe existir un mecanismo para mantener todas las réplicas sincronizadas con las últimas actualizaciones. Esto suele conseguirse asignando a una de las réplicas el papel de *líder*, responsable de enviar las actualizaciones a las demás réplicas (las *seguidoras*). Esta situación puede dar lugar a un retraso en la replicación, en el que los seguidores tardan algún tiempo en recibir la actualización y alcanzar el mismo estado que el líder. El retraso podría ser del orden de unos segundos a unos minutos, en función de la latencia de la red, el tráfico a esa instancia, etc. Este modelo de alcanzar un estado consistente tras un periodo de tiempo se denomina *consistencia eventual*.

NOTA

Para saber más sobre otros modelos de coherencia, consulta la [guía de Jepsen](#), con un mapa en el que puedes hacer clic.

El modelo de consistencia eventual es adecuado para aplicaciones de como Twitter o Facebook, donde mostrar publicaciones ligeramente más antiguas puede no tener un gran impacto en los usuarios. Sin embargo, en algunas otras aplicaciones, el desfase podría confundir a los usuarios si no se gestiona adecuadamente, e incluso dañar la confianza. Analicemos algunos de los posibles problemas que pueden surgir:

Leer tus propios escritos

Supongamos que un usuario actualiza la información de su perfil, luego quiere confirmar los cambios y vuelve a abrir la página del perfil unos segundos después. Es posible que las actualizaciones aún no se hayan propagado a todos los seguidores, por lo que si la aplicación lee a un seguidor rezagado, éste puede acabar viendo su antigua información de perfil. Confundidos por esto, pueden volver a introducir los cambios. Si este ciclo se repite varias veces, además de frustrarse el usuario, el sistema podría sobrecargarse, prolongando el retraso en la replicación.

Viajar en el tiempo

Supongamos que un usuario está siguiendo las actualizaciones de cricket en directo de en un sitio web de deportes. Actualizan la página cada pocos segundos para ver los resultados. Si el sitio web lee de varios seguidores con una coherencia eventual, podrían experimentar una sensación de viaje en el tiempo. Por ejemplo, la primera actualización podría mostrar el resultado como 116 carreras en 5 overs, mientras que en la siguiente actualización el sitio web podría leer de un seguidor rezagado y mostrar el resultado como 110 en 4,5 overs.

Ordenación incoherente

A veces los datos se encadenan secuencialmente, y no tiene sentido si la secuencia no se mantiene. Por ejemplo, en una conversación sobre un post de Facebook es necesario mantener el orden en que los usuarios escriben sus comentarios, pero cuando no se piensa en los retrasos de replicación y se gestionan adecuadamente, es posible que un usuario vea un comentario que responde a una pregunta sin ver la pregunta anterior.

Escribir conflictos

Para evitar puntos únicos de fallo , a veces se asigna más de un líder para gestionar la replicación. En estos casos, las nuevas actualizaciones podrían enviarse a distintos líderes, lo que provocaría conflictos de escritura. Los conflictos de escritura se producen siempre que un mismo recurso es alterado por muchas partes, como una diapositiva de Google que es editada al mismo tiempo por distintos miembros del equipo. En tal caso, las ediciones del mismo texto podrían ser aceptadas por diferentes líderes, pero cuando se combinen las actualizaciones habrá un conflicto sobre cuál de ellas se tomará como actualización final.

La buena noticia es que las soluciones a estos problemas comunes están bien establecidas, y la mayoría de las veces las propias bases de datos los gestionan de forma inherente. Sin embargo, es esencial ser consciente y estar atento a estos posibles problemas, tanto en el desarrollo de aplicaciones como en las pruebas.

Para resumir los puntos clave de esta sección, cuando pruebes bases de datos ten en cuenta los datos de tu aplicación y sus variaciones, así como problemas potenciales como fallos de red, choques de concurrencia y otros retos de datos distribuidos.

Cachés

Una *caché* es un almacén de datos en memoria en el que los datos persisten como pares clave/valor. Almacenar los datos en memoria aumenta el rendimiento en varios órdenes de magnitud, ya que la aplicación no tiene que hacer llamadas a un sistema de almacenamiento backend pesado, como una base de datos relacional tradicional. Las herramientas de almacenamiento en caché más populares hoy en día, como **Memcached** y **Redis**, pueden almacenar terabytes de datos y proporcionar respuestas por debajo del milisegundo. Sin embargo, cuando se trata de durabilidad, las bases de datos tienen ventaja, ya que los datos se escriben firmemente en el disco.

NOTA

Redis ha evolucionado para ofrecer muchas funciones, además de ser un sistema de almacenamiento en caché en memoria. Incluso puede configurarse para persistir datos puntuales (instantáneas) en el disco para su recuperación. Para saber más sobre las características de Redis, consulta la [documentación oficial](#).

Teniendo en cuenta estos pros y contras de las cachés, la práctica recomendada suele ser almacenar en caché sólo los datos de naturaleza transitoria y que la aplicación necesita con frecuencia. Por ejemplo, en nuestra aplicación de comercio electrónico, los tokens de acceso se almacenan en caché porque se espera que vivan sólo durante un breve periodo de tiempo (hasta que el usuario inicie sesión) y la aplicación debe acceder a ellos con frecuencia internamente, para validar la autenticidad de cada solicitud de servicio. Además, en una situación desafortunada, como un fallo de la caché, si se pierden todos los tokens de acceso de los usuarios, el impacto es mínimo: para recuperarse, el conjunto actual de usuarios conectados simplemente tendrá que desconectarse y volver a conectarse (una molestia menor, no equiparable a la pérdida de valiosos datos personales o del historial del cliente). Una caché encaja perfectamente en este escenario, ya que no hay una demanda de durabilidad sólida como en una base de datos.

Un enfoque alternativo y común es replicar los datos de la aplicación a los que se accede con frecuencia tanto en la caché como en una base de datos. En estos casos, el código de la aplicación tiene que asumir la responsabilidad de mantener el ciclo de vida de los datos almacenados en caché: mantenerlos sincronizados con la base de datos, purgar los datos antiguos, volver a la base de datos en caso de fallo de la caché, etc. Estos escenarios se convertirán en casos de prueba cuando los datos se repliquen en la base de datos y en la caché. Otros casos de prueba, en general, serían:

- Los datos de la caché se configurarán con un valor de tiempo de vida (TTL) a partir del cual caducarán. Por ejemplo, los tokens de acceso podrían configurarse para vivir 30 segundos. Más allá de 30 segundos, debemos asegurarnos de que el servicio de autenticación genera un nuevo token y lo almacena de nuevo en la caché.
- Si la caché se convierte en un único punto de fallo para la aplicación, como en el caso de un fallo de la caché que provoque que todos los usuarios tengan que desconectarse y volver a conectarse, hay que probar el flujo de redirección de usuarios.
- Si las instancias de servicio están replicadas, sus cachés también lo estarán, lo que dará lugar a un almacenamiento distribuido de cachés. Es posible que tengas que asegurarte de que la funcionalidad sigue funcionando correctamente. (La mayoría de las implementaciones de caché distribuida, como Redis Cluster, admiten la redirección a la instancia de caché correcta de forma inmediata, por lo que se trata de una cuestión de verificar el flujo funcional).
- Probar el rendimiento de la aplicación con la carga máxima de volverá a ser fundamental.

Sistemas de procesamiento por lotes

Un *sistema de procesamiento por lotes* es aquel en el que se escribe un programa o trabajo para transformar un conjunto de datos de entrada en la salida deseada, no en tiempo real, sino en lotes reunidos durante un período de tiempo. Estos trabajos por lotes pueden escribirse utilizando marcos y bibliotecas como Spring Batch o Apache Spark y pueden ejecutarse de forma autónoma sin intervención del usuario. Los datos de entrada a los trabajos por lotes pueden estar en forma de archivos, registros de bases de datos, imágenes, etc. El volumen de datos de entrada puede ser

masivo, haciendo que el trabajo tarde horas o incluso días en completarse. De hecho, el rendimiento de un trabajo por lotes se mide en función del tamaño del archivo de entrada que puede procesar y en qué tiempo, no en función de los tiempos de respuesta, como ocurre con las bases de datos o las cachés.

Algunos casos de uso típicos de un sistema de procesamiento por lotes son la generación de informes, la generación de facturas, la generación de nóminas mensuales y la limpieza de datos antes de entrenar modelos de aprendizaje automático. Un par de patrones observables en estos casos de uso son que los trabajos por lotes transforman datos desorganizados o dispersos en estructuras de datos significativas, y no es necesario que tales transformaciones se produzcan en tiempo real para que la aplicación funcione sin problemas.

Para ilustrarlo, volvamos a la aplicación de comercio electrónico. Cuando los vendedores quieren mostrar sus catálogos de productos nuevos o actualizados en la aplicación, envían sus últimos detalles de artículos como archivos. Los archivos pueden tener miles de registros de artículos, con cada artículo representado por su SKU, color, talla, precio, etc. Las claves de los registros de artículos pueden ser diferentes según el vendedor, y los archivos pueden estar en formatos diferentes, como JSON o CSV, dependiendo de los sistemas internos del vendedor. Estos datos desorganizados, con todas sus variaciones, deben transformarse en una estructura común para que tengan sentido para la aplicación, es decir, los archivos deben transformarse en registros de base de datos para que la aplicación pueda mostrarlos en la interfaz de usuario. Un punto a tener en cuenta aquí es que basta con reflejar el catálogo actualizado al día siguiente o unos días después, y no necesariamente en tiempo real. Por tanto, un sistema de procesamiento por lotes encaja perfectamente.

Se puede escribir un trabajo por lotes (o varios trabajos por lotes) para que lea los registros uno a uno desde distintos archivos,

extraiga la información adecuada y la transforme en registros de la base de datos. El trabajo puede programarse para que se ejecute de forma autónoma todos los días a la misma hora; por ejemplo, a medianoche, cuando el tráfico del sitio es bajo. En tal caso, los archivos de proveedores enviados ese día antes de medianoche forman un lote. Normalmente, en caso de fallo, los trabajos por lotes se vuelven a ejecutar, con una disposición para descartar o sobrescribir los datos creados durante la ejecución anterior fallida.

Dada la naturaleza de los sistemas de procesamiento por lotes, algunos de los casos de prueba generales en los que hay que pensar al probarlos son:

- Verificar que los archivos de entrada se procesan en su totalidad y que la operación no se abandona a mitad de camino.
- Manejar entradas que están corruptas, como tener valores nulos inesperados, números enteros grandes y otras anomalías
- Marcar y aislar los registros incompletos que no pueden transformarse en la estructura requerida
- Garantizar que los mecanismos de reintento limpian o sobrescriben los datos de la ejecución fallida
- Verificar que los trabajos por lotes, que pueden ocupar una capacidad de procesamiento importante, no afectan negativamente al rendimiento de la aplicación.

A veces, durante las pruebas, puedes descubrir que diferentes partes envían datos en nuevos formatos, lo que puede obligar a tu equipo a actualizar el trabajo por lotes. Además, los distintos proveedores pueden tener recuentos diferentes de productos en determinadas categorías, como ropa de hombre, calzado deportivo, etc. Si la cantidad de productos en una categoría concreta es demasiado alta, lo que también se conoce como *desviación de datos*, el rendimiento del trabajo por lotes puede verse afectado,

dependiendo de cómo esté programado. Por eso, obtener varias muestras de entradas por adelantado de las partes pertinentes ayudará en las pruebas.

Flujos de eventos

Un *evento*, en su sentido literal, se refiere a una acción, mientras que un *flujo* representa una entidad que fluye o, en otras palabras, de naturaleza continua. Así pues, *los flujos de eventos* son sistemas en los que los eventos específicos de una aplicación se publican continuamente en un flujo, del que otros sistemas relevantes consumen a su vez esos datos cada vez que tienen tiempo para procesarlos. Por ejemplo, en la aplicación de comercio electrónico, como se ve en [la Figura 5-2](#), un evento de pedido con los detalles del pedido se publica en el flujo de eventos inmediatamente cuando un cliente hace un pedido, y los sistemas posteriores leen el evento y toman las acciones respectivas para satisfacer el pedido. Desde la perspectiva del flujo de datos, los datos del pedido se almacenan en el flujo de eventos durante un periodo de tiempo, y los sistemas previstos leen de él hasta entonces.

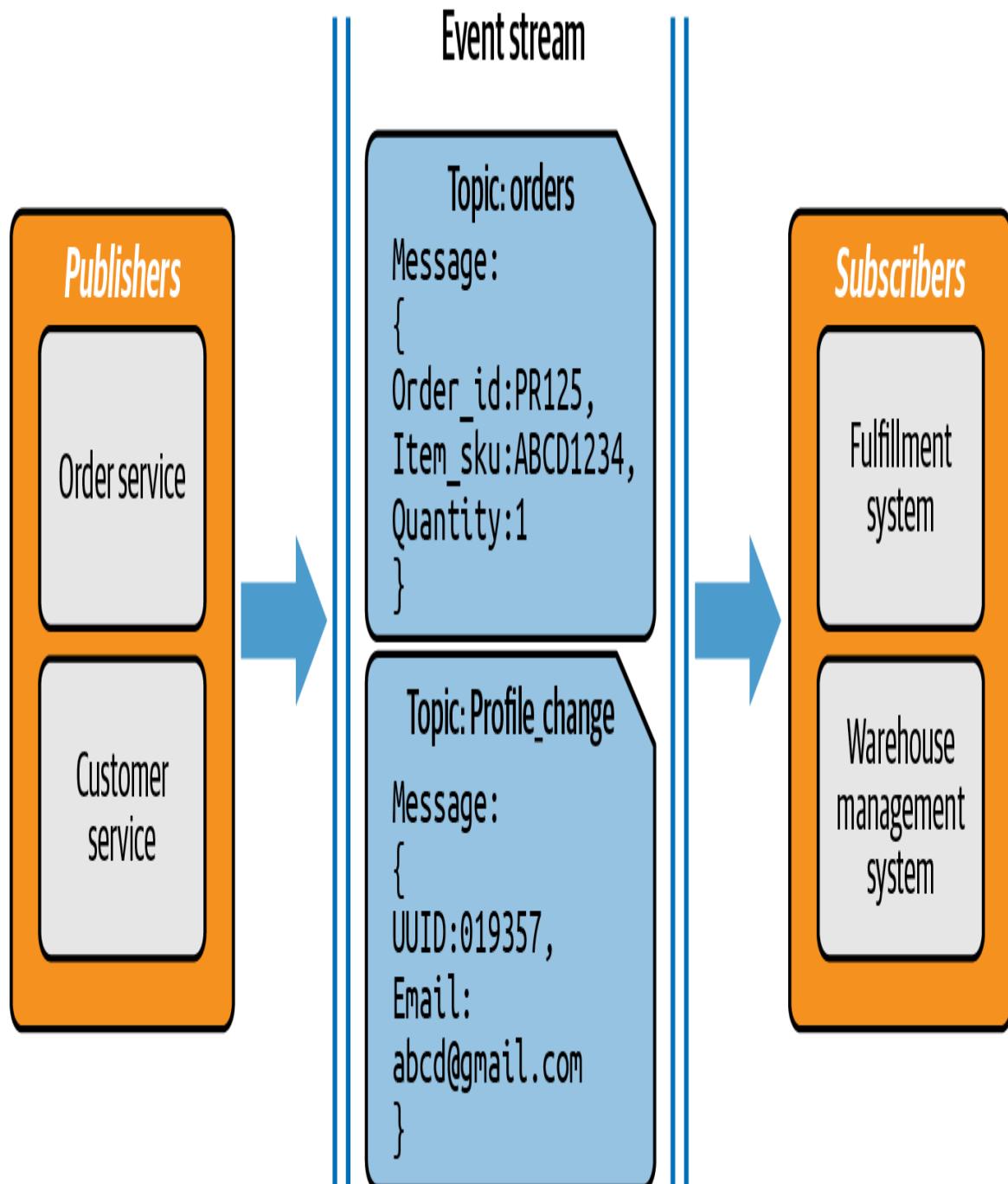


Figura 5-2. Un sistema de flujo de eventos

Aquí, el servicio de pedidos es llamado *editor*, ya que publica los eventos, y los sistemas posteriores que consumen los eventos se llaman *suscriptores*. Cada evento se publica con un nombre de *tema* específico para que los suscriptores puedan identificar los eventos que son relevantes para ellos. Algunos sistemas de flujo de eventos,

como **Google Cloud Pub/Sub** y **RabbitMQ**, eliminan cada evento después de que todos sus suscriptores previstos lo hayan consumido. En otros sistemas, como **Apache Kafka**, los eventos se borran después de un tiempo configurado. Esta característica de retención permite a los suscriptores ponerse al día tras fallos provisionales . Los flujos de eventos también proporcionan durabilidad, ya que escriben los eventos en el disco, como las bases de datos.

Dadas las características de un sistema de flujo de eventos, es justo preguntarse si un trabajo por lotes no encajaría en su lugar. Un sistema de procesamiento por lotes difiere de un sistema de procesamiento de flujo de eventos principalmente en su naturaleza limitada en el tiempo; es decir, un trabajo por lotes procesa las entradas en o después de un tiempo preconfigurado, mientras que con un flujo de eventos el procesamiento ocurre casi en tiempo real. Para profundizar en este término, el servicio de pedidos de la aplicación de comercio electrónico publica un evento inmediatamente después de crear el pedido, pero no requiere acuse de recibo de los sistemas posteriores, es decir, es asíncrono. Por tanto, colocar el pedido en el flujo de eventos permite que el procesamiento del pedido no se retrase varias horas o más, como ocurriría con el procesamiento por lotes, pero el pedido tampoco se procesa sincrónicamente como una solicitud de servicio web. Como resultado, esto se denomina *procesamiento casi en tiempo real* y no *en tiempo real*, aunque los eventos puedan ser consumidos en pocos segundos por sus suscriptores. Este modelo asíncrono sirve bien para el procesamiento paralelo y el escalado. Actualmente se utiliza mucho en el desarrollo de aplicaciones web y móviles.

Algunos de los casos de prueba en los que pensar en un sistema de transmisión de eventos son:

- La estructura de eventos es un acuerdo entre el editor y el suscriptor, por lo que cada vez que se produce un cambio en la estructura, hay que volver a probar todo el flujo funcional.

- A veces, hay que probar la compatibilidad con las estructuras de eventos antiguas y nuevas.
- Puede existir el requisito de procesar los eventos en una secuencia específica. Por ejemplo, el envío de un artículo no puede procesarse hasta que el almacén confirme su disponibilidad. Como el procesamiento de los eventos se produce de forma asíncrona, hay que probar el flujo.
- Un abonado, en caso de fallo, debe ser capaz de ponerse al día con los nuevos acontecimientos en el orden correcto.
- Si incluso después de varios reintentos se producen errores en el procesamiento de un evento, éste se traslada a una cola separada llamada *cola de letra muerta*, a la que se añaden los detalles del error para ayudar en la depuración. Este flujo de eventos a la cola de letra muerta debe probarse.
- ¿Qué ocurre cuando falla el flujo de eventos? ¿Cómo gestionan el fallo los editores y suscriptores? ¿Cuándo y cómo vuelven a intentarlo?
- El rendimiento de los suscriptores podría ser más lento que el del editor, provocando hinchazón en el flujo. Por tanto, hay que comprobar su capacidad para consumir eventos a tiempo.

Como puedes ver, los distintos sistemas de almacenamiento y procesamiento de datos desempeñan cada uno un papel único en el ecosistema más amplio de la aplicación y, por tanto, exigen una atención específica a la hora de realizar las pruebas. Esto nos lleva a la siguiente sección, en la que veremos un enfoque para probar los cuatro sistemas de datos comentados anteriormente.

Estrategia de comprobación de datos

En su libro *Designing Data-Intensive Applications*, Martin Kleppmann escribe:

Sería imprudente suponer que los fallos son poco frecuentes y limitarse a esperar lo mejor. Es importante considerar una amplia gama de posibles fallos -incluso bastante improbables- y crear artificialmente esas situaciones en tu entorno de pruebas para ver qué ocurre.

No podría estar más de acuerdo con él en eso, especialmente cuando se trata de pruebas de datos. El 90% de las pruebas de datos implican pensar en posibles fallos, a diferencia de las pruebas funcionales, en las que el proceso de pensamiento gira en torno a las probables acciones del usuario en la aplicación. Esto habrá quedado patente en la sección anterior, en la que nos hemos centrado en los casos de prueba causantes de fallos.

Con esa mentalidad en su núcleo, una estrategia de comprobación de datos típica de puede visualizarse comprendiendo cuatro ramas, como se muestra en la Figura 5-3.

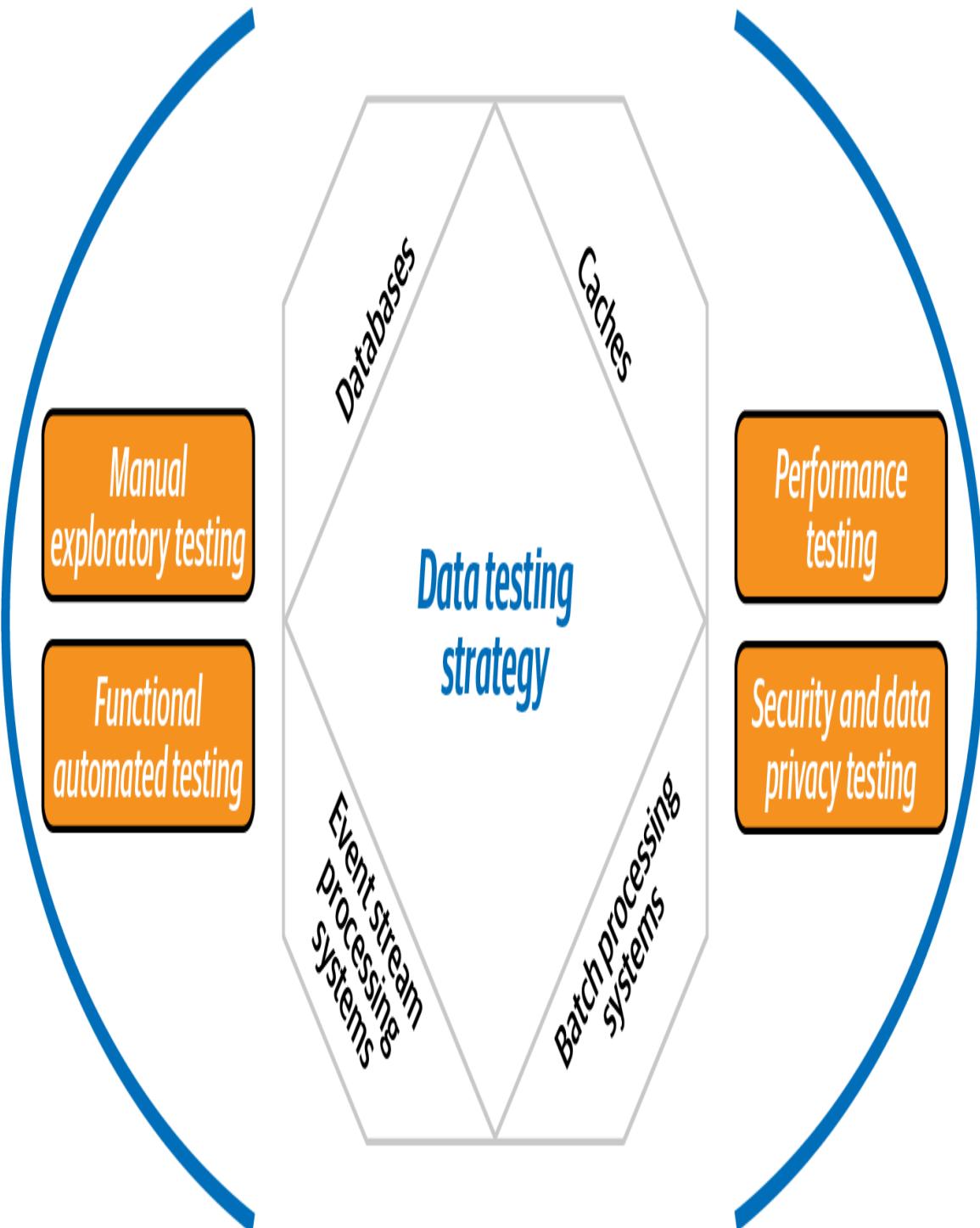


Figura 5-3. Una estrategia de comprobación de datos

Las ramas son:

Pruebas exploratorias manuales

Las pruebas exploratorias manuales pueden hacer que descubra muchos casos de prueba que provocan fallos y son muy importantes en las pruebas de datos. En [el Capítulo 2](#) aprendiste sobre las técnicas de muestreo, que son esenciales para las pruebas de datos y pueden aplicarse principalmente cuando se trata de bases de datos y sistemas de procesamiento por lotes. Además, conocer las propiedades específicas de las herramientas de procesamiento de datos que utilizas (como Apache Kafka, Redis, etc.) te ayudará a identificar los aspectos particulares de cada herramienta que merece la pena explorar manualmente.

Puede que también necesites aprender herramientas adicionales para la exploración manual. Por ejemplo, SQL es una herramienta esencial para las pruebas exploratorias de bases de datos relacionales; en los ejercicios de este capítulo se incluye una introducción a su uso.

Pruebas funcionales automatizadas

Para obtener información rápida sobre los casos de prueba de relacionados con los datos, tenemos que automatizarlos e integrarlos con CI. Empezar con pruebas unitarias o de integración es el enfoque recomendado para los cuatro sistemas de datos tratados en este capítulo. En la siguiente sección se comentan algunas herramientas relevantes.

Pruebas de rendimiento

Como hemos visto, los sistemas de almacenamiento de datos y procesamiento son componentes críticos en cualquier aplicación, y por tanto, su rendimiento afecta en gran medida al rendimiento general de la aplicación. Así que es importante realizar pruebas de carga y estrés en todos los sistemas de almacenamiento y procesamiento de datos de la aplicación. Las pruebas de rendimiento del backend se tratan en detalle en [el Capítulo 8](#).

Seguridad y privacidad

Las violaciones de datos causan enormes pérdidas a los clientes y dan lugar a fuertes sanciones para las empresas. La comprobación de la seguridad es uno de los aspectos más críticos de la comprobación de datos, y se tratará en detalle en el [Capítulo 7](#). Además, existen leyes de protección de datos específicas de cada país que imponen la privacidad de los datos por diseño. Estas normativas y la comprobación de su cumplimiento se tratarán en el [Capítulo 10](#).

Para resumir la estrategia de prueba de datos, recordemos rápidamente algunos puntos clave anteriores: al probar cada una de estas ramas, ten en cuenta los tipos y variaciones de datos, la concurrencia, la naturaleza distribuida de los datos y los sistemas, y la posibilidad de fallos en la red. Ten en cuenta también que algunos casos de prueba relacionados con los datos, aunque se prueben, pueden no revelar los fallos subyacentes, ya que dependen en gran medida de la sincronización de las acciones (por ejemplo, los casos de prueba relacionados con la concurrencia). Recuerda analizar esos casos de prueba durante la propia fase de análisis. A continuación, nos pondremos manos a la obra con algunos ejercicios.

Ejercicios

Los ejercicios que aquí se presentan son algunas herramientas esenciales para las pruebas de bases de datos, como SQL y JDBC. También exploraremos Apache Kafka y Zerocode, una herramienta para escribir pruebas automatizadas de verificación de mensajes Kafka.

NOTA

Como ya se ha mencionado, lo ideal es que la mayoría de los casos de prueba relacionados con los datos se automaticen como parte de las pruebas de integración de unidades durante el desarrollo. Aquí hablaremos de las herramientas necesarias para los probadores, tanto en las pruebas exploratorias manuales como en las pruebas automatizadas funcionales a nivel macro.

SQL

El conocimiento de SQL es algo sin lo que no puedes vivir cuando pruebas las funcionalidades de una aplicación que incluye un componente de base de datos. Inevitablemente te encontrarás con situaciones en las que tendrás que consultar la base de datos y asegurarte de que los datos están intactos. Cuando se trata de bases de datos que tienen muchas tablas, columnas y filas, tener los conocimientos de SQL necesarios para filtrar rápidamente los datos y ver lo que se necesita para el caso de prueba te ahorrará un montón de frustraciones. Así que, si no estás familiarizado con las distintas facetas del lenguaje SQL, como ordenar, filtrar, agrupar, anidar, unir, etc., ¡prueba este ejercicio!

Para este ejercicio, necesitas una base de datos relacional. Si tienes una preparada, ¡genial! Si no, sigue los pasos de la sección de requisitos previos para configurar una.

Requisitos previos

Configura una base de datos PostgreSQL en tu máquina local descargando el paquete o instalador correspondiente del [sitio web oficial](#). Una vez instalado, [inicia el servidor Postgres](#) utilizando los comandos específicos de tu sistema operativo. Por ejemplo, si eres usuario de Mac, abre el Terminal y ejecuta los siguientes comandos:

1. Descargar PostgreSQL con `brew install postgresql`.

2. Inicia el servidor postgres con `brew services start postgresql`.
3. Abre el cliente shell, `psql`, utilizando el comando `psql postgres`. El cliente `psql` se conectará al servidor de la base de datos y ejecutará en él las consultas SQL. También puedes utilizar un cliente GUI como `pgAdmin`.

NOTA

Cuando hayas completado los ejercicios, no olvides **detener el servidor de bases de datos**, por ejemplo utilizando el comando `brew services stop postgresql`.

Flujo de trabajo

Como ya se ha mencionado, el lenguaje SQL se utiliza para operar sobre bases de datos relacionales (leer, escribir, actualizar y eliminar datos en ellas). El lenguaje se compone de varias palabras clave y funciones que hacen que ordenar, filtrar y unir datos de varias tablas sea bastante sencillo. Aquí te mostraré el conjunto de consultas que se necesitan con más frecuencia para la comprobación manual de bases de datos.

Crea

En primer lugar, crea una nueva tabla llamada `items` que almacene los detalles del artículo, como SKU, color, talla y precio. Para ello, ejecuta la siguiente consulta desde tu cliente `psql`:

```
postgres=> create table items (item_sku varchar(10), color  
varchar(3), size varchar(3), price int);
```

Esta consulta utiliza las palabras clave SQL `create table` para especificar el tipo de operación a realizar, y da un nombre a la tabla.

Además, elabora la estructura de columnas con nombres de columnas, tipos de datos (`varchar` y `int`, que indican caracteres y enteros), y una longitud máxima para cada columna. La longitud se especifica explícitamente para las tres columnas de caracteres e implícitamente para la columna de enteros, que almacenará valores de hasta 4 bytes de tamaño.

NOTA

La sintaxis SQL no suele distinguir entre mayúsculas y minúsculas. Es posible que veas las palabras clave escritas en mayúsculas (por ejemplo, `CREATE TABLE`, `VARCHAR`, `INT`). Esto no tiene ningún efecto en el procesamiento que realiza , y puedes utilizar el estilo que prefieras.

Inserta

Para llenar la tabla con datos -en nuestro caso, con diferentes detalles del artículo - ejecuta la siguiente consulta:

```
postgres=> insert into items values ('ABCD0001', 'Blk', 'S',  
200),  
( 'ABCD0002', 'Yel', 'M', 200);
```

Esta consulta utiliza tres palabras clave: `insert`, `into`, y `values`. Las dos primeras palabras clave indican el tipo de operación y apuntan a la tabla en la que se realizará la inserción (`items`). Los valores a insertar se proporcionan entre paréntesis y deben coincidir con el orden de las columnas. Puedes insertar tantas filas como necesites del mismo modo. Si intentas insertar datos que no caben dentro de las longitudes de columna máximas definidas o no coinciden con los tipos de datos especificados, la consulta `insert` fallará. Rellena tu tabla con más artículos con diferentes precios, colores y combinaciones de tallas antes de intentar el siguiente conjunto de consultas.

Selecciona

La operación más frecuente para las pruebas de la base de datos es la *lectura*. Para leer los datos de nuestra tabla, utiliza el comando siguiente (tus resultados variarán en función de los detalles exactos de los elementos que hayas añadido):

```
postgres=> select * from items;
item_sku | color | size | price
-----+-----+-----+
ABCD0001 | Red   | S    | 200
ABCD0002 | Blk   | S    | 200
ABCD0003 | Yel   | M    | 200
ABCD0004 | Blk   | S    | 150
ABCD0005 | Yel   | M    | 100
ABCD0005 | Blk   | S    | 120
ABCD0007 | Yel   | M    | 180
(7 rows)
```

Observa las palabras clave `select`, `*` y `from` en la consulta. El símbolo comodín `*` indica que se lean todas las filas y columnas de la tabla. Si tienes que seleccionar columnas concretas, puedes especificar los nombres de las columnas, separados por comas, en lugar de `*`.

Filtrar y agrupar

La mayoría de las veces, las tablas tienen muchas filas, y filas tienen n número de columnas. Puede que necesites filtrar los datos que son relevantes para un caso de prueba concreto. La siguiente consulta reduce los resultados:

```
postgres=> select item_sku, size from items limit 3;
item_sku | size
-----+-----
ABCD0001 | S
ABCD0002 | S
```

```
ABCD0003 | M  
(3 rows)
```

Aquí seleccionamos sólo las columnas `item_sku` y `size` de la tabla `items`, y la palabra clave `limit` mostrará los *n* registros principales de los resultados. Del mismo modo, la palabra clave `where` se utiliza para definir criterios de filtrado basados en los valores de las columnas, como en el siguiente ejemplo:

```
postgres=> select color from items where size='S';  
color  
-----  
Red  
Blk  
Blk  
Blk  
(4 rows)
```

Esta consulta ha filtrado los registros de artículos que tienen un tamaño de S y sólo muestra la columna de color en esos registros. Los resultados revelan que el mismo color se repite en varias filas, pero es difícil deducir un significado de esta observación inicial. Puede ser útil ver un resumen del número de elementos de tamaño S de cada color. Para obtener este resultado, se puede utilizar la palabra clave `group by` como se indica a continuación:

```
postgres=> select color, count(*) from items where size='S'  
group by color;  
color | count  
-----+-----  
Blk   |     3  
Red   |     1  
(2 rows)
```

Ten en cuenta que la palabra clave `group by` también puede utilizarse sin la palabra clave `where`. Esencialmente, resume varias

filas en función de los criterios definidos y presenta cada grupo como una única fila en los resultados de la consulta. Podemos filtrar aún más los resultados agrupados utilizando la palabra clave `having`, como se indica a continuación:

```
postgres=> select color, count(*) from items where size='S'  
group by color  
having count(*)>1;  
color | count  
-----+----  
Blk   |     3  
(1 row)
```

Esta consulta filtra los grupos que tienen un recuento de elementos superior a 1. Ten en cuenta que la palabra clave `having` sólo puede utilizarse junto con `group by`.

También te habrás fijado en la función `count(*)` de las cláusulas anteriores `select`, que realiza la tarea de sumar el número de registros de cada grupo. Pronto veremos más funciones SQL .

Clasificación

Además de filtrar, SQL también permite ordenar los resultados en orden ascendente/descendente en función de los valores de una o varias columnas utilizando la palabra clave `order by`, como se muestra aquí:

```
postgres=> select item_sku, color, size from items order by  
price asc;  
item_sku | color | size  
-----+----+----  
ABCD0005 | Yel   | M  
ABCD0005 | Blk   | S  
ABCD0004 | Blk   | S  
ABCD0007 | Yel   | M  
ABCD0001 | Red   | S
```

```
ABCD0003 | Yel    | M  
ABCD0002 | Blk    | S  
(7 rows)
```

Una consulta para ordenar varias columnas en distintos órdenes podría tener este aspecto:

```
postgres=> select * from items order by price asc, size desc;
```

Funciones y operadores

Vimos la función `count()` en algunos de los ejemplos anteriores. El lenguaje SQL proporciona un conjunto de funciones y operadores para realizar agregaciones, comparaciones y otras transformaciones. Algunas funciones útiles son `sum()`, `avg()`, `min()`, y `max()`, que pueden utilizarse del mismo modo que `count()`. Como era de esperar, éstas encuentran la suma, la media y los valores mínimo o máximo. Del mismo modo, operadores como `and`, `or`, `not`, y `null` son útiles para filtrar valores. Por ejemplo, prueba la siguiente consulta, que devuelve elementos de color negro con tamaño S:

```
postgres=> select * from items where size='S' and color='Blk';
```

Expresiones y predicados

También podemos utilizar expresiones y predicados en SQL. Las expresiones pueden ser fórmulas matemáticas como `price+100`, y los predicados son comparaciones lógicas, que pueden dar como resultado los valores `true`, `false` o `unknown`. Por ejemplo, prueba la siguiente consulta:

```
postgres=> select * from items where price=100+50 and color is not NULL;
```

El lenguaje SQL calculará el resultado de la expresión matemática para determinar el valor de precio por el que filtrar, y también ejecuta una condición lógica, `is not null`, sobre cada uno de los valores de color de los registros para asegurarse de que no son `null`.

Consultas anidadas

También podemos anidar consultas dentro de una otra si es necesario. La subconsulta puede colocarse en cualquier lugar dentro de la consulta principal, incluso en la cláusula `where`, la cláusula `group by`, etc. Este ejemplo de consulta muestra el recuento de artículos totales y el precio medio de todos los artículos. Observa la subconsulta anidada, encerrada entre paréntesis:

```
postgres=> select count(*) , (select avg(price) from items)
  from items;
   count |      avg
-----+-----
      7 | 164.2857142857142857
(1 row)
```

Únete a

La mayoría de las veces, los datos están repartidos en múltiples tablas, y puede que necesitemos correlacionarlas para verificar un caso de prueba. Para facilitar la consulta de varias tablas, SQL proporciona la palabra clave `join`. Nos permite unir dos tablas basándonos en sus atributos comunes. Para probarlo, crea otra tabla llamada `orders` con las columnas `order_id`, `item_sku`, y `quantity` e inserta varias filas, como se ve aquí:

```
postgres=> create table orders (order_id varchar(10) ,
  item_sku varchar(10) ,
  quantity int);
postgres=> insert into orders values ('PR123', 'ABCD0001',
```

```

1) ,
('PR124', 'ABCD0001', 3), ('PR125', 'ABCD0001', 2);

```

Ahora podemos utilizar la palabra clave `inner join` para fusionar las tablas `items` y `orders` basándonos en `item_sku`, la columna común entre las dos tablas. Prueba a hacerlo con esta consulta:

```

postgres=> select * from orders o inner join items i on
o.item_sku=i.item_sku;
order_id | item_sku | quantity |item_sku | color | size | price
-----+-----+-----+-----+-----+-----+-----+
PR124   | ABCD0001 |      3 | ABCD0001 | Red  | S    | 200
PR125   | ABCD0001 |      2 | ABCD0001 | Red  | S    | 200
PR123   | ABCD0001 |      1 | ABCD0001 | Red  | S    | 200
-----+-----+-----+-----+-----+-----+

```

Como puedes ver en los resultados, las columnas de las dos tablas se han fusionado en una sola fila. Observa, sin embargo, que sólo se incluyen en la fusión los `item_sku` presentes en ambas tablas: los demás elementos de la tabla `items` no aparecen aquí. Hay un par de especialidades más que observar en la consulta: utiliza la palabra clave `on` para describir la condición sobre la que fusionar, y define alias para los nombres de las tablas (`o` para `orders` y `i` para `items`) para facilitar su uso. Los alias se reutilizan en la condición de fusión .

Aparte de la unión interna, los otros tipos de unión utilizados habitualmente en son la *unión izquierda*, la *unión derecha* y la *unión externa completa*, en las que la sintaxis de la consulta sigue siendo la misma y sólo cambia la palabra clave. Una unión izquierda toma todas las filas de la tabla mencionada en primer lugar (a la izquierda) en la condición de combinación y las combina con las filas coincidentes de la otra tabla. Cuando una fila no tiene ninguna coincidencia en la otra tabla, las columnas se llenan en con los valores de `null`. Una unión a la derecha funciona en orden inverso, incluyendo todas las filas de la segunda tabla y sólo las filas

coincidentes de la primera. Una unión externa completa devuelve todas las filas de ambas tablas, y siempre que no haya filas coincidentes, se muestran los valores `null` en los resultados fusionados.

Estas consultas de unión pueden ampliarse con palabras clave de filtrado y ordenación para ver los resultados adecuados.

Actualizar y eliminar

Las dos operaciones CRUD restantes -actualizar y eliminar consultas- se presentan aquí. Si quieras actualizar el valor de una columna y ver los resultados, utiliza las palabras clave `update` y `set`, como se ve aquí:

```
postgres=> update items set color='BK' where color='Blk';
```

Del mismo modo, si quieras eliminar algunos de los registros que has creado para hacer pruebas, utiliza la palabra clave `delete`, como se muestra aquí:

```
postgres=> delete from items where price=180;
```

NOTA

El lenguaje SQL es mucho más rico de lo que sugiere esta breve introducción . Como ya se ha mencionado, la lista de comandos que aquí se presenta se elaboró principalmente para ayudar en las pruebas manuales de bases de datos. Si te interesa aprender más sobre SQL, explora la [Guía de Bolsillo SQL](#) de O'Reilly, de Alice Zhao.

JDBC

JDBC son las siglas de Java Database Connectivity. Proporciona a un conjunto de API de Java para conectarse a bases de datos relacionales y ejecutar consultas SQL en ellas. JDBC puede combinarse con cualquiera de los conjuntos de pruebas de automatización de UI o API mencionados en los capítulos anteriores para verificar directamente los datos de la base de datos. Existen varios controladores JDBC específicos de bases de datos que pueden importarse a un proyecto como dependencia de Maven; por ejemplo, podemos utilizar el controlador JDBC PostgreSQL para conectarnos a la BD PostgreSQL que creamos anteriormente y verificar los registros de la base de datos.

En nuestra prueba utilizaremos las tres API JDBC sencillas siguientes para conectarnos a la BD y ejecutar consultas:

```
// To connect to the database
connection =
DriverManager.getConnection("jdbc:postgresql://host/database",
"username", "password");

// To execute a SQL query
Statement statement = connection.createStatement();
ResultSet results = statement.executeQuery(String query);

// To close the connection after use
results.close();
statement.close();
```

SIGUE LA PIRÁMIDE

Los casos de prueba relacionados con la BD deben añadirse a como pruebas unitarias y de integración, y no como pruebas de interfaz de usuario o de API, por las razones que se exponen en el Capítulo 3. Además, la forma ideal de crear los datos de prueba necesarios para las pruebas automatizadas de macronivel es utilizar las API de la aplicación. Esto garantiza que, aunque cambie el esquema de la base de datos, las pruebas no tengan que preocuparse por ello, ya que el código de la aplicación subyacente a las API se encargará de los cambios. Las propias API deberían raramente cambiar, ya que eso dificultaría la integración con los clientes.

Dicho esto, puede haber situaciones en las que quieras verificar una funcionalidad de extremo a extremo, lo que podría requerir verificar también los sistemas descendentes. Si tus sistemas descendentes son sistemas heredados, es posible que no dispongas de API para verificar la funcionalidad. En estos casos, la única opción será conectarse directamente a la base de datos. Por ejemplo, en el ejemplo de la aplicación de comercio electrónico, para verificar que el flujo de procesamiento de pedidos es correcto de principio a fin, puede que tengamos que crear primero un pedido en la aplicación de comercio electrónico y, a continuación, verificar directamente la base de datos del sistema de cumplimiento, suponiendo que se trate de un sistema heredado y no haya API. Este ejercicio está pensado específicamente para ayudarte en estas situaciones.

Configuración y flujo de trabajo

Vamos a ampliar el marco de automatización Java-Selenium WebDriver creado como parte del Capítulo 3 para añadir una prueba que obtenga un pedido de la tabla `orders` y haga una afirmación

sobre los valores `order_id` y `quantity`. Los pasos para ello se indican aquí:

1. Añade el controlador PostgreSQL JDBC como dependencia en el archivo POM.
2. Crea un nuevo archivo de clase de prueba en el paquete tests llamado `DataVerificationTest.java`.
3. El Ejemplo 5-1 muestra la clase `DataVerificationTest`, donde la conexión a la BD PostgreSQL se inicia antes de cada prueba y se cierra después de la ejecución de la prueba. La prueba especifica la consulta SQL que pretende ejecutar y obtiene los registros de la BD. Por último, utiliza aserciones TestNG para validar los datos devueltos.

Observa la URL JDBC utilizada en la prueba para conectarse a la base de datos. Utiliza `localhost` como nombre de host (ya que la BD está en tu máquina local); en este ejemplo el nombre de la base de datos es `postgres`, y tendrás que sustituirlo por tu nombre de usuario y contraseña. Puedes ejecutar el comando `\l` desde el cliente `psql` para listar las bases de datos existentes, y `\dt` para ver todas las tablas y sus propietarios.

Ejemplo 5-1. Prueba que implica la conexión JDBC a la BD PostgreSQL

```
package tests;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
import static org.testng.Assert.*;
import java.sql.*;

public class DataVerificationTest {

    private static Connection connection;
    private static ResultSet results;
    private static Statement statement;
```

```

@BeforeTest
public void initiateConnection() throws SQLException {
    connection = DriverManager.getConnection(
        "jdbc:postgresql://localhost/postgres",
        "newuser", null);
}

public void executeQuery(String query) throws SQLException {
    initiateConnection();
    statement = connection.createStatement();
    results = statement.executeQuery(query);
}

@Test
public void verifyOrderDetails() throws SQLException {
    executeQuery("select * from orders where
item_sku='ABCD0006'");
    System.out.println(results);
    while (results.next()){
        assertEquals(results.getString("Quantity"), "1");
        assertEquals(results.getString("order_id"), "PR125");
    }
}

@AfterTest
public void closeConnection() throws SQLException {
    results.close();
    statement.close();
}

```

4. Puedes ejecutar la prueba desde la línea de comandos (utilizando `mvn clean test`) o desde tu IDE para ver los resultados. Recuerda iniciar el servidor postgres antes de ejecutar la prueba.

Así de sencillo es. También puedes elegir para abstraer los métodos relacionados con la conexión a la base de datos en una clase separada `utils` con fines de reutilización.

Apache Kafka y Zerocode

Kafka es una plataforma de streaming distribuido de código abierto. Permite a múltiples productores y consumidores de (o editores y suscriptores, para utilizar la terminología de nuestro anterior debate sobre el streaming de eventos) intercambiar información a través de un flujo común. El equipo de LinkedIn desarrolló originalmente Kafka como solución a su lucha por agregar información de múltiples sistemas y producir métricas significativas; les permitió escalar para procesar **billones de mensajes** y consumir petabytes de datos cada día.

NOTA

Por si te lo estabas preguntando, la herramienta lleva el nombre de Franz Kafka, el famoso autor de varias obras surrealistas, entre ellas el relato "La metamorfosis", simplemente porque el ingeniero principal que dirigió los esfuerzos de desarrollo era un admirador suyo.

Exploraremos un poco esta herramienta para que te hagas una idea de lo que es y de cómo probarla. [La Figura 5-4](#) muestra un sistema Kafka de ejemplo, compuesto por un servidor (el broker), productores y consumidores.

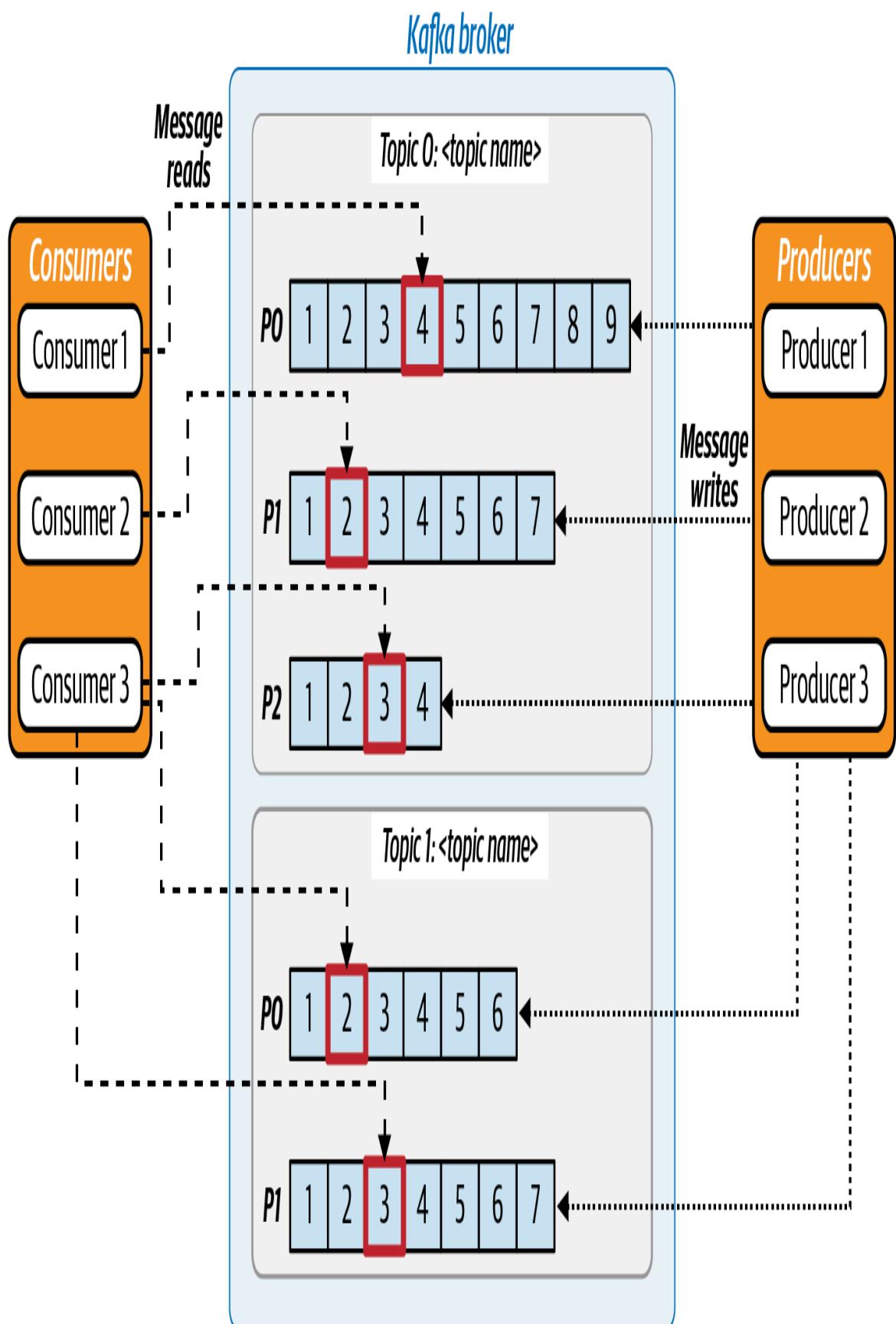


Figura 5-4. Flujo de trabajo de Apache Kafka

Para ayudarte a entender el flujo de trabajo, veamos algunos términos clave:

Mensajes

Los eventos se denominan mensajes en Kafka, y un mensaje se compone de una unidad de información, como los detalles de un pedido. Los mensajes se almacenan directamente en el disco, lo que garantiza su durabilidad.

Temas

Los mensajes se organizan bajo nombres de temas. Por ejemplo, en nuestro ejemplo de comercio electrónico, los detalles del pedido podrían enviarse al tema `orders`. Podría haber muchos mensajes con el mismo nombre de tema, enviados desde distintos productores. Esto permite la agregación de datos de múltiples productores y también facilita que los consumidores identifiquen qué mensajes son relevantes para ellos.

Particiones

Los mensajes de cada tema suelen almacenarse en varias particiones, como se ve en la [Figura 5-4](#). Los mensajes se anexan a una partición determinada, por lo que las particiones permiten la posibilidad de procesar los mensajes en una secuencia determinada si el flujo de trabajo así lo exige. Para gestionar la distribución de los mensajes a las particiones adecuadas, los productores anexan metadatos llamados *claves* dentro de los mensajes. Por ejemplo, si un conjunto de transacciones está vinculado a un identificador de cliente, éste se utiliza como clave para enviarlos a la partición correcta y, por tanto, proporcionar la posibilidad de procesar las transacciones en la secuencia correcta.

Las características relacionadas con las particiones son la forma en que Kafka consigue su rendimiento y escalabilidad. Las particiones también pueden replicarse por redundancia para evitar la pérdida de información por fallos inesperados.

Desplazamiento

Cada consumidor puede leer de muchos temas, y tiene que llevar la cuenta de qué mensajes ha leído de cada tema para no acabar reprocesando mensajes. Para ello, los consumidores utilizan el número de *desplazamiento* colocado dentro de los mensajes. Un offset es un número entero en continuo crecimiento creado por Kafka y añadido como metadatos a cada mensaje en el momento en que se produce. Así, siempre que se produce un fallo en un consumidor, éste comienza desde el último offset que procesó con éxito. Además, para incorporar un nuevo consumidor, sólo tiene que empezar desde un número de desplazamiento anterior para ponerse al día. Esta función también se denomina repetición de mensajes.

Intermediarios

El servidor Kafka, denominado broker en , media entre los productores y los consumidores. El corredor recibe mensajes de los productores, añade desplazamientos a los mensajes y los almacena en el disco ordenados por temas. Del mismo modo, responde a las peticiones de los consumidores, obteniendo los mensajes correctos de las particiones adecuadas.

Esquemas

Aunque Kafka considera los mensajes como sólo colecciones de bytes de datos, para facilitar la colaboración entre productores y consumidores necesitan acordar un formato y una estructura de datos, lo que se denomina un *esquema*. Por ejemplo, antes vimos el esquema del tema `orders`, en la [Figura 5-2](#): está compuesto por los campos `order_id`, `item_sku` y `quantity`.

Kafka admite mensajes en formatos JSON, XML y Apache Avro, entre otros. La estructura de datos de los mensajes no puede cambiar sin un cambio tanto en el código del consumidor como en el del productor. Básicamente, tiene que haber compatibilidad hacia atrás y hacia delante cuando hay diferentes versiones del esquema. Puedes compararlos con los contratos de solicitud y respuesta de un servicio web. Estas versiones del esquema se almacenan en un componente independiente llamado *Registro de Esquemas*, que ayuda a realizar comprobaciones de compatibilidad y garantiza que el contrato entre el productor y el consumidor no se rompa cuando el esquema evolucione.

Retención

Kafka retiene los mensajes durante un breve periodo de tiempo antes de eliminarlos. La configuración por defecto es persistirlos durante siete días o hasta que el tamaño de la partición alcance 1 GB. Este valor también puede configurarse por mensaje, asegurando que los mensajes con diferentes necesidades de retención sean bendecidos con un tiempo de vida apropiado.

Eso debería ser suficiente para empezar y ver cómo funcionan las cosas de primera mano.

Configurar

Puedes seguir los pasos que describe aquí para instalar Kafka en tu máquina local. Como el objetivo aquí es familiarizarte con el ecosistema de Kafka desde el punto de vista de las pruebas, haremos abstracción de los detalles de su instalación y utilizaremos contenedores Docker.

BREVE INTRODUCCIÓN A DOCKER

Supongamos que estás desarrollando una aplicación que requiere la instalación de una serie de herramientas: una BD PostgreSQL, Kafka, Nginx, etc. Un enfoque habitual para compartir los detalles de la instalación de con los nuevos miembros de tu equipo es entregarles un documento prolíjamente redactado en el que se indiquen las versiones correctas del software que hay que instalar y sus configuraciones específicas. Esto a menudo se convierte en un cuello de botella, ya que cada miembro del equipo podría estar utilizando una versión diferente del sistema operativo, enfrentarse a problemas de instalación debido a incompatibilidades con las herramientas existentes, etc. Pueden tardar unos días en resolver los problemas y tener su máquina configurada. Un enfoque más sencillo es empaquetar toda la configuración relevante para la aplicación como un *contenedor* utilizando **Docker** y distribuirlo a los miembros de tu equipo. Luego, sólo tienen que instalar Docker y descargar el contenedor, que pondrá en marcha la aplicación con un solo comando.

Docker aísla esencialmente la infraestructura y el software de la aplicación. Si ejecutas la aplicación en un contenedor, equivale a tener una máquina aislada dentro de tu máquina anfitriona con el software específico de la aplicación. La ventaja clave es que esta máquina es embarcable, a diferencia de tu máquina anfitriona. Este método es especialmente útil cuando se crean entornos de control de calidad y de producción, para que puedas disfrutar de las ventajas de utilizar exactamente los mismos binarios de aplicación en todas partes.

Un punto importante a añadir antes de instalar Docker es que es gratuito sólo para uso personal. Puede que tengas que cumplir

las políticas de tu empresa sobre portátiles cuando instales Docker en tu portátil del trabajo.

Para instalar Kafka utilizando Docker, sigue los pasos que se indican aquí:

1. Instala Docker Desktop obteniendo los binarios específicos del sistema operativo desde el [sitio web oficial](#). Una vez completado el procedimiento de instalación, la aplicación Docker Desktop aparecerá con un mensaje de inicio.
2. Al hacer clic en Iniciar, te sugerirá que ejecutes el comando `docker run -d -p 80:80 docker/getting-started`. Intenta ejecutarlo desde tu terminal para asegurarte de que Docker es accesible desde la línea de comandos. El comando básicamente descarga un contenedor `hello-world` de muestra y se asegura de que está asignado al puerto 80 de la máquina anfitriona.
3. Verás el contenedor `hello-world` ejecutándose en la aplicación Docker Desktop. Detén el contenedor una vez que esté en marcha haciendo clic en el botón Detener situado junto a él.
4. Presentaré Zerocode en breve, pero por ahora, clona el repositorio [de Zerocode Docker Factory](#) utilizando el comando `git clone`. (Consulta el [Capítulo 4](#) para obtener instrucciones sobre Git.) Este repositorio tiene todos los archivos de configuración necesarios para configurar Kafka y sus dependencias, y te permitirá escribir pruebas automatizadas utilizando la herramienta Zerocode.
5. Para finalizar la instalación de Kafka, utiliza el comando `cd` para ir a la carpeta `zerocode-docker-factory/compose` en tu terminal y ejecuta el siguiente comando:

```
$ docker-compose -f kafka-schema-registry.yml up -d
```

Cuando veas el "hecho" verde en la salida del comando, ejecuta `docker ps`, que debería mostrar un montón de contenedores levantados.

Con esto, iya tienes Kafka y sus dependencias funcionando correctamente en tu máquina! Ahora podemos escribir pruebas automatizadas para producir un mensaje y consumirlo utilizando Zerocode.

Flujo de trabajo

Zerocode es una herramienta de código abierto que permite a escribir pruebas automatizadas de estilo declarativo para API REST, API SOAP y sistemas Kafka. Los casos de prueba pueden crearse como archivos JSON o YAML y conectarse como pruebas JUnit normales. Puedes escribir pruebas para golpear una API y verificar los mensajes producidos por ella en Kafka, y viceversa. También puedes escribir pruebas para producir nuevos mensajes y verificar la estructura de los mensajes al consumirlos. El principal valor añadido de la herramienta es la capa de abstracción que oculta todas las API de Kafka necesarias para realizar esas operaciones y el código de serialización/deserialización para leer distintos tipos de respuestas.

Vamos a utilizar Zerocode para enviar un mensaje de pedido con una estructura JSON que contenga los valores `order_id`, `item_sku`, y `quantity` al broker local de Kafka que acabas de crear escribiendo casos de prueba declarativos. El mensaje se enviará al tema denominado `orders`. A continuación, consumiremos el mensaje, tal y como haría el sistema de cumplimiento, y verificaremos los detalles del pedido escribiendo de nuevo casos de prueba declarativos. Esto te dará una idea del aspecto de los mensajes y de los detalles que debes comprobar.

Aquí tienes un recorrido paso a paso de para crear pruebas con Zerocode:

1. Crea un nuevo proyecto Maven en IntelliJ, por ejemplo *KafkaTesting*, con el JDK de Java 1.8. Consulta [el Capítulo 3](#) si necesitas ayuda con este paso.
2. Añade JUnit 4 y la biblioteca *zeroCode-tdd* en tu archivo *pom.xml*, como se ve en el [Ejemplo 5-2](#).

Ejemplo 5-2. Archivo pom.xml para pruebas de Kafka con Zerocode

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                               http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>KafkaTesting</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.jsmart</groupId>
            <artifactId>zerocode-tdd</artifactId>
            <version>1.3.28</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

3. Crea una nueva carpeta llamada *kafka_servers* en *src/main/resources*.
4. En esa carpeta, añade tres archivos de propiedades: *broker.properties*, *producer.properties* y *consumer.properties*, con el contenido que se muestra en [el Ejemplo 5-3](#). Estos archivos especifican los detalles sobre cada uno de los contenedores broker, producer y consumer que se ejecutan en tu máquina local.

Ejemplo 5-3. Archivos de propiedades de prueba de Kafka

```
// broker.properties

kafka.bootstrap.servers=localhost:9092
kafka.producer.properties=kafka_servers/producer.properties
kafka.consumer.properties=kafka_servers/consumer.properties
consumer.commitSync = true
consumer.commitAsync = false
consumer.fileDumpTo= target/temp/demo.txt
consumer.showRecordsConsumed=false
consumer.maxNoOfRetryPollsOrTimeouts = 5
consumer.pollingTime = 1000
producer.key1=value1-testv ycvb

// producer.properties

client.id=zerocode-producer
key.serializer=org.apache.kafka.common.serialization.StringSerialize
r
value.serializer=org.apache.kafka.common.serialization.StringSeriali
zer

// consumer.properties

group.id=consumerGroup14
key.deserializer=org.apache.kafka.common.serialization.StringDeseria
lizer
value.deserializer=org.apache.kafka.common.serialization.StringDeser
ializer
max.poll.records=2
```

```
enable.auto.commit=false  
auto.offset.reset=earliest
```

5. Ahora crea otra carpeta bajo *src/main/resources* llamada *test_cases*.
6. Añade un nuevo archivo JSON llamado *orderMessages.json* en la carpeta *test_cases*. Aquí es donde escribirás tus pruebas en formato JSON.
7. Escribamos primero una prueba para producir un mensaje con los detalles del pedido y hacer una afirmación sobre los metadatos recibidos como respuesta del intermediario. Normalmente, obtenemos un valor de estado (igual que con las API), un número de partición y el nombre del tema en la respuesta. El archivo *orderMessages.json* (es decir, el caso de prueba del productor) se muestra en [el Ejemplo 5-4](#).

Ejemplo 5-4. Ejemplo de caso de prueba de productor en Zerocode

```
{  
  "scenarioName": "Produce an order details JSON message for the  
  orders topic",  
  "steps": [  
    {  
      "name": "produce order messages",  
      "url": "kafka-topic:orders",  
      "operation": "produce",  
      "request": {  
        "recordType" : "JSON",  
        "records": [  
          {  
            "value": {  
              "order_id" : "PR125",  
              "item_sku" : "ABCD0006",  
              "quantity" : "1"  
            }  
          }  
        ]  
      },  
    },  
  ],  
},
```

```

    "verify": {
        "status": "Ok",
        "recordMetadata": {
            "topicPartition": {
                "partition": 0,
                "topic": "orders"
            }
        }
    }
}
]
}

```

8. A continuación, tienes que conectar el caso de prueba JSON a una prueba JUnit. Para ello, crea un nuevo archivo llamado *ProducerTest.java* en *src/test/java* y añádele el código del **Ejemplo 5-5**.

Ejemplo 5-5. La clase ProducerTest

```

// ProducerTest.java

import org.jsmart.zerocode.core.domain.JsonTestCase;
import org.jsmart.zerocode.core.domain.TargetEnv;
import org.jsmart.zerocode.core.runner.ZeroCodeUnitRunner;
import org.junit.Test;
import org.junit.runner.RunWith;

@TargetEnv("kafka_servers/broker.properties")
@RunWith(ZeroCodeUnitRunner.class)
public class ProducerTest {

    @Test
    @JsonTestCase("testCases/orderMessages.json")
    public void
    verifySuccessfulCreationOfOrderDetailsMessageInBroker()
        throws Exception {

    }
}

```

Aquí, el atributo @TargetEnv indica a la prueba dónde encontrar la configuración del broker, el atributo @RunWith vincula Zerocode con JUnit, y el atributo @JsonTestCase apunta al archivo JSON o al caso de prueba que se ejecutará como parte de la prueba.

9. Puedes ejecutar la prueba desde el IDE de IntelliJ haciendo clic con el botón derecho en el botón verde situado junto al atributo @Test. Una vez superada, puedes encontrar los mensajes creados en tu instancia local de Kafka ejecutando los siguientes comandos en tu terminal:

```
// to get inside the container
$ docker exec -it compose_kafka_1 bash

// to see the records as a consumer
$ kafka-console-consumer --bootstrap-server kafka:29092
--topic orders --from-beginning
```

¡Enhорabuena por escribir tu primera prueba Kafka!

10. También puedes intentar una prueba de consumidor para validar el contenido del mensaje añadiendo el JSON que se muestra en el [Ejemplo 5-6](#) a la matriz steps del archivo *orderMessages.json*.

Ejemplo 5-6. Una prueba de consumo utilizando Zerocode

```
{
    "name": "consume order messages",
    "url": "kafka-topic:orders",
    "operation": "consume",
    "request": {
        "consumerLocalConfigs": {
            "recordType": "JSON",
            "commitSync": true,
```

```
        "showRecordsConsumed": true,
        "maxNoOfRetryPollsOrTimeouts": 3
    },
},
"assertions": {
    "size": 1,
    "records": [
        {
            "value": {
                "order_id" : "PR125",
                "item_sku" : "ABCD0006",
                "quantity" : "1"
            }
        }
    ]
}
```

Como parte de la prueba del consumidor, estamos validando el número de mensajes recibidos con el nombre del tema `orders` y el contenido del mensaje. Zerocode permite añadir validaciones a otras partes del contenido del mensaje, como desplazamiento, partición, claves/valores, etc., en el mismo estilo declarativo.

Puedes encontrar más información sobre este estilo de pruebas de Kafka en [la documentación oficial](#) de Zerocode.

Herramientas de comprobación adicionales

Aparte de las herramientas tratadas en los ejercicios, esta sección pretende arrojar luz sobre algunas otras que se utilizan habitualmente en el ámbito de la comprobación de datos para ofrecer una perspectiva más amplia .

Contenedores de prueba

Un requisito previo para algunos de los ejercicios anteriores de era configurar una base de datos PostgreSQL real en tu máquina local y

crear las estructuras de tablas pertinentes para que tus pruebas pudieran conectarse a la base de datos y verificar los datos. Si tuvieras acceso a la base de datos de la aplicación en un entorno de pruebas, las pruebas podrían conectarse a ella en su lugar. Un enfoque alternativo sería utilizar la herramienta [Testcontainers](#), que proporciona instancias de base de datos desechables en contenedores. Es especialmente útil para los desarrolladores que quieran ejecutar pruebas unitarias y de integración en sus máquinas locales sin necesidad de configurar una base de datos adecuada como requisito previo. Incluso si tuvieran una base de datos que utilizar, la probabilidad de que se contaminara debido al desarrollo activo de funciones es bastante alta. Testcontainers salva esta distancia creando una nueva instancia de base de datos en el mismo estado estable cada vez, una necesidad vital para que las pruebas tengan éxito.

Para que el código de la aplicación pueda utilizar una base de datos de Testcontainers, hay que modificar ligeramente la URL JDBC. Como alternativa, se pueden utilizar las API para instanciar instancias de bases de datos siempre que sea necesario. Por ejemplo, para activar una base de datos PostgreSQL en contenedor antes de ejecutar una prueba, se pueden añadir las siguientes líneas de código como parte de la configuración de la prueba:

```
PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>(  
    A_sample_image);  
postgres.start();
```

Puedes seguir operando sobre el objeto contenedor dentro de la prueba según sea necesario. Testcontainers proporciona diversas instancias de bases de datos, como MySQL, PostgreSQL, Cassandra, MongoDB, etc. Aporta la ventaja de trabajar con JUnit y la facilidad de lanzar la base de datos con un script init, una función o un archivo antes de que comience la ejecución de la prueba.

NOTA

Testcontainers te permite poner en marcha muchos otros tipos de contenedores aparte de las bases de datos (contenedores Kafka y RabbitMQ, contenedores que incluyen navegadores web, etc.) con una sola línea de código. También proporciona un marco genérico para importar cualquier otro contenedor personalizado que puedas necesitar para realizar pruebas. Consulta la documentación para obtener todos los detalles.

PRUEBAS DE PORTABILIDAD

La *portabilidad* se refiere a la capacidad de una aplicación para cambiar sus componentes internos sin mucho reajuste. La portabilidad de la base de datos a veces se convierte en un requisito esencial en el desarrollo de productos. Por ejemplo, supongamos que un equipo de producto desarrolla un sistema de gestión de pedidos (OMS). Se espera que el OMS como producto sea conectable entre cualquier plataforma de comercio electrónico utilizada actualmente por una empresa y sus sistemas internos heredados. En estos casos, la portabilidad de la base de datos -es decir, poder trabajar con distintos tipos de bases de datos, como Oracle o PostgreSQL- puede convertirse en un argumento de venta clave para el producto OMS, ya que significa que el equipo informático de la empresa no tiene que aprender una nueva herramienta. En un caso así, el equipo de desarrollo de OMS podría utilizar Testcontainers para verificar las funcionalidades de las aplicaciones con diferentes bases de datos como parte de las pruebas unitarias y de integración.

Deequ

Anteriormente en este capítulo, hablamos sobre los vendedores que envían sus catálogos actualizados a nuestra aplicación de

comercio electrónico como archivos, y los trabajos por lotes que los transforman en registros de base de datos. En mi experiencia trabajando en proyectos de venta al por menor, he visto que incluso los principales minoristas de EE.UU. y Europa siguen teniendo sistemas heredados que utilizan COBOL y mainframes para gestionar sus datos. Todos los días envían millones de registros con datos sobre sus catálogos de productos en forma de archivos para actualizar la disponibilidad de los artículos, y durante la noche se ejecutan trabajos por lotes para insertarlos en la base de datos de la aplicación. Los registros de los archivos suelen contener datos obsoletos o incorrectos, valores nulos, claves que faltan y muchas otras incoherencias. Es un escenario de pesadilla, ya que estos datos, una vez introducidos en la base de datos, también corrompen la aplicación. **Deequ**, una herramienta de pruebas unitarias de código abierto, puede ayudar en situaciones como ésta.

Deequ fue creado originalmente por **Amazon**, que sigue utilizándolo para comprobar la calidad de los datos producidos por sus sistemas internos. La herramienta está construida sobre Apache Spark, una plataforma de procesamiento de datos distribuidos a gran escala. Como ya se ha mencionado, Spark, entre otras muchas cosas, puede utilizarse para el procesamiento por lotes de datos a gran escala. La biblioteca Deequ encaja aquí para realizar pruebas unitarias de los datos antes y después del procesamiento por lotes. Por ejemplo, podemos añadir pruebas unitarias utilizando Deequ para verificar los tipos de datos esperados, la ausencia de valores nulos, la presencia de sólo ciertos valores permitidos, etc. En nuestra aplicación de comercio electrónico, cuando los archivos se cargan en Spark para el procesamiento por lotes, estas pruebas unitarias se pueden ejecutar primero contra ellos. Como parte de la ejecución de la prueba, los registros del archivo que no cumplen los requisitos se pondrán en cuarentena para su posterior análisis. Además, se puede escribir un conjunto de pruebas unitarias para comprobar los datos transformados después del procesamiento por lotes, con el fin de revelar errores en el propio trabajo de procesamiento por lotes.

Un ejemplo de prueba utilizando Deequ podría tener el siguiente aspecto:

```
val verificationResult = VerificationSuite()
    .onData(data)
    .addCheck(
        Check(CheckLevel.Error, "unit testing vendor files")
            .hasSize(_ > 100000) // we expect more than a million rows
            .isComplete("item_sku") // should never be NULL
            .isUnique("item_sku") // should not contain duplicates
            // should only contain the values "S", "M", "L", "XL"
            .isContainedIn("size", Array("S", "M", "L", "XL"))
            .isNonNegative("price") // should not contain negative values
    )
    .run()
```

Deequ genera diferentes métricas como resultado de las pruebas para indicar la calidad de un parámetro en todos los registros. Por ejemplo, los resultados pueden mostrar que el 90% de los valores de price son aceptables, pero el 10% no lo son, lo que indica que deben corregirse. La herramienta también proporciona otras facilidades relacionadas, como la detección de anomalías en las métricas de calidad de los datos, sugerencias automáticas de validaciones, etc.

NOTA

[TensorFlow Data Validation](#) y [Great Expectations](#) son otras dos herramientas de validación de datos similares a Deequ.

Con esto hemos terminado nuestro recorrido por el espacio de las pruebas de datos. Hemos cubierto mucho terreno: los distintos tipos de sistemas de almacenamiento y procesamiento de datos, los nuevos casos de prueba que añaden a la cartera de pruebas funcionales, un curso intensivo de SQL y algunos ejercicios prácticos

con algunas herramientas útiles para que empieces en tus propios proyectos de . La habilidad de comprobación de datos es muy necesaria en la industria actual.

Puntos clave

Éstos son los puntos clave de este capítulo:

- Hoy en día, los datos son el núcleo de cualquier aplicación en línea, y las características de la aplicación, la marca, el marketing y los aspectos de diseño giran en torno a ellos. Si se viola la integridad de los datos, las empresas se enfrentan a daños en su reputación y a clientes insatisfechos, lo que hace que la integridad de los datos no tenga precio. Por lo tanto, la comprobación de datos es una de las principales habilidades de comprobación.
- La habilidad de prueba de datos abarca el conocimiento en torno a diferentes sistemas de almacenamiento y procesamiento de datos, incluidas sus propiedades únicas, los casos de prueba específicos que imponen, y los métodos y herramientas de prueba necesarios para realizar pruebas de datos exploratorias automatizadas y manuales.
- En este capítulo se han tratado cuatro sistemas de almacenamiento y procesamiento de datos de uso común: bases de datos, cachés, sistemas de procesamiento por lotes y sistemas de procesamiento de flujos de eventos. Hemos abordado sus propiedades distintivas y los nuevos casos de prueba que exige cada uno de ellos.
- Una estrategia típica de pruebas de datos debe incluir pruebas exploratorias manuales, pruebas funcionales automatizadas, pruebas de rendimiento y pruebas de seguridad y privacidad de los datos. Como norma, al realizar pruebas de datos debes incluir casos de prueba en torno a los datos y sus variaciones,

su naturaleza distribuida, los factores de concurrencia y los fallos de red.

- Una mentalidad de búsqueda de fallos es imprescindible para las pruebas de datos, ya que el 90% de las pruebas de datos consisten en casos de prueba defectuosos. Esto contrasta con las pruebas funcionales, en las que el proceso de pensamiento gira en torno a las acciones del usuario.

1 Para una visión general de los distintos modelos de datos y lenguajes de consulta, consulta el Capítulo 2 de *Designing Data-Intensive Applications* de Martin Kleppmann (O'Reilly).

Capítulo 6. Pruebas visuales

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

La calidad visual amplifica el valor de la marca!

La calidad visual de una aplicación determina la primera impresión del cliente. Cuando el aspecto y la sensación son agradables, tienden a explorar más la aplicación. Imagina que un cliente tiene que hacer un pago en línea en , y el botón Continuar tiene el aspecto de la **Figura 6-1**. ¿Crees que tendrán confianza para seguir adelante? Lo dudo mucho. ¡Elegiría el sitio web de un competidor para realizar mi trabajo antes que arriesgarme a perder mi dinero!

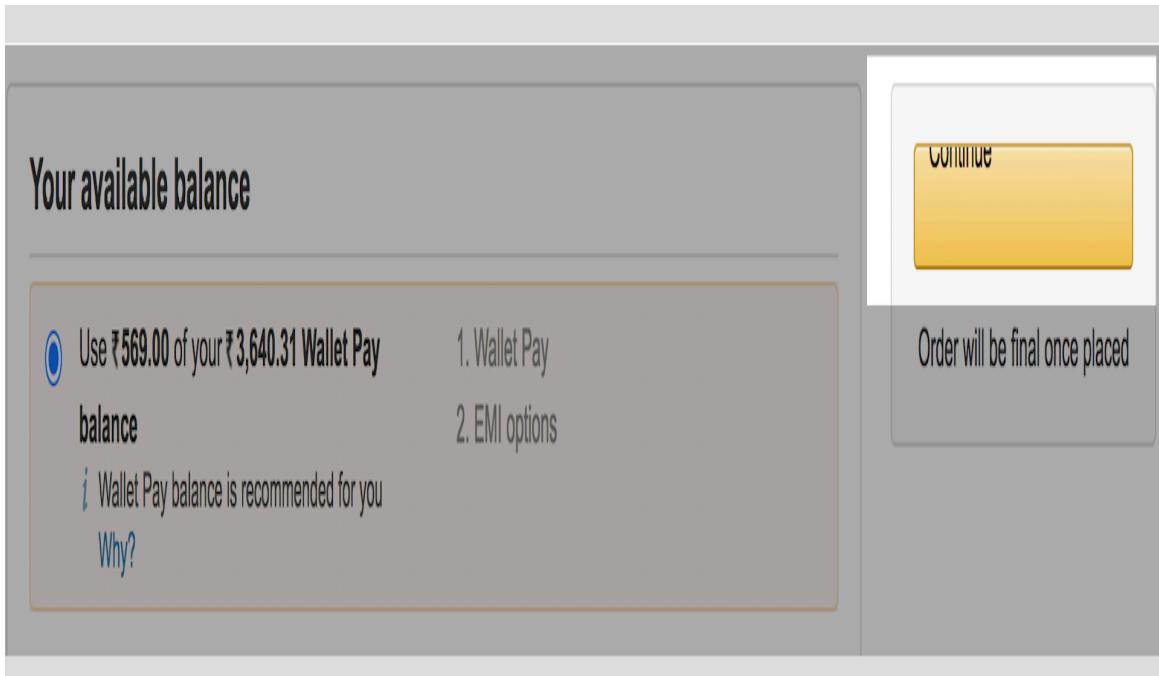


Figura 6-1. Un ejemplo de interfaz de pago con el texto del botón recortado

Con las empresas gastando mucho en estrategias para captar más clientes mediante anuncios, regalos, campañas y demás, que los equipos de software se olviden de centrarse en la calidad visual equivale a construir una casa lujosa y olvidarse de pintarla. La calidad visual de una aplicación es un factor crítico que acerca la empresa al cliente y ayuda a ganar su afinidad, y la afinidad del cliente amplifica directamente el valor de la marca. Las pruebas visuales consisten en validar la calidad visual de una aplicación, utilizando métodos de prueba tanto manuales como automatizados.

NOTA

Ten en cuenta que validar el diseño de la experiencia del usuario (UX) entra dentro de las pruebas de usabilidad, no de las pruebas visuales, y se trata en el [Capítulo 10](#).

Las pruebas visuales consisten en confirmar si la aplicación se ajusta a la apariencia *esperada* según el diseño en cuanto al tamaño y color de cada elemento, la posición relativa de los elementos y los atributos visuales similares entre dispositivos y navegadores. Este capítulo proporcionará una visión general compacta de las pruebas visuales, centrándose en casos de uso obligatorios específicos del proyecto/negocio, junto con ejercicios prácticos utilizando herramientas como Cypress y BackstopJS. También se incluye un vistazo a una nueva herramienta automatizada de pruebas visuales impulsada por IA, Applitools Eyes. Además, echaremos un vistazo holístico al panorama de las pruebas frontales y exploraremos cómo los distintos tipos de pruebas frontales, aparte de las pruebas visuales, contribuyen acumulativamente a validar la calidad visual de una aplicación.

Bloques de construcción

Empezaremos con una introducción a las pruebas visuales y a las distintas metodologías, y luego haremos un análisis coste-beneficio para determinar cuándo pueden resultar críticas para un proyecto.

Introducción a las pruebas visuales

Muchos equipos de desarrollo de software, incluso hoy en día, confían en gran medida en el globo ocular manual y en las pruebas automatizadas basadas en la interfaz de usuario para verificar la calidad visual de una aplicación. Aunque esto puede ser suficiente para algunas aplicaciones, es esencial comprender las ventajas y desventajas asociadas a este enfoque.

En primer lugar, debemos estar de acuerdo en que los ojos humanos no pueden notar los cambios a nivel de píxel, y que sólo podemos alcanzar un cierto nivel de precisión con el globo ocular manual. Por ejemplo, es bastante fácil pasar por alto detalles como los perímetros curvados de los botones, o si el logotipo se desplaza hacia arriba o hacia abajo unos pocos píxeles. De hecho, un [estudio de investigación](#) realizado en 2012 descubrió que los cambios en hasta una quinta parte del área de una imagen pueden pasar desapercibidos para los espectadores humanos. Este fenómeno, denominado *ceguera al cambio*, no tiene nada que ver con defectos de nuestra visión: es puramente psicológico. Así que puedes imaginar cómo los pequeños cambios en una aplicación podrían pasar fácilmente desapercibidos cada día en las pruebas manuales. Además, no olvidemos el tiempo y el esfuerzo que requeriría probar manualmente la calidad visual de la aplicación en multitud de navegadores, dispositivos y combinaciones de resolución de pantalla. Está claro que aquí necesitamos algo de automatización.

Pero cuando se trata de pruebas funcionales automatizadas basadas en la interfaz de usuario, aunque contribuyen parcialmente a validar la calidad visual, puede que no sea suficiente, ya que no

comprueban el "aspecto" de los elementos; simplemente identifican un elemento por su localizador, como un ID o XPath, y comprueban si se comporta funcionalmente como se espera. Por ejemplo, las pruebas basadas en la interfaz de usuario de [la Figura 6-1](#) habrían pasado porque el botón Continuar existía según su localizador con la etiqueta esperada, y al hacer clic, habría llevado al usuario a la página siguiente correctamente. No puedes culpar a la prueba en un caso como éste, ya que cumple su propósito de validar el flujo de usuario funcional de extremo a extremo. Otra advertencia sobre el uso de pruebas funcionales basadas en la interfaz de usuario para las pruebas visuales es que no puedes añadir pruebas para afirmar la presencia de cada elemento en cada página de toda la aplicación, ya que eso haría que su ejecución fuera mucho más lenta y añadiría mucho esfuerzo de mantenimiento.

Para ayudarnos a superar estos retos, ahora disponemos de herramientas de pruebas visuales automatizadas maduras, al igual que las herramientas de pruebas funcionales automatizadas. De hecho, estas herramientas existen desde hace tiempo y han adoptado diversas metodologías para realizar pruebas visuales, y con el tiempo se han hecho más estables y fáciles de usar. A continuación se indican algunas de las técnicas que emplean las herramientas existentes para realizar pruebas visuales:

- Requerir que escribamos código para verificar los aspectos CSS de los elementos (por ejemplo, una prueba para verificar si `border-width=10px`)
- Analizar el código CSS estático para encontrar problemas de incompatibilidad de los navegadores con los elementos de la interfaz de usuario
- Utilizar la IA para reconocer los cambios en la página, como lo harían los ojos humanos

- Tomar una captura de pantalla de la página y compararla píxel a píxel con una captura de pantalla base esperada

El último de estos métodos es el que más se utiliza actualmente en las pruebas de regresión visual. Por este motivo, a veces se le denomina *prueba de capturas de pantalla*. Algunas herramientas de código abierto que realizan este tipo de pruebas visuales son PhantomJS y BackstopJS. También hay herramientas de pago, como Applitools Eyes y Functionize, que funcionan con IA. Tras una comparación manual de la aplicación con el diseño UX, puedes utilizar estas herramientas para automatizar las pruebas visuales y ayudarte a detectar errores visuales, del mismo modo que las pruebas funcionales automatizadas te ayudan a detectar errores funcionales. A lo largo del desarrollo iterativo, las pruebas visuales te proporcionarán información continua sobre la calidad visual de la aplicación.

Un punto importante a tener en cuenta con las pruebas visuales automatizadas es que pueden volverse imperfectas en un proceso de desarrollo iterativo cuando no las añades en la fase adecuada. Por ejemplo, supongamos que tu equipo ha decidido reproducir la funcionalidad de inicio de sesión como parte de dos historias de usuario: una en la que se expone la funcionalidad básica y la segunda para perfeccionar la funcionalidad y el aspecto. Aunque añadir pruebas funcionales basadas en la interfaz de usuario como parte de estas dos historias de usuario tiene sentido, añadir pruebas visuales como parte de la primera historia podría no añadir valor. Así que, como parte de la planificación de la iteración, incluye claramente las pruebas visuales sólo en las historias de usuario relevantes.

Casos de uso críticos para el proyecto/negocio

Hemos hablado de por qué es importante añadir pruebas visuales automatizadas, pero puede que no sea obligatorio para todas las

aplicaciones de . Una consideración importante es el coste. Los costes de las pruebas son acumulativos. En cualquier proyecto, primero está el coste de desarrollar y mantener las pruebas funcionales basadas en la interfaz de usuario, que son un requisito absoluto para todas las aplicaciones. Además, tenemos el coste de desarrollar y mantener las pruebas visuales, aunque los dos tipos de pruebas puedan combinarse en un único conjunto de pruebas. Así pues, fíjate en la naturaleza de tu aplicación para decidir si las pruebas visuales automatizadas son imprescindibles o un "nice-to-have". Por ejemplo, para una aplicación interna que sólo utilizarán unos pocos administradores, probablemente no merezca la pena dedicar tiempo y esfuerzo a crear pruebas visuales automatizadas; las pruebas visuales manuales serán más que suficientes. Sin embargo, hay algunos casos de uso, como los siguientes, en los que las pruebas visuales automatizadas pueden aportar suficiente valor como para justificar su coste:

- Cuando desarrolles una aplicación de empresa a cliente (B2C) en , la calidad visual será un atributo de calidad crítico. Por lo tanto, necesitarás una retroalimentación continua sobre este aspecto de la aplicación durante el desarrollo. Por ejemplo, al desarrollar un sitio web de comercio electrónico global que tenga un gran número de componentes en cada página, necesitarás retroalimentación continua sobre la calidad visual, del mismo modo que recibes retroalimentación sobre la funcionalidad mediante pruebas basadas en la interfaz de usuario. En casos como éste, a menos que sólo estés desarrollando un prototipo rápido para evaluar las necesidades del mercado y planees reelaborar el diseño de tu aplicación más adelante, las pruebas visuales automatizadas te ayudarán a crear una aplicación estable.
- Cuando tengas que soportar tu aplicación en múltiples navegadores, dispositivos y resoluciones de pantalla, las

pruebas visuales automatizadas te ayudarán con la enorme carga de las pruebas de regresión.

La Figura 6-2 muestra las estadísticas de uso de la web por dispositivos, navegadores, fabricantes, sistemas operativos y resoluciones de pantalla en marzo de 2022, según gs.statcounter.com. Como puedes ver, hay más usuarios de móviles que de ordenadores de sobremesa. Chrome representa la mayor parte del mercado de navegadores, seguido de Safari. Entre los sistemas operativos, Android, Windows e iOS son actores importantes. Probar la calidad visual de tu aplicación en todas estas combinaciones podría convertirse fácilmente en un trabajo de 24 horas al día, 7 días a la semana, y las pruebas visuales automatizadas ahorrarán un esfuerzo significativo a tu equipo.

Mobile vs. desktop vs. tablet market shares

Mobile	56.45%
Desktop	41.15%
Tablet	2.40%

Browser market share

Chrome	64.53%
Safari	18.84%
Edge	4.05%
Firefox	3.40%
Samsung Internet and Opera	~5%

Device vendor market share

Samsung	28.22%
Apple	27.57%
Xiaomi	12.24%
Huawei	6.53%
Oppo	5.25%

OS market share

Android	41.56%
Windows	31.15%
iOS	16.85%
OS X	6.30%

Screen resolution market share

1920x1080	9.27%	360x800	5.35%
1366x768	7.32%	1536x864	4.05%

Figura 6-2. Datos de Statcounter sobre el uso global de dispositivos, navegadores, proveedores, SO y resolución de pantalla

- Normalmente, las empresas que poseen suites de aplicaciones tienen un equipo centralizado que desarrolla los componentes de IU, y varios equipos reutilizan estos componentes, a menudo denominados *sistemas de diseño*. Por ejemplo, los componentes de IU como los paneles de navegación de cabecera con elementos como "Preguntas frecuentes", "Contacto" y "Compartir en redes sociales" serán desarrollados por un único equipo y reutilizados en todo el conjunto de aplicaciones. Las pruebas visuales a nivel de componente se convierten en una necesidad en esos casos, ya que cualquier fallo en estos componentes estándar se filtrará por todo el conjunto.
- A veces, una aplicación se reconstruye por completo para mejorar la escalabilidad y otros aspectos de la calidad, pero la expectativa es mantener la experiencia del usuario tal como es, porque los clientes se han familiarizado con ella. Escribir pruebas visuales puede servir de red de seguridad para los equipos que trabajan en una aplicación de este tipo.
- Del mismo modo, las pruebas visuales serán útiles cuando realices una refactorización importante de una aplicación existente. Por ejemplo, mejorar el rendimiento del frontend puede requerir una reorganización considerable de los componentes de la interfaz de usuario. Las pruebas visuales automatizadas darán al equipo una gran confianza en un escenario así.
- Cuando una aplicación se amplía para llegar a públicos de distintos países, se incorporan funciones de localización como un aspecto diferente por región y texto en el idioma nativo. Estos cambios pueden afectar al diseño de la página. En estos casos, en los que deben probarse varias versiones, la

automatización de las pruebas visuales puede ser de gran ayuda.

En resumen, debes tener en cuenta factores como el impacto en el cliente, el tipo de trabajo que hay que hacer, la confianza del equipo y la cantidad de esfuerzo que requerirían las pruebas manuales a la hora de decidir si tu aplicación necesita pruebas visuales automatizadas. Si es necesario, intenta equilibrar tu estrategia de pruebas frontales con pruebas visuales mínimas sólo para los flujos más críticos. La siguiente sección presenta los distintos tipos de pruebas de que podría incorporar una estrategia de este tipo.

Estrategia de pruebas frontales

Las pruebas visuales automatizadas sólo pueden producir los beneficios adecuados cuando se equilibran proporcionalmente con otros tipos de pruebas frontales . Saber cómo son las distintas piezas del rompecabezas de las pruebas frontales te ayudará a encajarlas como requiere tu aplicación. También te darás cuenta de que algunos de los otros tipos de pruebas frontales contribuyen por sí mismos a las pruebas visuales. Debes tener esto en cuenta al planificar la estrategia de pruebas visuales de tu aplicación. También es crucial comprender dónde y cómo encajan las pruebas visuales automatizadas, para que los equipos no las propongan como solución a otros problemas no relacionados. Por ejemplo, obviamente no tienes que añadir pruebas visuales para cada tipo de mensaje de error que aparezca en la página: ese es el trabajo de las pruebas unitarias de interfaz de usuario. Así pues, ampliemos la visión y echemos un vistazo a la estrategia general de pruebas del frontend.

El código frontend de una aplicación web consta de tres partes principales: el código HTML que define la estructura básica de la página, el código CSS que da estilo a los elementos de la página y los scripts que dictan el comportamiento de esos elementos. Otro

componente importante es el navegador que reproduce este código. La mayoría de los navegadores más recientes siguen estándares para la representación de elementos. Como resultado, los marcos de desarrollo frontend pueden proporcionar soporte integrado para varios navegadores. Esto significa que se garantiza que los elementos o funciones creados con estos marcos se renderizarán correctamente en los principales navegadores, pero puede que tengas que comprobar si hay problemas de compatibilidad entre navegadores cuando utilices funciones que los marcos no han probado en navegadores antiguos y nuevos.

Para validar las distintas partes del código del frontend de , al igual que con el código del backend, podemos utilizar varios tipos de pruebas a nivel micro y macro. Es práctica habitual que los desarrolladores y los probadores se apropien colectivamente de estas pruebas del frontend. **La Figura 6-3** muestra cómo pueden emplearse las distintas pruebas de frontend de micro y macro nivel a lo largo del proceso de desarrollo para obtener una retroalimentación más rápida; en otras palabras, la figura arroja luz sobre la implementación de las pruebas de frontend de shift-left. Aunque en el **Capítulo 3** hemos tratado los fundamentos de algunos de estos tipos de pruebas, en esta sección los exploraremos desde la perspectiva del código del frontend y veremos cómo pueden contribuir a las pruebas visuales.

Shift-left frontend testing

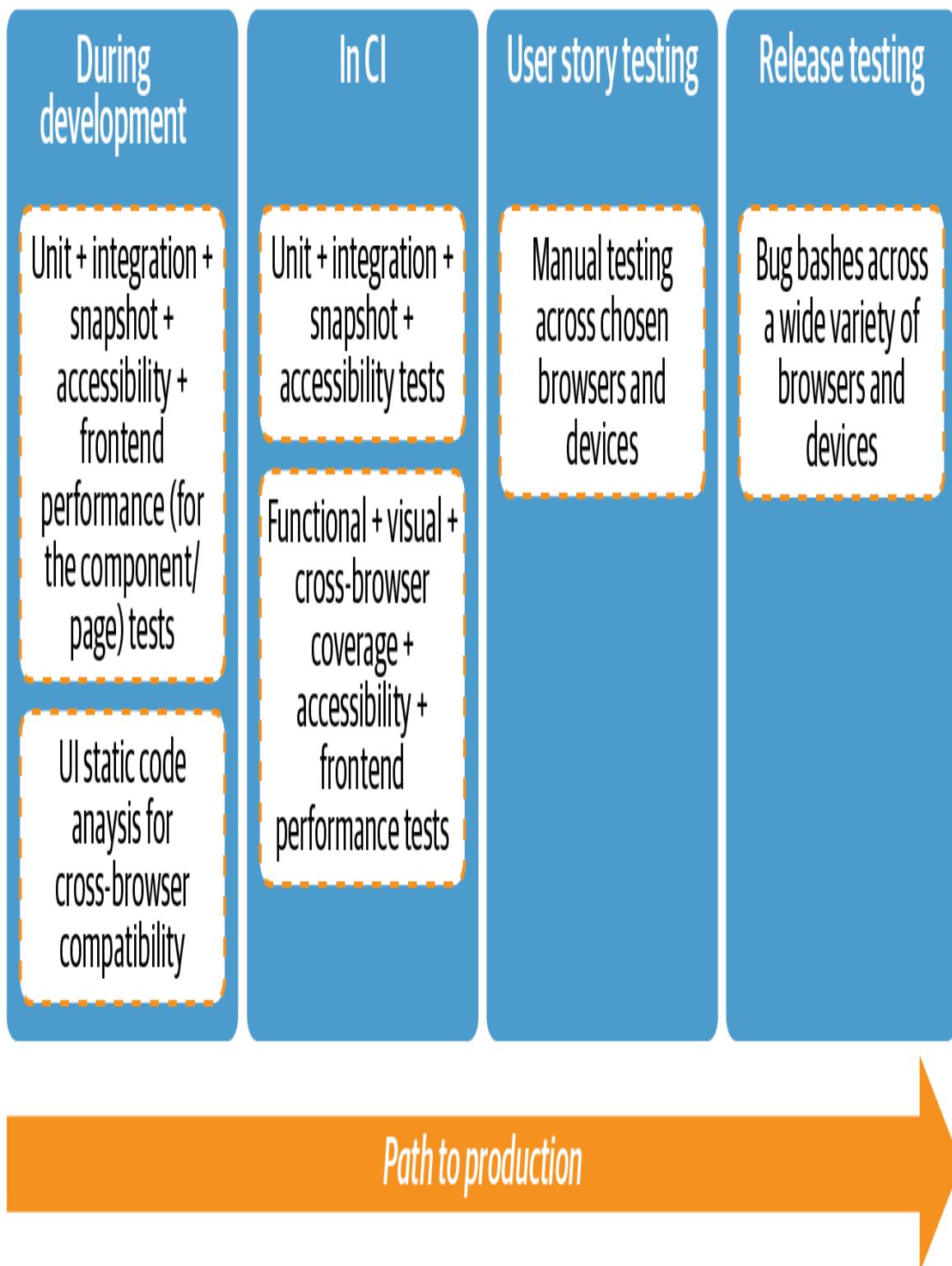


Figura 6-3. Realización de la prueba de desplazamiento a la izquierda en el

Pruebas unitarias

Las pruebas unitarias del frontend se escriben a nivel del componente para afirmar su comportamiento en diferentes estados. También contribuyen parcialmente a las pruebas visuales. Por ejemplo, las pruebas unitarias pueden confirmar el mensaje de saludo en el componente de título de la página o el estado desactivado/activado de un botón de envío. Normalmente, los desarrolladores escriben estas pruebas cuando comienzan el desarrollo, utilizando herramientas como Jest y la Biblioteca de Pruebas de React. Residen dentro de la base de código de desarrollo y proporcionan información rápida durante la fase de desarrollo.

Ejemplo 6-1 muestra una prueba unitaria de ejemplo para verificar un mensaje de saludo. Como puedes ver, obtiene el elemento de encabezamiento h1 y afirma su texto. Al afirmar que el elemento es h1, ofrece una contribución a la comprobación visual.

Ejemplo 6-1. Ejemplo de prueba unitaria con Jest

```
describe("Component Unit Testing", () => {
  it('displays greeting message as a default value', () => {
    expect(enzymeWrapper.find("h1").text()).toContain("Good Morning!")
  })
})
```

Pruebas de integración/componentes

Estas pruebas se escriben para validar la funcionalidad de un componente y las integraciones entre componentes; por ejemplo, el comportamiento del formulario de inicio de sesión, como se ve en **Ejemplo 6-2**. Esta prueba de ejemplo verifica la funcionalidad de todo el formulario, no de un solo componente como en las pruebas unitarias. Las pruebas de integración suelen simular las llamadas al servicio y los cambios de estado de los componentes de la interfaz de usuario. En **Ejemplo 6-2**, se simula la respuesta de inicio de

sesión, y la prueba afirma que, tras un inicio de sesión satisfactorio, los elementos del formulario de inicio de sesión desaparecen de la pantalla. Las pruebas de integración también pueden ayudar a verificar componentes con múltiples componentes hijos y las integraciones entre ellos en diferentes estados.

Ejemplo 6-2. Ejemplo de prueba de integración con Jest

```
test('User is able to login successfully', async () => {

  // mocking Login response
  jest
    .spyOn(window, 'fetch')
    .mockResolvedValue({ json: () => ({ message: 'Success' }) });

  render(<LoginForm />);

  const emailInput = screen.getByLabelText('Email');
  const passwordInput = screen.getByLabelText('Password');
  const submit = screen.getByRole('button');

  // enter login credentials and submit
  fireEvent.change(emailInput, { target: { value: 'testUser@mail.com' } });
  fireEvent.change(passwordInput, { target: { value: 'admin123' } });
  fireEvent.click(submit);

  // submit button should be disabled immediately
  expect(submit).toBeDisabled();

  // wait for form elements to be hidden after successful login
  await waitFor(() => {
    expect(submit).not.toBeInTheDocument();
    expect(emailInput).not.toBeInTheDocument();
    expect(passwordInput).not.toBeInTheDocument();
  });
});
```

Los desarrolladores escriben estas pruebas al final del desarrollo del componente y las mantienen dentro de la base de código de desarrollo. Al igual que las pruebas unitarias, ayudan a proporcionar información rápida sobre el comportamiento funcional durante la

fase de desarrollo y contribuyen a las pruebas visuales; por ejemplo, como en la prueba de ejemplo que se muestra aquí, al afirmar el estado desactivado de los elementos relevantes tras el inicio de sesión. Las mismas herramientas utilizadas para las pruebas unitarias pueden utilizarse para las pruebas de integración. También es una buena práctica añadir pruebas de accesibilidad a nivel de componente en .

Pruebas instantáneas

Las pruebas de instantáneas están pensadas para verificar los aspectos estructurales de componentes individuales y grupos de componentes, contribuyendo directamente a las pruebas visuales a nivel micro. Estas pruebas renderizan la estructura DOM real de los componentes utilizando renderizadores de pruebas y comparan los resultados con la estructura esperada, almacenada en una instantánea de referencia junto a la prueba. Para ello pueden utilizarse herramientas como **Jest** y **react-test-renderer**.

NOTA

Las pruebas instantáneas comparan fragmentos de código HTML. Este es diferente de las pruebas visuales, de las que hablaremos más adelante, que comparan imágenes (capturas de pantalla) píxel a píxel.

El [Ejemplo 6-3](#) muestra un ejemplo de prueba instantánea para verificar la estructura de un componente `Link` utilizando Jest.

Ejemplo 6-3. Ejemplo de prueba instantánea con Jest

```
import React from 'react';
import renderer from 'react-test-renderer';
import Link from '../Link.react';

it('renders correctly', () => {
  const tree = renderer
```

```
.create(<Link page="http://www.example.com">Sample Site</Link>)
    .toJSON();
expect(tree).toMatchSnapshot();
});
```

Por cada confirmación de código, esta prueba generará un nuevo archivo de instantánea con la estructura DOM del componente `Link`, como se muestra en el [Ejemplo 6-4](#), y lo comparará con la instantánea anterior del componente.

Ejemplo 6-4. Archivo de instantáneas generado por Jest

```
exports[`renders correctly`] = `

<a
  className="test"
  href="http://www.example.com"
  onMouseEnter={[Function]}
  onMouseLeave={[Function]}
>
  Sample Site
</a>
`;
```

Estas pruebas permiten obtener información rápida sobre los aspectos estructurales de los componentes durante la fase de desarrollo. (En cambio, las pruebas visuales requieren que la aplicación sea totalmente funcional en la máquina local). Estas pruebas adquieren importancia cuando los componentes se reutilizan en varias aplicaciones, como en los sistemas de diseño. Al igual que las pruebas unitarias y de integración, se escriben como parte del proceso de desarrollo y permanecen dentro de la base de código de desarrollo.

Se recomienda que las pruebas instantáneas mantengan un enfoque limitado, como probar un único componente pequeño (como un botón o un encabezado) o un componente algo mayor que no vaya a sufrir cambios frecuentes. Es mejor escribirlas después de desarrollar los componentes, con fines de regresión. De lo contrario, será necesario un esfuerzo adicional para mantenerlas cuando se produzca incluso un ligero cambio en el diseño.

Pruebas funcionales de extremo a extremo

Como se explica en el Capítulo 3, las pruebas funcionales automatizadas imitan las acciones de un usuario real en un sitio web en un navegador real . Estas pruebas se escriben para validar flujos de usuario funcionales completos de extremo a extremo, garantizando al mismo tiempo la integración entre los servicios frontend y backend. A diferencia de las pruebas comentadas anteriormente, las pruebas funcionales automatizadas requieren que la aplicación esté totalmente implementada y configurada con los datos de prueba adecuados. Aunque utilizan un navegador real, estas pruebas sólo contribuyen parcialmente a las pruebas visuales, ya que comprueban si un elemento está presente basándose en su localizador, pero no verifican el aspecto del elemento.

Pruebas visuales

Aunque todos los tipos de pruebas de descritos anteriormente contribuyen parcialmente a las pruebas visuales, éstas son las que realizan el trabajo más pesado. Al igual que las pruebas funcionales descritas en la sección anterior, abren la aplicación en un navegador; luego comparan una captura de pantalla de cada página con una captura de pantalla base. Las pruebas visuales pueden mantenerse como un conjunto separado de pruebas o integrarse en el conjunto de pruebas funcionales para que sean más fáciles de mantener. Para ello pueden utilizarse herramientas de código abierto como Cypress, Galen, BackstopJS y otras, así como herramientas de pago como Applitools Eyes, CrossBrowserTesting y Percy.

PRUEBAS VISUALES FRENTE A PRUEBAS INSTANTÁNEAS

Las pruebas visuales e instantáneas pueden parecer similares, pero operan en niveles diferentes, como las pruebas funcionales de extremo a extremo de alto nivel y las pruebas de API de bajo nivel. El ciclo de retroalimentación de los dos tipos de pruebas también difiere significativamente. Y, como ya hemos dicho, las pruebas visuales verifican la aplicación después de que se haya renderizado por completo en un navegador, mientras que las pruebas instantáneas proporcionan información sobre la estructura HTML y, por tanto, son fáciles de usar para el desarrollador y ayudan en las pruebas shift-left.

Las pruebas instantáneas funcionan bien cuando se centran en componentes individuales más pequeños, mientras que las pruebas visuales son ideales para validar la integración de varios componentes en una vista grande, como una página web.

Pruebas entre navegadores

Hay que hacer pruebas entre navegadores para que cumpla dos propósitos importantes: la verificación funcional y la verificación de la calidad visual entre navegadores. Aunque el flujo funcional de la aplicación no debería cambiar mucho entre navegadores, ha habido casos en los que se han observado discrepancias. Por ejemplo, en 2020 Twitter tuvo que solucionar un **problema de seguridad** por el que la información no pública de los usuarios se almacenaba en la caché del navegador Firefox. Aparentemente, Chrome no tenía ese problema. Por tanto, probar el flujo funcional en distintos navegadores debe formar parte de tu estrategia de pruebas entre navegadores.

Como primer paso en las pruebas entre navegadores, tienes que decidir en qué lista de navegadores vas a centrarte. Como hemos

visto antes, Chrome y Safari son los navegadores más utilizados en todo el mundo, y los usuarios pueden acceder a tu aplicación a través de estos navegadores utilizando distintos dispositivos, como ordenadores de sobremesa, tabletas y teléfonos. La capacidad de respuesta de la aplicación es, por tanto, otro criterio a tener en cuenta cuando realices pruebas en distintos navegadores. Una regla general es centrarse en los navegadores y resoluciones que representan el 80% de los usuarios. Puedes probar el 20% restante en pruebas de errores hacia el final del lanzamiento, en función de la prioridad.

Así que, para cumplir el propósito de obtener información funcional en todos los navegadores, dadas las advertencias de las pruebas funcionales basadas en la interfaz de usuario (que ofrecen información más lenta y no incluyen pruebas visuales), una buena estrategia podría ser elegir sólo los flujos funcionales más críticos y ejecutarlos en los navegadores seleccionados. Y para cumplir el propósito de obtener retroalimentación sobre la calidad visual, las pruebas visuales pueden reutilizarse. Pueden proporcionar información sobre tanto sobre la compatibilidad entre navegadores como sobre la capacidad de respuesta. Elige los navegadores y tamaños de pantalla utilizados por el 80% de tus usuarios finales, y añade pruebas visuales para los flujos de usuario más críticos. En general, deberías tener un puñado de pruebas funcionales y visuales (puedes combinarlas en las mismas pruebas utilizando herramientas como Cypress y AppliTools Eyes) para verificar la compatibilidad entre navegadores y la capacidad de respuesta de la aplicación.

Si te preocupa que estos esfuerzos no sean suficientes para tus necesidades de pruebas entre navegadores para todas las páginas de la aplicación, las herramientas/bibliotecas de desarrollo frontend como React, Vue.js, Bootstrap y Tailwind tienen soporte integrado para navegadores cruzados. Puedes confiar en estas herramientas para garantizar la calidad visual de los flujos de usuario no críticos de la aplicación. Una advertencia, sin embargo, es que estos marcos

sólo son compatibles con los navegadores estandarizados más recientes, y algunas de sus características pueden no ser compatibles con navegadores más antiguos.

Para comprobar si un determinado navegador admite una característica concreta de un marco de desarrollo (y, por tanto, si debes utilizarlo o no), puedes consultar las tablas de compatibilidad en [CanIUse](#). Por ejemplo, si quieras utilizar el diseño CSS *flexbox* en tu IU, puedes comprobar primero si los navegadores de destino admiten este diseño. Los equipos también pueden incluir plug-ins como [stylelint-no-unsupported-browser-features](#) para comprobar automáticamente las características CSS no compatibles con sus navegadores de destino basándose en los datos de CanIUse. Del mismo modo, el complemento [eslint-plugin-canuse](#) ayuda señalando las características de secuencias de comandos no compatibles con los navegadores de destino. También existe otra forma de proporcionar compatibilidad con versiones anteriores del código JavaScript, que consiste en utilizar transpiladores como Babel. Éstos convierten el código escrito en el último JavaScript en una versión compatible con navegadores antiguos. Utilizando todas las disposiciones mencionadas, puedes asegurarte de que, por defecto, todas las páginas de tu aplicación cumplen tus requisitos de compatibilidad entre navegadores, especialmente en lo que se refiere a su calidad visual .

DESPLAZAR A LA IZQUIERDA LAS PRUEBAS ENTRE NAVEGADORES

Empezando por la izquierda:

- Utiliza bibliotecas de desarrollo como React, Vue.js, etc., que tengan soporte para navegadores estandarizados.
- Utiliza complementos como `stylelint-no-unsupported-browser-features` y herramientas como CanIUse para asegurarte de que las funciones de la interfaz de usuario son compatibles con los navegadores de destino durante el desarrollo.
- Disponer de un puñado de pruebas funcionales basadas en la interfaz de usuario combinadas con pruebas visuales para ejecutarlas en un conjunto seleccionado de navegadores y dispositivos que cubran el 80% del uso objetivo de tu aplicación.
- Lleva a cabo batidas de errores con frecuencia para probar la mayor parte posible de el 20% restante de uso objetivo.

Pruebas de rendimiento del frontend

Las pruebas de rendimiento del frontend implican comprobar el retraso en la renderización de los componentes del frontend por parte del navegador. Puedes embellecer tu aplicación y mejorar su calidad visual añadiendo imágenes atractivas y gestos extravagantes, pero si el rendimiento no es bueno, es poco probable que los usuarios vuelvan. Está ampliamente reconocido que la descarga de los componentes del frontend representa alrededor del 80% del tiempo de carga de la página. Como resultado, equilibrar el rendimiento del frontend y la calidad visual se convierte en algo muy importante. Las herramientas y buenas prácticas para las pruebas de

rendimiento del frontend se tratan en detalle en [el Capítulo 8](#), pero merece una mención aquí dada su importancia relativa.

Pruebas de accesibilidad

La accesibilidad web es obligatoria por ley en muchos países de , y como resultado, el código frontend debe diseñarse de acuerdo con los requisitos [WCAG 2.0](#). Las características de accesibilidad tienen un impacto significativo y, en general, mejoran la calidad visual de un sitio web, ya que hacen hincapié en tener un diseño coherente en todo el sitio, disponer de texto comprensible, un espacio adecuado para hacer clic, etc. En [el Capítulo 9](#) aprenderás todo sobre las herramientas de comprobación de la accesibilidad y las buenas prácticas.

Con esto concluye nuestra visión general de los distintos tipos de pruebas frontales. En resumen, el equipo debe adaptar su estrategia de pruebas frontales en función de las intenciones de los distintos tipos de pruebas y de las necesidades de la aplicación. La recomendación general es tener más pruebas de micronivel (como las pruebas unitarias) y menos pruebas de macronivel (como las pruebas funcionales visuales y de extremo a extremo).

Ejercicios

Ahora estamos preparados para empezar a explorar las herramientas automatizadas de pruebas visuales. Existen herramientas en este espacio que pueden manejarse desde la línea de comandos o mediante código, o puedes subcontratar la tarea a proveedores de software como servicio (SaaS). Aquí, recorreremos dos ejercicios utilizando BackstopJS y Cypress. Puedes añadir estas pruebas visuales a tus canalizaciones CI y ejecutarlas como parte de cada confirmación posterior a la implementación, al igual que las pruebas funcionales.

BackstopJS

BackstopJS es una popular herramienta de pruebas visuales con una activa comunidad de código abierto que la respalda. Se presenta como una biblioteca Node fácil de integrar con CI y adopta un estilo de configuración para escribir pruebas, lo que significa que no necesitas añadir código de programación de alto nivel. Utiliza *Puppeteer*, una herramienta de automatización de IU de , para renderizar y navegar por la aplicación en Chrome y *Resemble.js* para comparar capturas de pantalla de páginas web. Tras la comparación de imágenes, BackstopJS genera los resultados como informes HTML. Tiene un buen soporte para configurar la sensibilidad de la comparación de imágenes y el automantenimiento durante los fallos de las pruebas, que exploraremos ahora.

Este ejercicio te guiará en la creación de una prueba visual utilizando BackstopJS para verificar una aplicación web en tres resoluciones: tablet, móvil y navegador normal.

Configurar

Como requisito previo, puede que necesites para configurar Node.js y Visual Studio Code (como en el ejercicio Cypress del [Capítulo 3](#)). Una vez hecho esto, sigue estos pasos para instalar la herramienta y obtener el andamiaje básico del proyecto:

1. Crea una nueva carpeta de proyecto y ejecuta el siguiente comando desde tu terminal para instalar BackstopJS:

```
$ npm install -g backstopjs
```

BackstopJS se instalará globalmente en tu máquina local para que puedas reutilizarlo en distintos proyectos. Verás que el comando ha instalado los motores chromium (para Chrome) y puppeteer junto con él.

2. Establece las configuraciones por defecto y el andamiaje del proyecto con el siguiente comando:

```
$ backstop init
```

Ahora deberías poder ver el archivo de configuración por defecto, *backstop.json*, en la carpeta del proyecto. Este es el archivo donde añadirá tus pruebas visuales como configuraciones.

Flujo de trabajo

Ahora, toma cualquier sitio web público de muestra para la prueba visual y sigue estos pasos para crear una prueba:

1. Para verificar la página web en tres resoluciones de pantalla diferentes , como dicta nuestro caso de prueba, añade la configuración *backstop.json* que se muestra en [el Ejemplo 6-5](#).

*Ejemplo 6-5. Ejemplo de prueba en el archivo de configuración *backstop.json**

```
{
  "id": "backstop_demo",
  "viewports": [
    {
      "label": "browser",
      "width": 1366,
      "height": 784
    },
    {
      "label": "tablet",
      "width": 1024,
      "height": 768
    },
    {
      "name": "phone",
      "width": 320,
      "height": 480
    }
  ],
  "onBeforeScript": "puppet/onBefore.js",
```

```

"onReadyScript": "puppet/onReady.js",
"scenarios": [
  {
    "label": "Application Home page",
    "cookiePath": "backstop_data/engine_scripts/cookies.json",
    "url": "<give site URL here>",
    "referenceUrl": "<give same URL here>",
    "readyEvent": "",
    "delay": 5000,
    "hideSelectors": [],
    "removeSelectors": [],
    "hoverSelector": "",
    "clickSelector": "",
    "readySelector": "",
    "postInteractionWait": 0,
    "selectors": [],
    "selectorExpansion": true,
    "expect": 0,
    "misMatchThreshold" : 0.1,
    "requireSameDimensions": true
  }
],
"paths": {
  "bitmaps_reference": "backstop_data/bitmaps_reference",
  "bitmaps_test": "backstop_data/bitmaps_test",
  "engine_scripts": "backstop_data/engine_scripts",
  "html_report": "backstop_data/html_report",
  "ci_report": "backstop_data/ci_report"
},
"report": ["browser"],
"engine": "puppeteer",
"engineOptions": {
  "args": ["--no-sandbox"]
},
"asyncCaptureLimit": 5,
"asyncCompareLimit": 50,
"debug": false,
"debugWindow": false
}

```

Hay algunas cosas importantes que debes tener en cuenta en este archivo de configuración:

- La matriz `viewports` tiene tres resoluciones de pantalla diferentes definidas para los usuarios de navegador, tableta y móvil.
- Los scripts de Puppeteer para interactuar con los elementos de la interfaz de usuario en Chrome se configuran con los parámetros `onBeforeScript` y `onReadyScript`. También puedes añadir tus propios scripts para definir nuevas acciones.
- El caso de prueba se define en la matriz `scenarios` con parámetros como `url`, `referenceURL`, `clickSelector`, `hideSelectors`, etc. En breve comprenderás la necesidad de cada parámetro.
- Las ubicaciones para almacenar las capturas de pantalla de referencia y de prueba se especifican mediante los parámetros `bitmaps_reference` y `bitmaps_test`. La ubicación para almacenar los informes la proporciona el parámetro `html_report`.
- El parámetro `report` se establece en el valor "browser" para permitir la visualización de los resultados en el navegador. Cuando se establece en "CI", se genera un informe en formato JUnit.
- El parámetro `engine` se utiliza para configurar el navegador adecuado. Por defecto, está configurado como "puppeteer" y se ejecuta en Chrome sin navegador. Puedes cambiarlo a "phantomjs" para ejecutar las pruebas en Firefox utilizando versiones anteriores de BackstopJS.
- El parámetro `asyncCaptureLimit`, ajustado a 5, ejecutará las pruebas en paralelo en cinco hilos.

2. El siguiente paso es tomar capturas de pantalla de referencia de la página web en diferentes tamaños de pantalla para comparar

las pruebas. BackstopJS te echa una mano aquí haciendo esto automáticamente. Abre la URL del parámetro `referenceURL` para tomar capturas de pantalla de referencia de todos los tamaños de pantalla definidos en la matriz `viewports` y las almacena en la carpeta mencionada en el parámetro `bitmaps_reference`. El comando que hace todo eso es

```
$ backstop reference
```

3. El siguiente paso es ejecutar las pruebas. Para ello, utiliza el siguiente comando:

```
$ backstop test
```

A continuación, BackstopJS verificará el sitio web indicado por el parámetro `url` comparándolo con las capturas de pantalla de referencia para todas las resoluciones.

Una vez finalizada la ejecución de la prueba, puedes ver los resultados de la prueba en el navegador, como se ve en la [Figura 6-4](#). He elegido la página de inicio de Amazon como sitio de prueba, y los resultados muestran que una prueba se ha superado y dos han fallado; en concreto, la prueba del navegador se ha superado y las otras dos han fallado. Al seleccionar las pruebas fallidas, puedo ver las capturas de pantalla de referencia y reales; además, puedo ver una tercera imagen que destaca las diferencias entre ambas. [La Figura 6-4](#) muestra las tres capturas de pantalla. Puedes ver las diferencias resaltadas en la mitad inferior de la tercera imagen.

1 passed

2 failed

Filter tests with search...

backstop_demo_Amazon_Home_page_0_document_1_tablet.png

REFERENCE



TEST



DIFF



Figura 6-4. Informe BackstopJS en la página de inicio de Amazon

Los fallos se deben al contenido dinámico de la página de inicio de Amazon. Aunque controlarías los datos de prueba en tu entorno de pruebas, si te encuentras con estos casos de contenido dinámico en tu aplicación, BackstopJS proporciona facilidades para ocultarlo durante la ejecución de la prueba. Puedes utilizar los parámetros `hideSelectors` o `removeSelectors` en el archivo `backstop.json` para ocultar o eliminar por completo el contenido dinámico. Los selectores pueden ser nombres de clase o nombres de ID, como se muestra aquí:

```
"hideSelectors": [".feed-carousel-viewport"]
```

Alternativamente, podrías elegir componentes concretos de la pantalla para la prueba visual y omitir el contenido dinámico utilizando el parámetro `selectors`.

A veces, incluso después de eliminar el contenido dinámico, puede que veas que las pruebas fallan debido a pequeños cambios a nivel de píxel que realmente no dificultan la calidad visual. En tales casos, puedes configurar la sensibilidad de las pruebas mediante el parámetro `misMatchThreshold`, que puede establecerse en un valor porcentual de 0,00% a 100,00%. Esto puede evitarte estragos en el mantenimiento de las pruebas.

BackstopJS también te ayuda con el mantenimiento de las pruebas. Supongamos que tu aplicación ha evolucionado, y tienes que actualizar las capturas de pantalla de referencia. Puedes simplemente aprobar las nuevas capturas de pantalla tomadas durante la última ejecución de la prueba con el siguiente comando (por supuesto, después de verificarlas manualmente una vez):

```
$ backstop approve
```

También puedes mejorar la prueba para buscar un producto y verificar la página del producto utilizando `keyPressSelectors`. El [Ejemplo 6-6](#) muestra la configuración para introducir el texto de búsqueda en el cuadro de búsqueda de Amazon y pulsar el botón de búsqueda.

Ejemplo 6-6. Introducir texto de búsqueda utilizando `keyPressSelectors` en el archivo de configuración `backstop.json`

```
"keyPressSelectors": [
  {
    "selector": "#twotabsearchtextbox",
    "keyPress": "Women's Tshirt"
  }
],
"clickSelectors": ["#nav-search-submit-button"],
```

Un caso de uso común en muchos proyectos es comparar las páginas en distintos entornos; por ejemplo, en la máquina local y en el entorno de pruebas. Puedes hacerlo indicando la URL local como `url` y la URL del entorno de pruebas como `referenceURL`.

Al integrarlo con CI, debes cambiar el valor del parámetro `report` por "CI" y guardar los artefactos de salida, incluidas las capturas de pantalla. También es una buena idea archivar las capturas de pantalla antiguas de para que puedas explorar el historial en caso necesario.

Ciprés

En el [Capítulo 3](#) hablamos de los requisitos previos y la configuración de un marco de automatización de pruebas funcionales utilizando Cypress. Puedes incorporar pruebas visuales como parte del mismo marco utilizando el plug-in [cypress-plugin-snapshots](#). Al igual que BackstopJS, el plug-in compara capturas de pantalla y resalta las diferencias entre ellas. También proporciona muchas de las mismas funciones, como permitirte configurar la sensibilidad de las pruebas, seleccionar elementos para la comparación, etc.

Configurar

Para empezar a utilizar el plug-in Cypress, sigue estos pasos:

1. Ejecuta el siguiente comando para instalar el plug-in:

```
$ npm i cypress-plugin-snapshots -S
```

2. En los archivos *cypress/plugins/index.js* y *cypress/support/index.js*, añade código para importar los comandos del plugin, como se muestra en el [Ejemplo 6-7](#).

Ejemplo 6-7. Configuración del plug-in Cypress

```
// cypress/plugins/index.js

const { initPlugin } = require('cypress-plugin-snapshots/plugin');

module.exports = (on, config) => {
  initPlugin(on, config);
  return config;
};

// cypress/support/index.js

import 'cypress-plugin-snapshots/commands';
```

3. El archivo de configuración de Cypress se llama *cypress.json*. Añade aquí las configuraciones de la prueba, como se ve en el [Ejemplo 6-8](#). Algunos parámetros a tener en cuenta son `threshold`, que define la sensibilidad de la prueba; `auto cleanup`, que elimina automáticamente las capturas de pantalla no utilizadas; y `excludeFields`, que excluye una serie de componentes de la comparación de capturas de pantalla.

Ejemplo 6-8. Ejemplo de prueba en el archivo de configuración cypress.json

```
{"env": {
  "cypress-plugin-snapshots": {
    "autoCleanup": false,
```

```

    "autopassNewSnapshots": true,
    "diffLines": 3,
    "excludeFields": [],
    "ignoreExtraArrayItems": false,
    "ignoreExtraFields": false,
    "normalizeJson": true,
    "prettier": true,
    "imageConfig": {
        "createDiffImage": true,
        "resizeDevicePixelRatio": true,
        "threshold": 0.01,
        "thresholdType": "percent"
    },
    "screenshotConfig": {
        "blackout": [],
        "capture": "fullPage",
        "clip": null,
        "disableTimersAndAnimations": true,
        "log": false,
        "scale": false,
        "timeout": 30000
    },
    "serverEnabled": true,
    "serverHost": "localhost",
    "serverPort": 2121,
    "updateSnapshots": false,
    "backgroundBlend": "difference"
}
}}

```

Flujo de trabajo

Para añadir pruebas visuales, el plug-in Cypress proporciona el método `toMatchImageSnapshot()`, que tomará una captura de pantalla del componente especificado o de la página actual y la comparará con la captura de pantalla base. Cypress utiliza las capturas de pantalla de la primera prueba ejecutada como capturas de pantalla base/de referencia. [El Ejemplo 6-9](#) muestra una prueba que abre la URL de una aplicación, espera a que la página sea visible y, a continuación, realiza una captura de pantalla de todo el contenido de la página para hacer una comparación de imágenes.

Ejemplo 6-9. Una prueba visual utilizando Cypress para validar la página de inicio de una aplicación

```
describe('Application Home page', () => {
  it('Visits the Application home page', () => {
    cy.visit('<give application URL here>')
    cy.get('#twotabsearchtextbox')
      .should('be.visible')
    cy.get('#pageContent').toMatchImageSnapshot()
  })
})
```

Si ejecutas la prueba contra la página de inicio de Amazon, fallará debido al contenido dinámico. Puedes ver los resultados en el ejecutor de pruebas Cypress con las diferencias de imagen resaltadas, como se ve en la [Figura 6-5](#).

Amazon Home page > Visits the Amazon home page #0

Expected result



Actual result



Figura 6-5. Resultados de la comparación de instantáneas de Cypress de la página de inicio de Amazon

Las capturas de pantalla base, de referencia y de comparación con los puntos destacados se almacenan por separado en la carpeta de capturas de pantalla, y pueden guardarse como artefactos de salida en CI para la depuración. Las ventajas de agrupar las pruebas visuales con las pruebas funcionales son un mantenimiento más sencillo y la reutilización de los datos de prueba scripts de creación.

Herramientas de comprobación adicionales

Como he mencionado antes, existen muchos métodos para incorporar pruebas visuales. Te presentaré un par de herramientas más que puedes utilizar para añadir pruebas visuales automatizadas a tu aplicación, para que tengas una idea más amplia del espacio de las pruebas visuales.

Applitools Eyes, una herramienta potenciada por IA

La IA ha penetrado en el espacio de las pruebas visuales, utilizando tecnologías de visión por ordenador y aprendizaje profundo. La visión por ordenador es una rama de la IA que permite a los ordenadores ver contenidos multimedia digitales, como imágenes y vídeos, y obtener una comprensión de alto nivel de los mismos. Desde el punto de vista de la ingeniería, el objetivo es hacer que los ordenadores comprendan y realicen tareas que puede hacer el sistema visual de un ser humano; por ejemplo, ver una página web y evaluar los cambios.

Applitools Eyes es una de esas herramientas de comprobación visual potenciada por IA. La tecnología de visión por ordenador en la que se basa, denominada *IA Visual*, tiene capacidades cognitivas que le permiten analizar la estructura y el diseño de una página, incluidos

los colores y las formas de los elementos, y detectar las diferencias como lo haría un ojo humano. Se ofrece en como solución SaaS de pago.

La IA visual está entrenada para resolver algunos obstáculos comunes de las pruebas visuales, como éstos:

Mantenimiento

Cuando las pruebas fallan debido a cambios visuales comunes, los reconoce, y al aprobarlos autocorrige las capturas de pantalla base según sea necesario.

Tratamiento dinámico de datos

Puede omitir datos dinámicos, al contrario que en una comparación exacta píxel a píxel.

Control de sensibilidad de la prueba

La sensibilidad puede ajustarse para que no tenga en cuenta los pequeños cambios en la IU que no tengan importancia.

Para configurar Eyes, tienes que registrarte en AppliTools y obtener una clave API privada para acceder al servidor Eyes alojado en su nube. El servidor Eyes realiza la comparación real. Tendrás que descargar el kit de desarrollo de software (SDK) de Eyes para poder hablar con el servidor, y configurarlo con tu clave API. El SDK de Eyes proporciona las respectivas API para capturar y enviar una captura de pantalla al servidor de Eyes. Puedes utilizar estas API dentro de tus pruebas Selenium WebDriver existentes para realizar pruebas visuales. (El SDK también integra muchas otras herramientas de desarrollo y pruebas de interfaz de usuario, como Cypress, React Storybook e incluso Appium, una herramienta de pruebas automatizadas para móviles).

Las API del SDK de Eyes que tendrás que utilizar en las pruebas son:

- eyes.open(driver) para instanciar una conexión entre la instancia WebDriver y el servidor Eyes
- eyes.checkWindow() para comprobar la calidad visual de la página en todos los dispositivos y navegadores prescritos
- eyes.closeAsync() para que el servidor Eyes sepa que la prueba ha finalizado y para generar los resultados

El Ejemplo 6-10 muestra un fragmento de código de ejemplo que integra las API dentro de las pruebas de Selenium WebDriver.

Ejemplo 6-10. Integración de Applitools Eyes con pruebas WebDriver

```
// Visual checkpoint 1 after navigating to the application home page
```

```
driver.get("<give application URL here>");  
eyes.checkWindow("Application Homepage");
```

```
// Visual checkpoint 2 after clicking a button
```

```
driver.findElement(By.className("searchbutton")).click();  
eyes.checkWindow("After clicking search button on home page");
```

Applitools Eyes aumenta el rendimiento tomando la instantánea del DOM de la página web (en lugar de la captura de pantalla) generada al ejecutar las pruebas de Selenium WebDriver y utilizando la instantánea para comparar la página web en paralelo en múltiples navegadores, dispositivos y resoluciones de pantalla alojados en la nube de Applitools. Esto no sólo proporciona un rendimiento ultrarrápido, sino que ahorra en los costes de infraestructura de tu proyecto, ya que todos los dispositivos están alojados en su nube. El producto también proporciona una vista alojada en un panel de control para gestionar el flujo de trabajo global.

Libro de cuentos

Storybook es una herramienta de código abierto que ayuda en el desarrollo de la interfaz de usuario y que es bastante popular (~70K

estrellas de GitHub) en el mundo del desarrollo frontend. Tiene integraciones con varios marcos y bibliotecas de desarrollo frontend de uso común, como React, Vue.js y Angular. Te ayuda a construir componentes de interfaz de usuario de forma aislada, lo que significa que los desarrolladores pueden construir toda la interfaz de usuario sin necesidad de configurar una compleja pila de aplicaciones, crear datos de prueba o navegar por la aplicación.

Storybook permite a los desarrolladores crear nuevos componentes y verificar manualmente su comportamiento y apariencia renderizándolos dentro de la propia herramienta. Del mismo modo, puedes verificar los diferentes estados del componente dentro de la herramienta. Storybook guarda los componentes renderizados como *historias*. Para cada estado del componente, se almacena una historia; por ejemplo, un componente botón podría renderizarse en diferentes estados como Grande, Pequeño, etc., y Storybook almacenaría cada uno de ellos como historias separadas. Esto sirve como un excelente repositorio para pruebas visuales.

De hecho, la herramienta ofrece pruebas visuales de forma inmediata a través de **Chromatic**, un servicio alojado que los mantenedores de Storybook han ampliado para permitir pruebas visuales automáticas en varios navegadores (hay disponible un nivel gratuito con límites). Puedes utilizar Chromatic para comprobar automáticamente cada nueva historia con respecto a la historia anterior: esto es realmente desplazar la comprobación visual hacia la izquierda.

En las organizaciones en las que un equipo de desarrollo de IU centralizado produce componentes compartidos para su uso en sistemas de diseño, sin un backend que integrar, esta herramienta es de gran utilidad.

Como has visto, hay múltiples formas de hacer pruebas visuales e integrarlas en el flujo de trabajo de desarrollo. Puede integrarse en el entorno de desarrollo con Storybook, integrarse en el proceso de

desarrollo con BackstopJS, o integrarse en las pruebas funcionales con Cypress y Applitools Eyes. Esto te da un abanico de opciones para obtener un feedback rápido de forma que se adapte a las necesidades de tu proyecto.

Perspectivas: Desafíos de las pruebas visuales

Uno de los retos de las pruebas visuales es elegir las herramientas que se van a utilizar. Las mencionadas aquí son sólo una pequeña muestra; con la IA y los proveedores de SaaS en el juego, las opciones son realmente numerosas. He aquí algunas características de que debes buscar al elegir una herramienta de pruebas visuales automatizadas:

- Facilidad en el flujo de trabajo, desde la creación de pruebas hasta el mantenimiento y la integración CI.
- Técnicas sólidas de gestión de capturas de pantalla. Si tienes que sustituir cientos de imágenes de base por cada pequeño cambio, pagarás enormes costes por las pruebas visuales. Las herramientas que echan una mano con la limpieza y actualización automáticas de las capturas de pantalla tienen las de ganar.
- Prueba el control de sensibilidad, para que la herramienta pueda ignorar pequeños cambios en la IU.
- Capacidad para manejar datos dinámicos.
- Capacidad para funcionar en diferentes navegadores y combinaciones de dispositivos.
- Rendimiento al ejecutar pruebas visuales en las distintas combinaciones de navegadores y dispositivos.

Aparte de la elección de las herramientas, el reto tácito es hacer que tus compañeros de equipo acepten plenamente la idea de las

pruebas visuales automatizadas, porque se requiere un esfuerzo adicional para crear y mantener estas pruebas. Pero si lo haces en las etapas adecuadas y eliges herramientas que ofrezcan opciones de mantenimiento de pruebas más sencillas, tu equipo pronto apreciará el valor. Dicho esto, recuerda también que no todas las aplicaciones necesitan pruebas visuales automatizadas; antes de seguir este camino, considera el caso de uso específico de tu proyecto/negocio y haz un análisis de costes y beneficios.

Puntos clave

Éstos son los puntos clave de este capítulo:

- Las pruebas visuales son una forma de garantizar que la aplicación tiene el aspecto que se supone que debe tener, según el diseño. La calidad visual lleva a la empresa un paso más cerca de ganarse la confianza de los clientes, lo que a su vez amplifica el valor de marca de la empresa.
- Aunque las pruebas visuales manuales y las pruebas funcionales basadas en la interfaz de usuario pueden contribuir parcialmente a las pruebas visuales, no son suficientes por sí solas, ya que las pruebas manuales pueden ser propensas a errores y las pruebas funcionales no verifican los aspectos visuales de la aplicación. De ahí que necesites pruebas visuales automatizadas independientes.
- Las pruebas visuales automatizadas pueden aportar un gran valor, dependiendo de la naturaleza de la aplicación y del alcance del trabajo. En general, piensa en factores como el impacto en el cliente, la confianza del equipo, el esfuerzo manual y el tipo de trabajo para determinar si tu aplicación necesita pruebas visuales automatizadas.
- Las herramientas de código abierto como BackstopJS, Storybook y Cypress ofrecen funciones variadas que son adecuadas para

realizar pruebas visuales automatizadas en los proyectos. Las soluciones SaaS como AppliTools Eyes y Chromatic proporcionan funciones adicionales de infraestructura y gestión del flujo de trabajo a cambio de un coste.

- Aplica pruebas visuales en las fases adecuadas de la aplicación para evitar la flaqueza y elige herramientas que puedan dar una respuesta rápida y estable al principio del ciclo de entrega.
- Las pruebas visuales son sólo una pieza del ecosistema de pruebas frontales. Aprovechar las demás pruebas frontales de micro y macronivel e idear una buena estrategia de pruebas frontales puede ayudarte a obtener información más rápida sobre la calidad visual.

Capítulo 7. Pruebas de seguridad

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

Una cadena es tan fuerte como su eslabón más débil.

-Thomas Reid, Ensayos sobre las facultades intelectuales del hombre (1786)

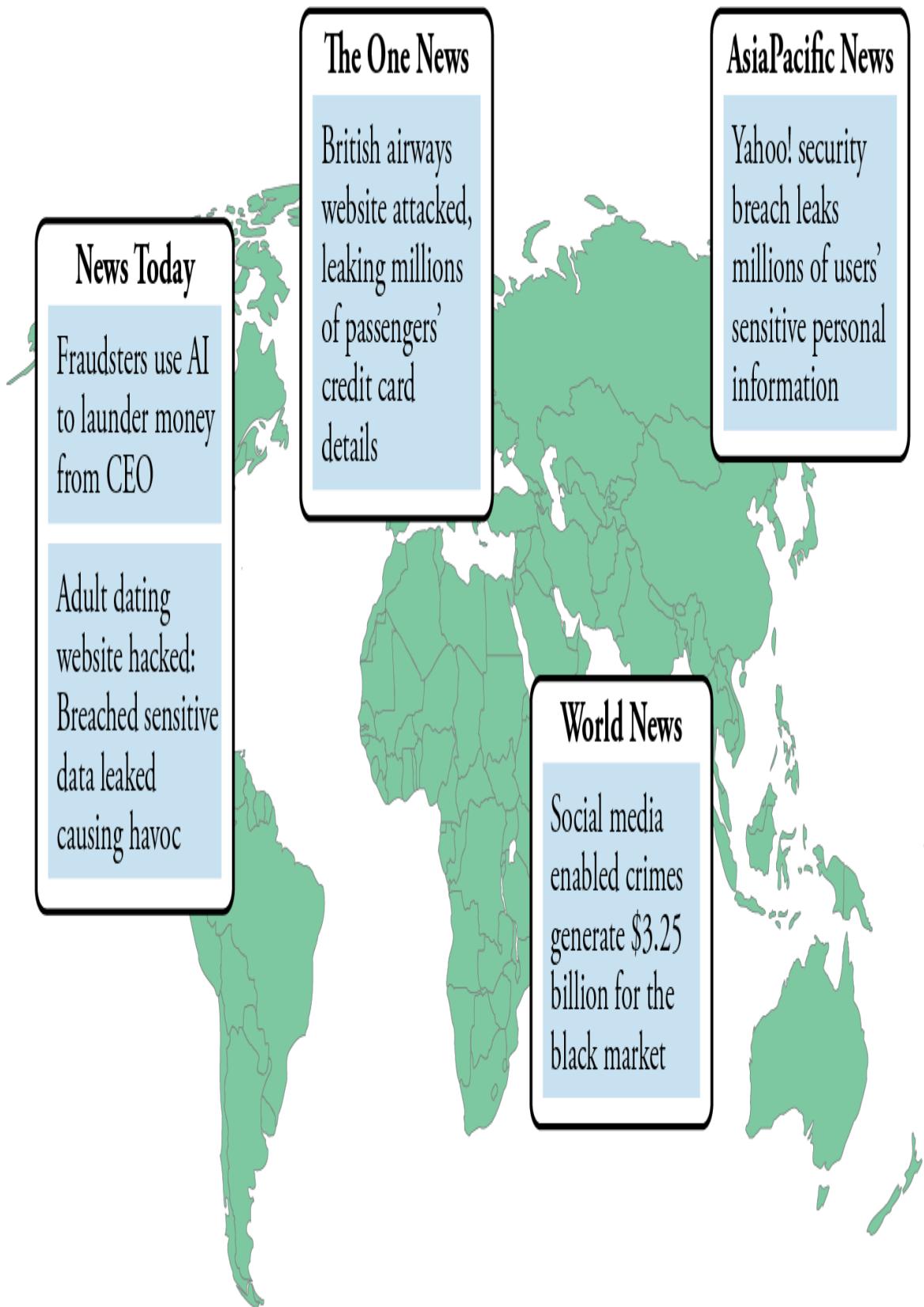


Figura 7-1. Ejemplos de titulares de noticias demuestran que la seguridad es una

preocupación mundial

Vivimos en un mundo en el que somos más susceptibles que nunca a los ciberdelitos, sobre todo cuando tenemos cuentas en las redes sociales! *Ciberdelincuencia* es un término general que se refiere a todas las actividades ilegales que pueden realizarse con un ordenador y una red, incluidos el robo financiero, el robo de activos privados como documentos de ventas e informes de investigación, la explotación de información sensible como los datos biológicos de una persona, y mucho más. Los expertos en ciberseguridad calculan que en 2025 el coste mundial anual de la ciberdelincuencia (incluidos los costes directos e indirectos para sus víctimas) alcanzará los **10,5 billones de dólares**, frente a los 6 billones estimados en 2021. Los ciberdelitos cometidos a través de las redes sociales representan la mayor parte de esta cifra, y un estudio de 2019 calcula que los ingresos anuales de los autores ascienden a **3.250 millones de dólares**. Se trata, sin duda, de cantidades enormes y, por desgracia, el botín puede incluir parte de nuestro dinero y el de nuestros amigos!

Las cifras implican que los ciberdelitos son mucho más frecuentes de lo que cabría imaginar. Como muestra **la Figura 7-1**, las noticias diarias también demuestran que los ciberdelitos no se limitan a los sitios bancarios o de redes sociales, sino que se extienden a todo tipo de sitios web: también se atacan sitios de reservas de vuelos y de citas para adultos. Analizaremos ejemplos de estos ataques reales más adelante en el capítulo para obtener más información. Todo esto plantea una cuestión crucial para los equipos de software: ¿qué medidas podemos tomar para proteger una aplicación de estos ataques?

Para construir un sistema fuerte y seguro, la recomendación del sector es construir tus defensas en profundidad, es decir, crear medidas de seguridad en múltiples capas de la aplicación, en lugar de centrarse en una única capa sólida exterior. Esto es análogo a cómo se protegían los castillos hace mucho tiempo: con un foso,

luego fuertes puertas de hierro, guardias armados en la torre del homenaje, y así sucesivamente. Cada capa tiene que ser lo bastante fuerte para ofrecer resistencia, ya que cada capa que los intrusos traspasan les da acceso a más y más recursos.

Además, recuerda que la seguridad de tu sistema es tan fuerte como tu eslabón más débil, ya sean tus administradores internos o las contraseñas pirateadas. Por tanto, explorar los eslabones débiles es un paso crucial en el esfuerzo por construir sistemas fuertes e impenetrables. Y las pruebas de seguridad son la principal actividad que desvela esos eslabones débiles.

Las pruebas de seguridad son la habilidad de pensar como un hacker y buscar posibles vulnerabilidades, amenazas y riesgos en el sistema que puedan abrir las puertas a los ciberdelitos. Los probadores profesionales de seguridad o de penetración (pen) han desarrollado esta habilidad a lo largo de muchos años, y pueden guionizar diferentes ataques a la aplicación para exponer las vulnerabilidades. Sin embargo, no hace falta ser un "pen tester" para utilizar algunas buenas herramientas y técnicas automatizadas de pruebas de seguridad en las actividades cotidianas de tu equipo para prevenir problemas de seguridad importantes.

Los fallos de seguridad, al igual que cualquier fallo funcional del software, son increíblemente costosos de solucionar cuando se detectan en las últimas fases del ciclo de desarrollo. Por lo tanto, para reducir la probabilidad de dejar lagunas en los sistemas de defensa de tu aplicación, debes practicar pruebas de seguridad por turnos, en lugar de esperar a contratar a pen testers hasta el final del ciclo de entrega. Por lo tanto, debes practicar pruebas de seguridad por turnos.

De hecho, la mejor práctica recomendada es empezar a pensar en los aspectos de seguridad de una función desde el principio de la fase de recopilación de requisitos. Por ejemplo, en una aplicación bancaria, el requisito de ocultar todas las transacciones de la cuenta

a cualquier persona que no sea el titular de la cuenta y el administrador del banco es una característica de seguridad obvia y necesaria. Del mismo modo, el requisito de disponer de una función de autenticación de dos factores proporciona una capa adicional de seguridad, como tener guardias armados además de un muro de piedra protegiendo un castillo. Incorporar las buenas prácticas de seguridad a lo largo de las fases de análisis, desarrollo y prueba te ayudará a construir sistemas sólidos y seguros.

Este capítulo cubrirá los aspectos básicos que necesitas conocer para perfeccionarte en las pruebas de seguridad, de modo que puedas desplazar tus pruebas de seguridad a la izquierda en el ciclo de entrega. Aprenderás sobre los ataques de la vida real, las vulnerabilidades de las aplicaciones y el modelo de amenazas STRIDE. El capítulo incluye un ejercicio sobre modelado de amenazas, una estrategia de pruebas que implementa pruebas de seguridad de desplazamiento a la izquierda, y ejercicios guiados sobre herramientas de pruebas de seguridad con instrucciones sobre cómo integrarlas en las canalizaciones de CI para obtener información continua sobre la seguridad de tu aplicación.

NOTA

Convertirse en un comprobador de seguridad con formación profesional no es el resultado que se espera de este capítulo. Como se ha mencionado antes, requiere años de práctica. Pero eso no elimina la responsabilidad de un equipo de software de crear aplicaciones seguras. Para ayudarte a alcanzar este objetivo, este capítulo se centra en las prácticas recomendadas, las herramientas de comprobación y, sobre todo, en cómo construir una mentalidad similar a la de un hacker.

Bloques de construcción

Para empezar a pensar como un hacker, el primer paso es observar distintos tipos de ciberataques y comprender las vulnerabilidades

que conducen a ellos. Esto te hará pensar en las amenazas potenciales para tu aplicación y te ayudará a prevenirlas. Así pues, empecemos hablando de los tipos de ciberataques más frecuentes.

TÉRMINOS COMUNES RELACIONADOS CON LA SEGURIDAD

Los siguientes son algunos términos que encontrarás en este capítulo y en otras lecturas sobre temas de seguridad:

- *Los activos* son las entidades críticas en la aplicación que hay que proteger construyendo mecanismos de defensa adecuados.
- Un *compromiso de seguridad* se produce cuando los mecanismos de defensa del sistema fallan a la hora de proteger los activos.
- *Las vulnerabilidades* son las brechas potenciales de un sistema que pueden aprovecharse para comprometer su seguridad.
- *Las amenazas* son las posibles acciones o acontecimientos negativos que pueden aprovecharse de las vulnerabilidades subyacentes para comprometer la seguridad del sistema.
- Un *ataque* es una acción maliciosa no autorizada que se realiza en un sistema con el objetivo de comprometer su seguridad.
- *La encriptación* es una técnica para codificar la información de forma que sólo el destinatario previsto que tenga la clave para descifrar el código pueda entender la información.
- *El hash* es una técnica para asignar datos de cualquier longitud a una salida de tamaño fijo mediante algoritmos (conjuntos de reglas para realizar cálculos u otras operaciones). La salida resultante, o *hash*, puede utilizarse para verificar la autenticidad de los datos, ya que incluso el más mínimo cambio en esos datos produciría un hash diferente. Los hash son inmutables -cada entrada única

produce la misma salida única- y es prácticamente inviable descifrarlos para recuperar el contenido original. Por eso se dice que el hash es una técnica unidireccional.

Ciberataques habituales

Esta sección presenta los tipos más comunes de ciberataques, con ejemplos de la vida real.

Raspado web

La forma más fácil de abusar de un sistema es para explotar los datos disponibles públicamente en el sitio web, especialmente los datos personales de los usuarios. Un ataque de web scraping utiliza software o un script para rastrear automáticamente un sitio web y recopilar información que potencialmente puede utilizarse con fines maliciosos. Las aplicaciones de las redes sociales son objetivos principales, ya que pueden proporcionar muchos datos personales, como la ubicación de los usuarios, sus números de teléfono, etc. Para un atacante, es como exponer tus objetos de valor en el escaparate. El estudio mencionado al principio del capítulo estima que los datos personales extraídos de las redes sociales generan unos ingresos de 630 millones de dólares al año.

Un ejemplo destacado de ataque de web scraping se reveló en 2019, cuando se encontraron **419 millones de registros de usuarios de Facebook**, incluidos números de teléfono, en un servidor de base de datos desprotegido en Internet. Aunque para entonces Facebook había eliminado la función de mostrar los números de teléfono en los perfiles de los usuarios, parecía que la información había sido raspada mientras la función seguía disponible. Otro ejemplo de exposición involuntaria de datos se produjo en 2018, cuando Twitter detectó **un error en una herramienta interna** que registraba las contraseñas de los usuarios en texto plano sin hash (incluso se sabe

dónde está el eslabón débil!). Afortunadamente, no había pruebas de que la información estuviera comprometida, pero, como medida de precaución, la empresa se vio obligada a pedir a sus 330 millones de usuarios que cambiaran sus contraseñas.

Los datos expuestos son siempre una vía de explotación, ya sea en un sitio web o en cualquier otro lugar. Si piensas como un hacker, debes buscar esos datos expuestos en toda tu aplicación.

Fuerza bruta

Si tuvieras que adivinar la contraseña de tu amigo , ¿cómo lo harías? Podrías probar con su fecha de nacimiento, su color favorito, el nombre de su cónyuge o una combinación de esas cosas, ¿correcto? Cuando extiendes el mismo método de ensayo y error para incluir una lista organizada de todas las combinaciones de teclas posibles, se llama ataque de fuerza bruta.

En 2016, un ataque de fuerza bruta a las bases de datos de FriendFinder Networks expuso **412 millones de registros de usuarios** con contraseñas y otra información sensible, como las preferencias sexuales de los usuarios. Al parecer, los nombres de usuario y las contraseñas se cifraron mediante el algoritmo criptográfico SHA-1, pero con las modernas técnicas de fuerza bruta y potencia de cálculo, esto no proporcionó una defensa suficiente.

Ingeniería social

La ingeniería social es la manipulación psicológica de las personas para que faciliten su información confidencial. Es posible que hayas recibido llamadas telefónicas de empresas que parecían de buena reputación, en las que amables representantes del servicio de atención al cliente te pedían los datos de tu tarjeta de crédito a cambio de algún servicio; si te han engañado, no eres el único. En 2019, **el director general de una empresa energética del Reino Unido** fue manipulado por una llamada telefónica de alguien que parecía su jefe (más tarde se descubrió que era un programa de IA entrenado)

para que transfiriera una suma global de 243.000 dólares a la cuenta del pirata informático.

Phishing

El phishing es un tipo de ataque de ingeniería social en el que un atacante envía una comunicación fraudulenta (normalmente un correo electrónico) a la víctima con la intención de robar sus datos personales. El objetivo puede ser engañado para que descargue un archivo adjunto con malware o haga clic en un enlace que le lleva a un sitio web falso que se parece mucho a uno real, donde se le piden datos de acceso o información de la tarjeta de crédito. Sería una sorpresa que no hubieras recibido al menos un correo electrónico de este tipo en el mundo actual. En 2021, **los usuarios de Microsoft 365** fueron el objetivo de una campaña de este tipo: recibieron un correo electrónico con un archivo adjunto que supuestamente contenía detalles sobre una revisión de precios, pero al abrirlo estaba programado para capturar sus datos de autenticación.

Secuencias de comandos en sitios cruzados

En un ataque de secuencia de comandos en sitios cruzados (XSS), los atacantes se aprovechan de un sitio web no seguro e inyectan código para manipular el comportamiento de la aplicación. Por ejemplo, pueden intentar inyectar código que les permita redirigir los datos de pago de los clientes a sus propios servidores. En 2018, **British Airways** fue víctima de un ataque XSS que expuso los datos de la tarjeta de crédito de 380.000 clientes. La empresa fue multada severamente debido a la falta de un sistema de defensa adecuado.

Ransomware

Los ataques de ransomware implican un malware que bloquea el sistema hasta que se paga un rescate al atacante. **El Canal Meteorológico** estuvo fuera de línea durante una hora en 2019 debido a un ataque malicioso de ransomware a su red. Como la

empresa tenía servidores de copia de seguridad, pudo superar el episodio con éxito.

Falsificación de galletas

La falsificación de cookies es una técnica para manipular las cookies de que almacenan información del usuario en un sitio web y obtener acceso a las cuentas de los usuarios. En 2017 Yahoo! reveló que alrededor de **32 millones de cuentas de usuario** habían sido violadas después de que unos hackers accedieran al código propietario de la empresa y aprendieran a falsificar cookies que les permitieran acceder a las cuentas.

Secuestro de criptomonedas

El cryptojacking se ha convertido en un ataque generalizado en estos días. Es la actividad de minar criptomonedas en secreto utilizando los dispositivos de otras personas sin su autorización.

Normalmente, los bots se configuran para rastrear los repositorios públicos de GitHub con el fin de encontrar claves de acceso a la infraestructura (por ejemplo, AWS). Una vez encontradas, las instancias se explotan en cuestión de segundos, lo que supone enormes pérdidas para los propietarios de la infraestructura. En 2018, **Tesla Inc.** fue víctima de este tipo de minería ilegal.

Los ataques tratados en esta sección son sólo la punta del iceberg. Como se ve en **la Figura 7-2**, pueden producirse muchos más tipos de ataques en diferentes capas: aplicación, infraestructura y red. Y los hackers siguen creando nuevos tipos de ataques cada día. Es responsabilidad primordial de un equipo de software adelantarse a los acontecimientos, para proteger a los clientes y a la empresa. De hecho, a menudo se trata de una responsabilidad legal, ya que los gobiernos han desarrollado normativas como el Reglamento General de Protección de Datos (RGPD) de la Unión Europea y la Directiva Revisada sobre Servicios de Pago (PSD2) para hacer cumplir la seguridad y la privacidad de los datos.

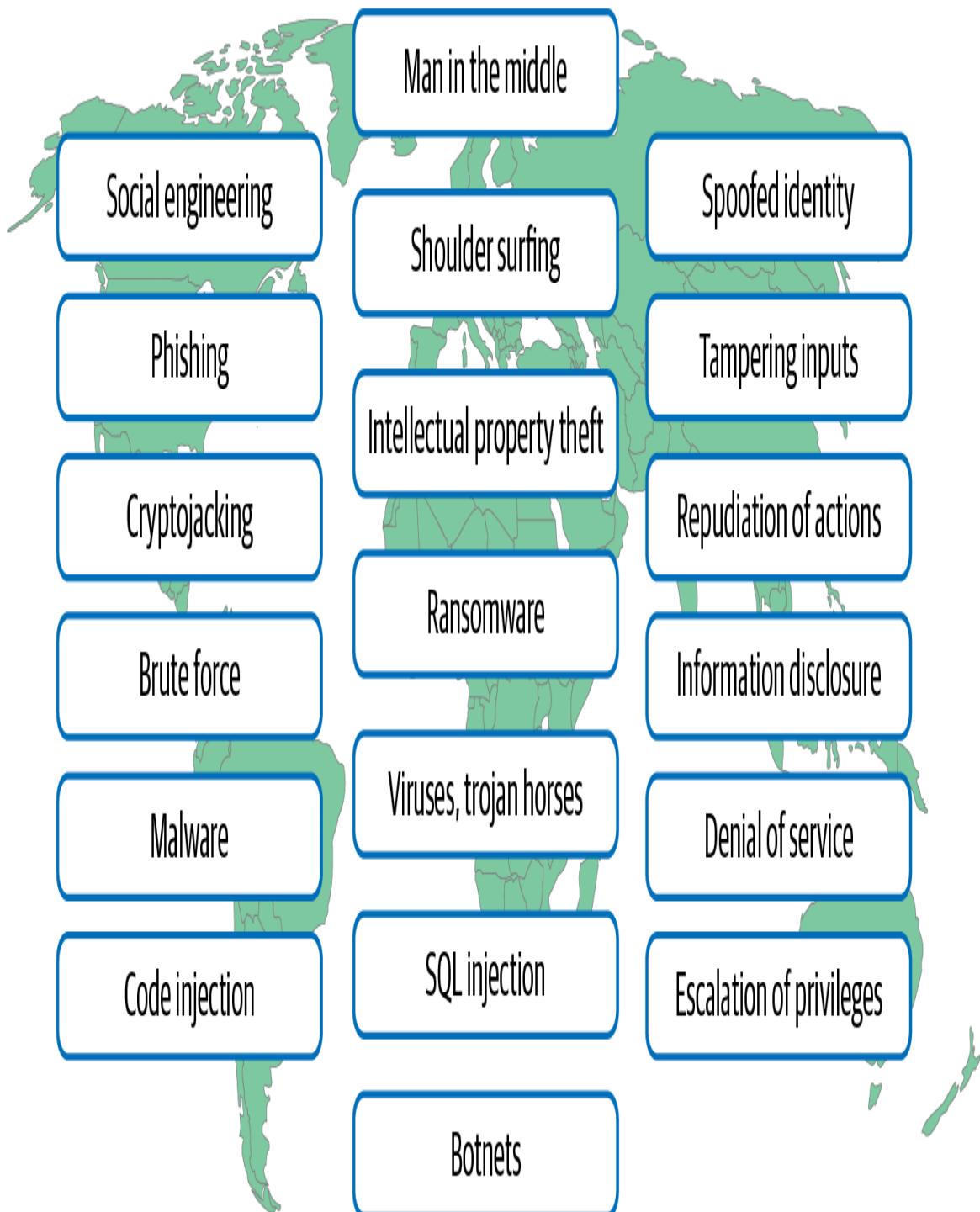


Figura 7-2. Amenazas comunes a los sistemas de software

El modelo de amenaza STRIDE

Los ejemplos de la vida real que comentamos en la sección anterior ilustran claramente los motivos de los atacantes: pretenden secuestrar o abusar de los datos, las finanzas, la infraestructura, el acceso a los servicios o la reputación de marca de los usuarios o las empresas. Estos son los principales activos que debemos proteger en cualquier aplicación, y es importante no subestimar la magnitud del reto. Una vez trabajé en un proyecto en el que diseñamos el sistema central de seguridad de un banco. La cantidad de posibles amenazas a la seguridad que se nos ocurrieron y que podrían poner en peligro cualquiera de estos cinco activos fue realmente reveladora.

Del mismo modo, cuando te plantees la seguridad de tu aplicación, tienes que pensar en todas las posibles amenazas que podrían comprometer los activos de la aplicación. Para ayudarte a identificarlas, puedes adoptar un marco de modelado de amenazas llamado modelo *STRIDE*, inventado por Loren Kohnfelder y Praerit Garg de Microsoft. *STRIDE* es el acrónimo de **Spoofedidentity (Suplantación de identidad)**, **Tamperingwith inputs (Manipulación de entradas)**, **Repudiationof actions (Repudio de acciones)**, **Informationdisclosure (Revelación de información)**, **Denialof service (Denegación de servicio)** y **Escalationof privileges (Escalada de privilegios)**. Puedes tomarlas de una en una y analizar todas las posibles amenazas de esa categoría para tu aplicación.

Veamos más detenidamente cada una de estas clasificaciones de amenazas a la seguridad.

Identidad suplantada

Los ataques de suplantación de identidad son aquellos en los que un hacker asume la identidad de otra persona para acceder a los activos. Recuerda el ejemplo de ingeniería social de antes, en el que

la IA se hizo pasar por el jefe del director general para convencerle de que enviara dinero. Estos robos de identidad son frecuentes hoy en día, y son posibles gracias a la ingeniería social, el phishing, el malware y el shoulder surfing (espiar por encima del hombro a alguien cuando introduce su información personal, como una contraseña o el pin de un cajero automático), por nombrar sólo algunos.

Algunos de los mecanismos de defensa adoptados para contrarrestar este tipo de amenaza son la autenticación multifactor, las recomendaciones de contraseñas seguras y el cifrado seguro al almacenar datos y al transferir credenciales.

Manipulación de entradas

La manipulación de entradas implica modificar algo en la aplicación (código, datos, memoria, etc.) de forma que se viole su integridad. Esto se suele hacer inyectando código malicioso, como un script, en la interfaz de usuario u otras capas. Por ejemplo, en el ejemplo de British Airways mencionado anteriormente, el hacker cambió el comportamiento del sitio web inyectando un script para obtener los datos de la tarjeta de crédito de los clientes.

Las defensas contra esta amenaza incluyen añadir validaciones apropiadas (por ejemplo, validar los campos de entrada para evitar que alguien envíe consultas SQL), mecanismos de autenticación y autorización, y otras prácticas de seguridad estándar¹ mientras se codifica para evitar vulnerabilidades que puedan dar lugar a una inyección de código.

Repudio de acciones

El repudio se produce cuando las acciones de un usuario malintencionado de no pueden probarse o rastrearse hasta él. Por ejemplo, un cliente podría negar haber recibido un artículo después de la entrega si no hay prueba de su recepción. Este es un aspecto crítico que hay que tener en cuenta al diseñar una funcionalidad, ya

que puede provocar la pérdida de bienes, reputación y dinero, y a veces acciones legales. Para combatir esta amenaza, la aplicación debe disponer de registros y mecanismos de auditoría adecuados para garantizar el no repudio.

Divulgación de información

Las amenazas de revelación de información implican que entidades no autorizadas accedan a los activos de la aplicación. Como vimos antes en el ejemplo de Twitter, los empleados pudieron ver expuestas las contraseñas de los usuarios, a las que se suponía que no tenían acceso. En ese caso, la exposición fue una consecuencia no intencionada del diseño y, afortunadamente, no se puso en peligro la seguridad. Sin embargo, los ataques para obtener acceso no autorizado a la información son habituales. Un método popular consiste en configurar un malware para que escuche en segundo plano y transmita la información de un sitio legítimo al hacker. Es lo que se conoce como ataque "*man-in-the-middle*". Para proporcionar una defensa adecuada contra este tipo de amenaza, debes construir un mecanismo de autorización fuerte en tu aplicación, encriptar todos los secretos y tener protocolos de transmisión seguros.

Denegación de servicio

Un ataque de denegación de servicio (DoS) tiene como objetivo hacer caer los servicios de la aplicación, causando pérdidas de ingresos y daños a la reputación de la empresa. Una forma en que podría manifestarse esta amenaza es mediante un ataque *de denegación de servicio distribuido* (DDoS), en el que el sistema se sobrecarga intencionadamente con millones de peticiones procedentes de múltiples dispositivos, de modo que se vuelve lento y finalmente falla.

Las defensas contra este tipo de amenaza incluyen añadir equilibradores de carga, limitar las peticiones por dirección IP, permitir sólo determinadas direcciones IP, crear copias de seguridad

del sistema, poner en marcha automáticamente nuevas máquinas cuando aumente la carga y establecer sistemas de monitoreo para alertar sobre aumentos repentinos en los volúmenes de peticiones, por citar algunas.

Escalada de privilegios

Un ataque de escalada de privilegios se produce cuando un usuario malintencionado es capaz de obtener privilegios no autorizados que le dan acceso elevado. ¡Imagina a un hacker consiguiendo privilegios de superadministrador en un sistema! En mi opinión, éste es el peor tipo de amenaza al que hacer frente, ya que puede abrir todo tipo de riesgos: robo de datos privados, denegación de servicio, pérdidas económicas, etc. Una buena práctica es seguir el principio del menor privilegio, concediendo a los usuarios del sistema sólo los privilegios mínimos que necesitan para realizar sus tareas y nada más. También podemos aplicar este principio en nuestros equipos individuales, por ejemplo, concediendo privilegios de compromiso de código sólo a los desarrolladores y ampliéndolos a otros sólo cuando sea necesario. Algunas técnicas de defensa útiles para contrarrestar esta amenaza son la actualización frecuente de los tokens de acceso, las funciones de firma múltiple para autorizar transacciones, el almacenamiento de secretos en bóvedas, etc.

Por tanto, utiliza este modelo STRIDE para hacer una lluvia de ideas sobre todas las posibles amenazas a la seguridad de tu aplicación. Evidentemente, tienes que pensar en soluciones para prevenir los distintos tipos de ataques, pero también piensa en cómo contener el impacto de un ataque en caso de que se produzca.

Vulnerabilidades de las aplicaciones

Al aprender a pensar como un hacker, hemos hablado de algunos de los tipos habituales de ataques, de los activos potenciales que los hackers pueden intentar secuestrar y de un marco de trabajo que puedes utilizar para identificar todas las posibles amenazas a tu

aplicación. El siguiente paso es acercarse al código de la aplicación para conocer las distintas vulnerabilidades de seguridad que pueden explotar las amenazas. Comprender estas vulnerabilidades te ayudará a añadir código defensivo y a realizar pruebas para detectarlas.

Inyección de código o SQL

Un atacante podría injectar comandos maliciosos o consultas SQL para alterar el comportamiento de un sitio web si está desprotegido. **El Ejemplo 7-1** muestra una consulta para recuperar el expediente de un alumno por su nombre.

Ejemplo 7-1. Una consulta SQL en código que toma una variable de entrada

```
// SQL query in the code to fetch student's record by name  
  
SELECT * FROM Students WHERE name = '$name'
```

Como puedes ver, la consulta toma una variable de entrada, \$name, del usuario. La consulta está escrita de tal forma que si un usuario malintencionado introduce una consulta SQL para eliminar toda la tabla en lugar del nombre de un alumno, funcionará perfectamente, como se ve en **el Ejemplo 7-2**.

Ejemplo 7-2. La consulta SQL injectada eliminará toda la tabla

```
// If a malicious user inputs this as the student name in the UI:  
Name: Alice'; DROP TABLE Students; --  
  
// The application will execute this command:  
SELECT * FROM Students WHERE name = 'Alice'; DROP TABLE Students; --'
```

Por lo tanto, es necesario comprobar la correcta validación de las entradas .

Secuencias de comandos en sitios cruzados

Como ya se ha comentado, el cross-site scripting consiste en ejecutar un script en el navegador de la víctima que permite al

atacante apoderarse de la sesión del usuario, redirigirlo a un sitio malicioso o incluso alterar el código para desfigurar el sitio web. Este tipo de ataque es frecuente cuando no hay validación o sanitización adecuada de la entrada del usuario.

Por ejemplo, un usuario de Twitter publicó el sencillo script JavaScript que se muestra en [la Figura 7-3](#) para revelar una vulnerabilidad XSS en la aplicación TweetDeck. El código hace que el tuit se retuitee automáticamente cada vez que aparece en la línea de tiempo de alguien, apareciendo después un cuadro de diálogo de alerta. Esto podría haberse evitado si la aplicación hubiera validado correctamente el texto de la publicación para detectar la presencia de scripts, pero como no lo hizo, los navegadores de los espectadores ejecutaron inocentemente el script.



Figura 7-3. Tweet con XSS

Vulnerabilidades conocidas no gestionadas

Si una aplicación depende de software de terceros (por ejemplo, sistemas operativos, bibliotecas, marcos de trabajo, herramientas), podría aprovecharse una vulnerabilidad en cualquiera de ellos para acceder al sistema. Estas vulnerabilidades suelen encontrarse y corregirse, y los mantenedores envían parches como actualizaciones. Sin embargo, es posible que los equipos no actualicen regularmente todos los componentes vulnerables de sus aplicaciones, lo que hace

que permanezcan expuestos. Las herramientas pueden proporcionar cierta ayuda en este caso. Por ejemplo, **Dependabot** de GitHub ofrece actualizar automáticamente cualquier dependencia con vulnerabilidades conocidas en el código de la aplicación. Del mismo modo, las herramientas de escaneo de vulnerabilidades como Snyk y OWASP Dependency-Check ayudan a resaltar los componentes vulnerables.

Autenticación y mala gestión de la sesión

A veces, los mecanismos de autenticación de un sitio web no son infalibles, lo que deja espacio para que los atacantes roben tokens de sesión y exploten los privilegios que obtienen. Si almacenas identificadores de sesión y datos sensibles del usuario en cookies de sesión, tienes que actualizarlas muy a menudo e invalidar las cookies más antiguas. Además, debes tener cuidado con vulnerabilidades como exponer los ID de sesión en las URL, utilizar conexiones no cifradas para enviar datos sensibles de autenticación, etc.

Datos privados no encriptados

Los usuarios suelen ser presa de la vulnerabilidad común de los datos privados sin cifrar, como en el caso descrito anteriormente , donde se capturaron los números de teléfono de los usuarios de Facebook y se almacenaron en una base de datos. Debes asegurarte de que los datos privados no se exponen sin cifrar en registros, bases de datos, repositorios de código, documentación de proyectos, servicios alojados públicamente, etc. Además, elige **algoritmos criptográficos** de gama alta como AES, HMAC o SHA-256, con **técnicas** dinámicas de sal y **pimienta**, para ayudar a proteger los datos en tránsito y en reposo.²

Mala configuración de la aplicación

Es un error frecuente dar permisos generales de administrador en a todos los usuarios de la aplicación, sin más razón que ahorrar esfuerzos de mantenimiento. Configurar mal los permisos

pertinentes a usuarios, carpetas, sistemas, etc., puede dar lugar a accesos no autorizados y a una escalada de privilegios al contenido de la aplicación, como la base de datos y los puntos finales de administración, de los que se puede abusar fácilmente. Los equipos deben adherirse estrictamente al principio del menor privilegio, como se ha comentado antes.

Exposición de secretos de aplicación

Una práctica común que conduce al compromiso es hardcoding the application secrets, such as environment credentials, superuser credentials, etc., in code and configuration files as plain text. Una medida adecuada es utilizar servicios de gestión de secretos, como las bóvedas, y acceder a los secretos sólo desde allí. Esto se aplica al código de la aplicación, a los conductos de CI/CD, a los archivos de configuración y a todos los lugares donde puedas tener que acceder a los secretos.

La lista de esta sección ha puesto de relieve una serie de vulnerabilidades que hay que tratar cuidadosamente durante el desarrollo y las pruebas. El Proyecto Abierto de Seguridad de las Aplicaciones Web (OWASP), una organización sin ánimo de lucro dirigida por la comunidad, ha identificado las **10 vulnerabilidades más comunes** en la web, que a ti también podría interesarle leer.

Modelización de amenazas

Habiendo llegado hasta aquí, puede que estés pensando en las amenazas y vulnerabilidades de seguridad a las que podría estar expuesta tu aplicación en este momento. En esta sección hablaremos de un enfoque metódico del modelado de amenazas - una forma estructurada de agregar todas las amenazas potenciales a la seguridad- que puedes aplicar a tu propia aplicación.

Una buena práctica general es realizar este ejercicio de modelado de amenazas para cada pequeño ámbito de la aplicación; por ejemplo,

podrías hacer 15 minutos de modelado de amenazas por historia de usuario. Una vez modeladas las amenazas a la seguridad, puedes priorizarlas en función del impacto y la probabilidad de riesgo, y luego incorporar las soluciones como parte de la historia o como una nueva función. Cuando priorices las amenazas, utiliza la siguiente regla general: *el coste de crear medidas de seguridad para hacer frente a una amenaza potencial no debe ser superior al valor del activo que intentas proteger.*

Por ejemplo, supongamos que tu equipo está desarrollando una plataforma de blogs. Antes de construir la página de inicio, hacéis un ejercicio de modelado de amenazas de 15 minutos y descubrís la amenaza potencial de que un ataque de ransomware podría hacer caer la página. El equipo propone implantar un sistema de monitoreo de seguridad como solución. Calculan que el sistema de monitoreo costará 400.000 dólares al año. ¿Merece la pena implantar esta solución para protegerse de la amenaza del ransomware? Tal vez no, ya que el coste podría ser superior al beneficio anual de la empresa. Además, ¿con qué frecuencia se produce un ataque de ransomware en una plataforma de blogs? La probabilidad es muy baja. Por otro lado, una amenaza como un ataque de inyección de código en un sitio web de comercio electrónico que podría conducir a la pérdida de datos de tarjetas de crédito puede considerarse una amenaza de alto impacto y alta probabilidad.

Una vez que hayas identificado y priorizado las amenazas, aborda las soluciones en la misma historia de usuario o, si es necesario, crea nuevas historias de "abusador" o "usuario malvado" con este fin. Por ejemplo

Como usuario abusivo, no puedo injectar código para redirigir el contenido del sitio web.

También puedes derivar casos de prueba relacionados con la seguridad a partir del ejercicio de modelado de amenazas y de los

criterios de aceptación de las historias de abuso con fines de desarrollo y prueba.

Pasos del modelado de amenazas

Echemos un vistazo más de cerca al marco para completar un ejercicio de modelado de amenazas. Se recomienda hacer este ejercicio como un equipo con todos los roles representados. Una pizarra blanca con pegatinas de colores funciona bien, ya que puedes capturar los pensamientos de tu equipo y categorizarlos rápidamente. En un mundo remoto, elige herramientas como MURAL para realizar el ejercicio. Una vez reunido tu equipo, navega por los siguientes hitos:

Define la característica

El primer paso es definir el alcance de la función de modelado de amenazas. A continuación, dibuja los flujos de usuarios y los distintos tipos de usuarios o actores del sistema. Una vez que esto esté claro, traza el flujo de datos de un componente a otro. De este modo, abarcarás los flujos de usuarios, los actores, el flujo de datos y la integración entre los componentes del sistema.

Definir los activos

El segundo paso es identificar en los activos de la función que hay que proteger. Analiza el impacto de perder cada activo y capta la gravedad del riesgo.

Pensamiento de sombrero negro

A continuación, abre la puerta al pensamiento de sombrero negro , en el que empiezas a pensar como un hacker y se te ocurren formas de atacar los activos de la aplicación. Aquí, la mentalidad del equipo debe ser "¡Vamos a romper el sistema!". Utiliza el modelo STRIDE para estructurar este debate. Deja que la imaginación fluya libremente sin debatir si algo es realmente una

amenaza o no por ahora, y capture todas las ideas como notas adhesivas.

Prioriza las amenazas y capta las historias

Analiza la probabilidad y el impacto potencial de las amenazas que has identificado, y dales prioridad. Captúralas como historias de abuso para que el equipo pueda actuar sobre ellas después de la sesión de tormenta de ideas de modelado de amenazas.

Ahora que conoces los conceptos básicos, estás preparado para adquirir experiencia de primera mano realizando un ejercicio de modelado de amenazas de muestra.

Ejercicio de modelización de amenazas

Para este ejercicio, supongamos que tenemos una aplicación para gestionar (crear/ver/actualizar/borrar) pedidos en una tienda minorista. La aplicación tiene una interfaz de usuario web y servicios REST backend para realizar operaciones de negocio sobre los datos de los pedidos almacenados en la base de datos. Vayamos al primer paso de definir los actores, el flujo de datos y las integraciones entre los distintos componentes.

NOTA

Este ejercicio sólo pretende que te familiarices con los pasos del modelado de amenazas, no que proporciones un modelo de amenazas exacto para un sistema de gestión de pedidos.

Los usuarios del sistema son

- El asistente de tienda que realiza, edita y cancela pedidos
- El administrador del sistema que gestiona la infraestructura, las configuraciones y las Implementaciones

- El ejecutivo de atención al cliente que utiliza la aplicación para responder por teléfono a consultas relacionadas con el estado de los pedidos

El flujo de usuario es sencillo: el dependiente de la tienda y el ejecutivo de atención al cliente tienen que acceder a la aplicación para ver la lista de pedidos más recientes, y se les ofrecen opciones para gestionar los pedidos. **La Figura 7-4** muestra la integración de componentes y el flujo de datos entre los componentes para ayudar a ese flujo de usuario.

Store Customer
assistant service exec

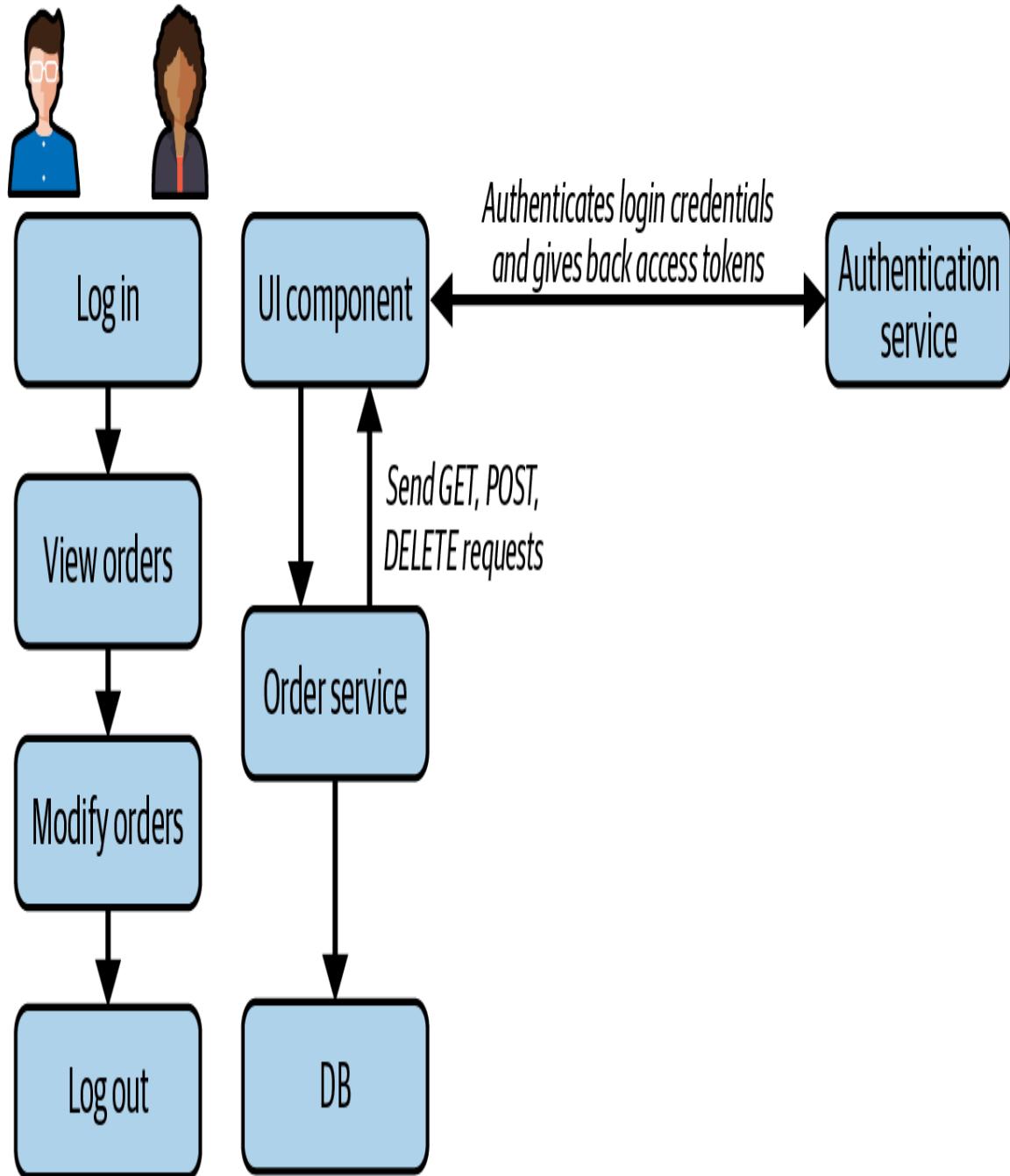


Figura 7-4. Flujos de usuarios y flujos de datos del sistema de gestión de pedidos de muestra

Del mismo modo, como se muestra en la [Figura 7-5](#), el administrador del sistema tiene que iniciar sesión en las máquinas virtuales (VM) para ejecutar cualquier script o configurar la infraestructura.

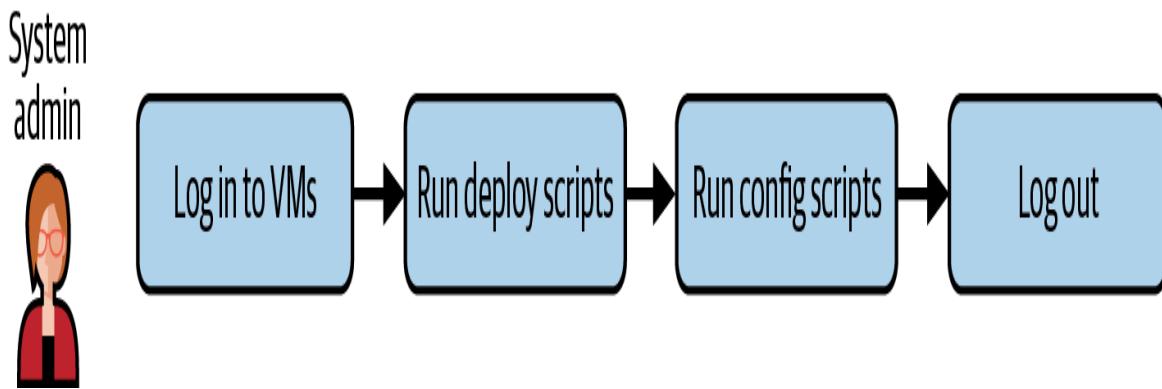


Figura 7-5. Flujo de usuarios del administrador del sistema

Hablemos ahora de los bienes que debemos proteger. He aquí algunos:

1. La información de los pedidos es un activo crítico para la empresa. Los clientes se sentirán insatisfechos si se manipulan sus pedidos, con la consiguiente pérdida de reputación.
2. Los pedidos incluyen datos privados de los clientes, como nombres, números de teléfono, datos de pago y direcciones particulares. Cualquier exposición de información confidencial dará lugar a una demanda judicial y causará perjuicios a los clientes, por lo que los datos de los clientes son otro activo esencial.
3. La base de datos contiene toda la información de ventas de la empresa. Una brecha en ella será peligrosa para los clientes y la empresa, ya que los datos podrían venderse en el mercado negro o a la competencia.
4. También es crucial proteger la infraestructura que aloja la aplicación, ya que cualquier tiempo de inactividad provocará fallos en las transacciones de pedidos y pérdidas de ventas.

Hemos terminado los dos primeros pasos del modelado de amenazas. Lo siguiente es el pensamiento de sombrero negro. Tómate un momento para idear. Recuerda los flujos de usuarios y datos y piensa cómo podrían los hackers hacerse con el control de los activos. Utiliza el modelo STRIDE para estructurar tu pensamiento.

Cuando hayas terminado, compara lo que has obtenido con las posibles amenazas identificadas en [la Figura 7-6](#).

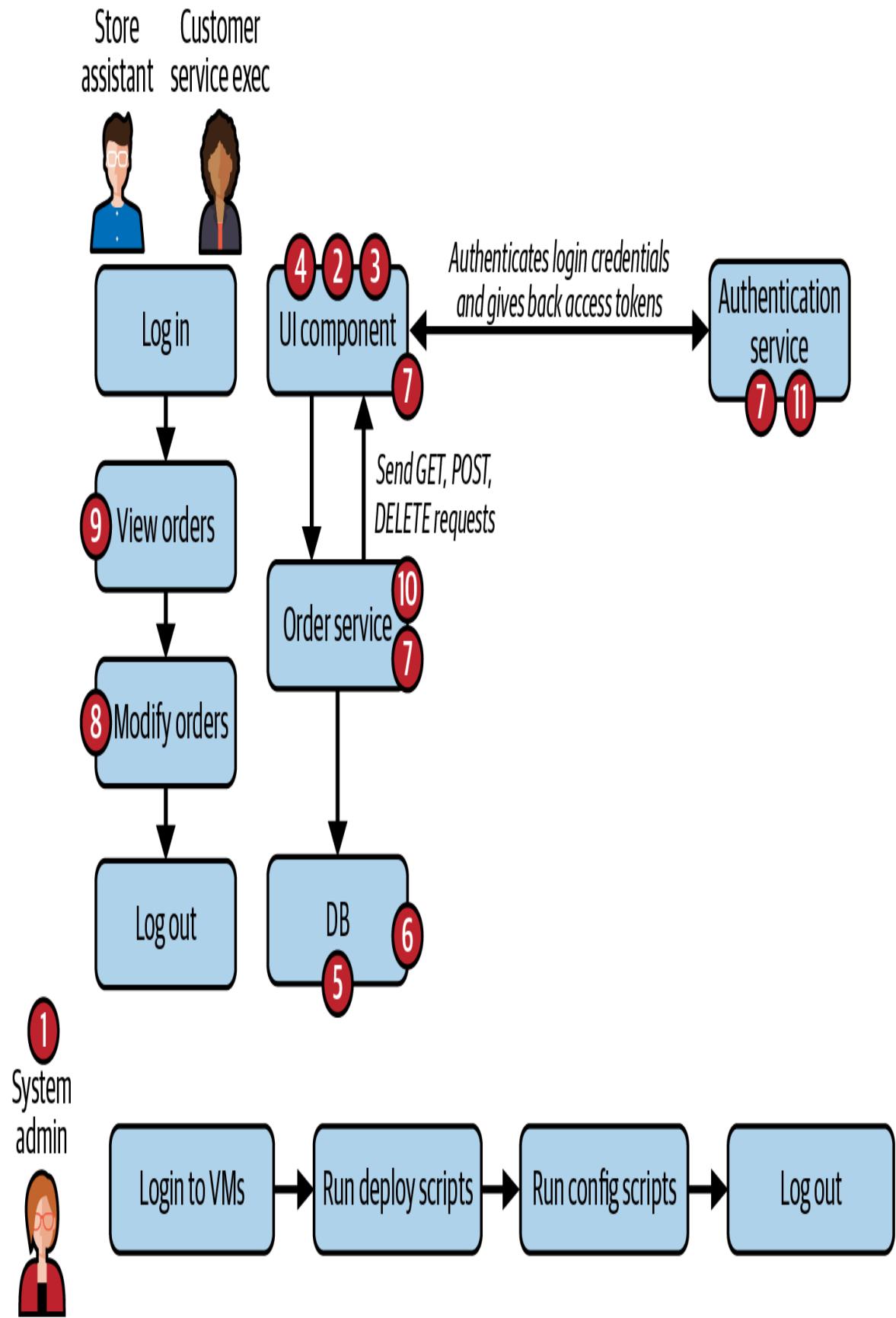


Figura 7-6. Mapa de las amenazas identificadas

Echemos un vistazo a cada uno de ellos:

Identidad suplantada

1. Se podrían utilizar trucos de ingeniería social con el administrador del sistema para obtener sus credenciales de acceso, o simplemente podría bastar con la navegación por el hombro o el malware. Dado que el rol de administrador del sistema tiene superpoderes, un actor malicioso podría utilizarlo para derribar la infraestructura.
2. El dependiente podría olvidarse de cerrar la sesión, y cualquier persona de la tienda podría utilizar la sesión iniciada para cambiar las direcciones de entrega de los pedidos existentes (por ejemplo, a su propia dirección).

Manipulación de entradas

3. El atacante podría hacerse con los puntos finales del servicio de pedidos desde cualquier sesión de navegador abierta y manipular los pedidos posteriormente, ya que los puntos finales pueden no estar protegidos.
4. La inyección de código podría utilizarse al realizar un pedido para secuestrar los datos de pago del cliente.

Repudio de acciones

5. El administrador del sistema, cuando descubra que no hay registros de sus acciones, podría crear pedidos masivos para sus familiares y amigos insertando directamente registros en la base de datos y activando otros procesos relevantes.

Divulgación de información

6. Si la base de datos es atacada a través de una puerta trasera, toda la información que contiene quedará expuesta, ya que los datos se almacenan en texto plano.
7. Robar contraseñas de registros no encriptados u otro tipo de almacenamiento permitiría al atacante manipular los datos de los pedidos.
8. El ejecutivo de atención al cliente no tiene ninguna restricción en sus operaciones: todo lo que debe hacer es transmitir la información sobre el estado actual del pedido a los clientes, pero también podría editar los pedidos. Podrían trabajar con un cómplice para abusar de sus permisos.
9. El endpoint `/viewOrders` permite devolver cualquier número de registros. Una vez comprometido, este endpoint podría utilizarse para ver todos los pedidos. Al menos deberíamos pensar en reducir el radio de explosión.

Denegación de servicio

10. El atacante podría realizar un ataque DDoS y hacer caer el servicio de pedidos, con la consiguiente pérdida de ventas.

Escalada de privilegios

11. Si un atacante consigue hacerse con las credenciales de administrador, podría añadir nuevos usuarios o escalar los privilegios de los usuarios existentes para mantener un nivel elevado de acceso al sistema en el futuro. También podrían crear, modificar o eliminar registros de pedidos sin que nadie se diera cuenta, ya que no hay registros de las acciones del administrador del sistema.

Como puedes ver, incluso en un sistema pequeño con sólo un puñado de componentes y usuarios, hay muchos puntos de ataque.

¡Imagina cuántos habrá en un sistema real con miles de componentes y usuarios!

El siguiente paso es priorizar las amenazas y captar historias. Basándonos en la probabilidad y el impacto de las amenazas identificadas aquí, podríamos añadir nuevas historias de usuarios y abusadores relacionadas con la seguridad, como las siguientes:

1. "Como usuario abusivo, no debería poder ver los datos de los clientes aunque acceda a la base de datos".
2. "Como usuario abusivo, no debería poder aprovecharme de las sesiones abiertas del navegador".
3. "Como usuario abusivo, si consigo acceder a las credenciales de acceso del administrador del sistema o del ejecutivo de atención al cliente, no debería poder editar los pedidos".
4. "Como ayudante de tienda, debería ser la única persona autorizada para hacer solicitudes de edición al servicio de pedidos".
5. "Como dependiente de una tienda, se me debería pedir con frecuencia que cambie mi contraseña por una contraseña segura".

Como ya se ha dicho, para descubrir todas las amenazas potenciales para tu aplicación, debes realizar estos ejercicios de modelado de amenazas de forma iterativa a lo largo de tu ciclo de desarrollo, manteniendo un alcance reducido. Es probable que descubras nuevas amenazas para las funciones más antiguas a medida que vayas incrementando las amenazas para las nuevas funciones.

Casos de prueba de seguridad a partir del modelo de amenazas

A partir del modelo de amenazas, puedes obtener en una visión de las muchas formas en que puede producirse un ataque a tu aplicación. Y junto con las historias de los agresores, obtendrás una

idea vívida sobre los casos de prueba relacionados con la seguridad. Tras el modelado de amenazas, aplica la mentalidad de pruebas exploratorias descrita en [el Capítulo 2](#) para capturar los casos de prueba de seguridad para cada capa de la aplicación.

NOTA

La confianza cero es un principio que sugiere no depositar tu confianza en ninguna entidad, ya sea una persona o un componente, ni siquiera dentro de tu perímetro seguro. Las arquitecturas de confianza cero verifican la autenticidad de cada solicitud antes de ejecutarla, utilizando un protocolo de autenticación y autorización como OAuth 2.0, que utiliza [tokens portadores](#) para verificar la autenticidad. Verás que los casos de prueba hacen referencia a estos tokens.

Suponiendo que se implemente una arquitectura de confianza cero utilizando OAuth 2.0 como solución para hacer frente a las amenazas comentadas anteriormente, he aquí algunos casos de prueba de seguridad en diferentes capas de la aplicación para el ejemplo del sistema de gestión de pedidos:

1. En la capa UI:

- Comprueba que, una vez finalizada la sesión, se pide al usuario que vuelva a conectarse.
- Comprueba que las credenciales de usuario se bloquean tras un número determinado de intentos fallidos de inicio de sesión.
- Comprueba que los campos de entrada tienen validación para entradas ilegítimas (por ejemplo, código JavaScript, consultas SQL, etc.).
- Comprueba que los tokens de acceso caducan al cabo de poco tiempo. Sin embargo, se debe realizar una llamada de

actualización del token desde la interfaz de usuario para mantener al usuario conectado hasta que caduque la sesión.

- Comprueba que cuando inicias sesión como administrador del sistema o ejecutivo de atención al cliente no hay ninguna opción para editar un pedido en la interfaz de usuario.

2. En la capa API:

- Comprueba que la reutilización de un token de acceso caducado hace que el servicio de pedidos devuelva una respuesta 401 No autorizado (aunque es preferible 400 para evitar revelar más información al atacante).
- Comprueba que se realiza una validación adecuada de los valores de los parámetros de la API (similar a la de los campos de entrada de la interfaz de usuario) y que las API devuelven un error 404 si no se superan las comprobaciones de validación.
- Comprueba que el endpoint /editOrder devuelve una respuesta 401 No autorizado si se utiliza el token de acceso de un administrador del sistema o de un ejecutivo de atención al cliente.

3. En la capa BD:

- Comprueba que las contraseñas se almacenan como hashes (con una sal dinámica) en la BD, según **las directrices del NIST**.
- Comprueba que los datos sensibles de los clientes están encriptados en la BD.

4. En los registros de la aplicación:

- Comprueba que las contraseñas no se registran como texto plano en los registros de la aplicación.
- Comprueba que la información sensible para el usuario no se registra como texto plano en los registros de la aplicación.
- Comprueba que existen registros de aplicación adecuados para todas las acciones realizadas en el sistema, incluso por el administrador del sistema, con marcas de tiempo.

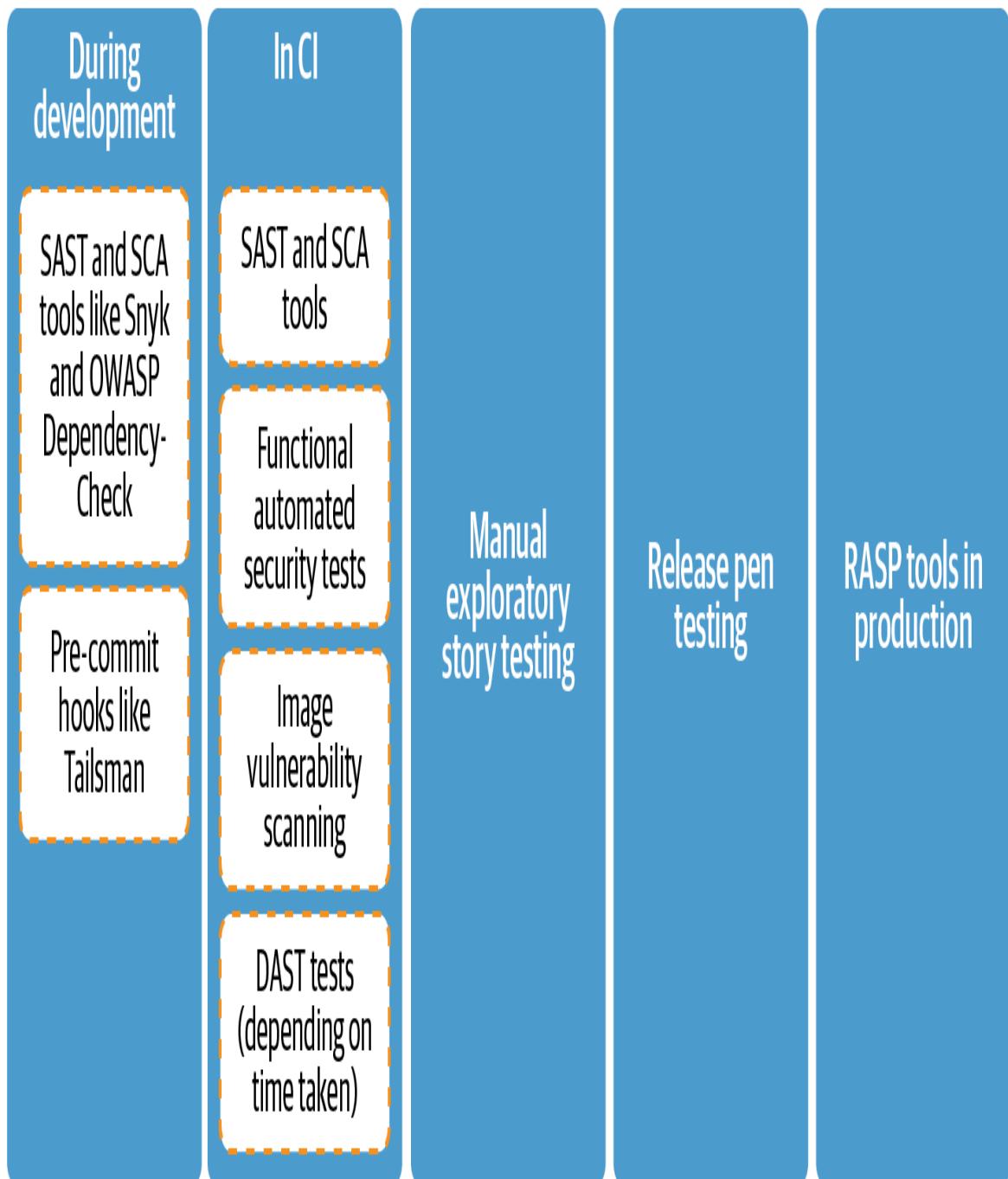
Éstos son sólo algunos casos de prueba relacionados con la seguridad. Es posible que se te ocurran más. Espero que este ejercicio te haya ayudado a comprender cómo los equipos pueden incorporar la seguridad al ciclo de vida del desarrollo de software, desde el modelado de amenazas durante el análisis hasta la concepción de soluciones para posibles amenazas y la comprobación de los aspectos de seguridad.

Estrategia de pruebas de seguridad

Las prácticas que hemos discutido hasta ahora -realizar un ejercicio de modelado de amenazas de 15 minutos por historia de usuario, escribir historias de abusadores, construir medidas de seguridad en capas, derivar casos de prueba relacionados con la seguridad, etc.- ayudarán a reforzar tus sistemas de defensa de forma significativa. Otro paso fundamental es añadir mecanismos que proporcionen información continua sobre posibles vulnerabilidades en el código que se está desarrollando, de modo que puedan solucionarse lo antes posible. Aquí es donde tienes que pensar en desplazar las pruebas de seguridad hacia la izquierda. Aquí hablaremos de una estrategia de pruebas de seguridad que aplica este enfoque.

La Figura 7-7 muestra una estrategia de pruebas de seguridad por turnos a lo largo de las distintas etapas del camino hacia la producción, empezando por el desarrollo.

Shift-left security testing



Path to production

Figura 7-7. Una estrategia de pruebas de seguridad por turnos

Veamos algunas de las herramientas y técnicas que puedes utilizar en las distintas etapas:

Herramientas de pruebas estáticas de seguridad de las aplicaciones (SAST)

SAST es una técnica para analizar el código fuente, el código de bytes y el código ensamblado de la aplicación estática en busca de vulnerabilidades conocidas. Por ejemplo, analiza el código de la aplicación en busca de secretos no cifrados. Las herramientas SAST se presentan en diversas formas, como plug-ins, bibliotecas y soluciones SaaS (por ejemplo, plug-ins Snyk IDE, Checkmarx SAST, Security Code Scan), y pueden integrarse con los procesos CI para ejecutarse en cada confirmación. SAST es una parte importante del cambio a la izquierda, ya que te ayuda a descubrir problemas durante el desarrollo.

Talisman, aunque no es exactamente una herramienta SAST, está diseñada específicamente para escanear el código de la aplicación en busca de secretos como claves privadas, credenciales de entorno, etc., y puede integrarse como gancho previo al envío. Esto evita que los secretos se envíen al repositorio. Echaremos un breve vistazo al complemento Snyk JetBrains IDE y a Talisman más adelante en el capítulo.

Herramientas de Análisis de la Composición del Software (ACS)

SCA es una técnica que identifica vulnerabilidades en las dependencias de terceros de la aplicación. Especialmente cuando utilizas muchas bibliotecas de código abierto, estas herramientas (por ejemplo, OWASP Dependency-Check y Snyk) añadirán mucho valor. También proporcionan información durante el desarrollo y pueden integrarse con CI para cada commit. SCA combinado con SAST ayudará a descubrir las vulnerabilidades del código estático de la aplicación durante la propia fase de

desarrollo. En la siguiente sección se incluye un ejercicio guiado utilizando OWASP Dependency-Check.

Automatización de pruebas de seguridad funcional

Puedes añadir pruebas automatizadas utilizando herramientas de automatización funcional, como se explica en el [Capítulo 3](#), para cubrir los casos de prueba de seguridad funcional. Por ejemplo, en el ejemplo de modelado de amenazas OMS de antes, el caso de prueba de seguridad que verifica que sólo el dependiente de la tienda está autorizado a editar pedidos puede añadirse como prueba de servicio en el servicio de pedidos.

Escaneado de imágenes

Los contenedores se han convertido en una forma ampliamente adoptada de empaquetar e implementar aplicaciones. La comprobación de vulnerabilidades en las imágenes de los contenedores, si los utilizas, es fundamental. Se pueden utilizar herramientas como Snyk Container, Anchore y otras para realizar el escaneado de imágenes, y se pueden integrar con CI. Docker tiene un comando incorporado, `docker scan`, para realizar un escaneo de vulnerabilidades en imágenes Docker. Del mismo modo, Amazon Elastic Container Registry (ECR) ofrece funciones de escaneado de imágenes para las imágenes enviadas al registro. Al escribir infraestructura como código (por ejemplo, con Terraform o Kubernetes), pueden utilizarse herramientas como Snyk IaC y `terraform-compliance` para aplicar también las buenas prácticas de seguridad.

Pruebas dinámicas de seguridad de las aplicaciones (DAST)

DAST es una técnica de pruebas de caja negra . Encuentra problemas de seguridad analizando cómo responde la aplicación a peticiones especialmente diseñadas que imitan ataques reales. Por ejemplo, las herramientas DAST como OWASP ZAP y Burp Suite intentan injectar scripts maliciosos en la aplicación para

comprobar si hay vulnerabilidades de inyección de código. También pueden integrarse en los pipelines de CI, pero dependiendo de la aplicación pueden tardar un poco en ejecutarse, así que elige la fase de CI adecuada en la que ejecutarlas (como se explica en el [Capítulo 4](#)). En la siguiente sección se proporciona un ejercicio detallado que recorre la realización de DAST con OWASP ZAP.

NOTA

[La Prueba Interactiva de Seguridad de Aplicaciones \(IAST\)](#) es una nueva técnica que pretende combinar SAST y DAST para analizar el comportamiento de la aplicación durante el tiempo de ejecución. Funciona mediante la instrumentación del software y escanea en busca de vulnerabilidades de seguridad en tiempo real. Este espacio está evolucionando; algunos ejemplos de herramientas IAST son Contrast Security y Acunetix.

Pruebas exploratorias manuales

Durante las pruebas exploratorias manuales, tú puedes derivar casos de prueba relacionados con la seguridad a partir de ejercicios de modelado de amenazas en todas las capas (interfaz de usuario, servicios, base de datos). Tanto Chrome DevTools como Postman ofrecen diversas opciones para ejecutar casos de prueba de seguridad, como verás más adelante en este capítulo.

Pruebas de penetración (pen)

Dependiendo de la criticidad de la aplicación y de la competencia del equipo de desarrollo con respecto a la seguridad, puedes optar por implicar a un comprobador de seguridad profesional cerca del final del ciclo de entrega, para asegurarte de que la aplicación no sufre problemas de seguridad más adelante.

Autoprotección de aplicaciones en tiempo de ejecución (RASP)

Técnicas como SAST y DAST, comentadas anteriormente, ayudan a encontrar vulnerabilidades en el código de la aplicación . Pero cuando los ataques se producen realmente en el entorno de producción, necesitas una capa de defensa para evitar que tengan éxito. **El RASP** es una técnica de seguridad que consiste en monitorizar la aplicación para detectar posibles ataques en el entorno de producción y evitar que se produzcan. Las herramientas RASP (por ejemplo, Twistlock, Aqua Security) amplían el concepto tradicional de cortafuegos trabajando dentro del tiempo de ejecución de la aplicación y acumulando conocimientos sobre lo que es y lo que no es el comportamiento esperado de la aplicación. A continuación, escuchan los procesos de la aplicación en tiempo de ejecución y adoptan automáticamente medidas de protección, como terminar automáticamente los procesos de criptominado, examinar las cargas útiles de las solicitudes entrantes y denegarlas si son maliciosas, y ayudar a **prevenir los ataques de malware**. Actualmente, las herramientas RASP sólo están disponibles como productos de pago.

Veremos cómo se pueden aplicar en la práctica algunas de estas herramientas en las próximas secciones.

Ejercicios

Los ejercicios aquí te guiarán en la realización de SCA automatizados utilizando OWASP Dependency-Check y DAST utilizando OWASP ZAP, e integrándolos con CI para obtener retroalimentación continua.

Comprobación de dependencia OWASP

Como has visto antes, una amenaza habitual es que tenga dependencias con vulnerabilidades. OWASP Dependency-Check es una herramienta SCA de código abierto que busca vulnerabilidades

conocidas en las bibliotecas y dependencias externas del proyecto. Puede utilizarse a través de la línea de comandos o como complemento de Jenkins o Maven, entre otras opciones.

Configuración y flujo de trabajo

Sigue estos pasos para configurar la herramienta Dependency-Check de la línea de comandos y ejecutar un análisis en el proyecto de prueba de automatización de Selenium WebDriver que creaste en [el Capítulo 3](#):

1. Para instalar Dependency-Check en macOS, utiliza el siguiente comando:

```
$ brew install dependency-check
```

Puedes descargar el archivo ZIP de comprobación de dependencias para otros sistemas operativos desde el [sitio web oficial](#).

2. Una vez instalado, ejecuta el análisis en el proyecto de automatización Selenium WebDriver utilizando el comando que se muestra aquí:

```
// On macOS

$ dependency-check --project project_name -s
project_path --prettyPrint

// On Windows (the dependency-check.bat file is inside the bin
// folder
// when you unzip the folder you downloaded in the previous step)

> dependency-check.bat --project "project_name" --scan
project_path"
```

Este comando puede integrarse en tu canal de CI para que falle si se detectan vulnerabilidades.

3. El comando generará un informe HTML de los resultados del escaneo en la misma carpeta, con una lista de todas las vulnerabilidades encontradas. El proyecto Selenium WebDriver puede tener vulnerabilidades o no. En la **Figura 7-8** se presenta un ejemplo de informe con vulnerabilidades a modo de ilustración.

Summary

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
jquery-1.8.2.min.js		pkg:javascript/jquery@1.8.2.min	MEDIUM	5		3

Published Vulnerabilities



[CVE-2012-6708](#) [suppress](#)

jQuery before 1.9.0 is vulnerable to Cross-site Scripting (XSS) attacks. The `jQuery(stringInput)` function does not differentiate selectors from HTML in a reliable fashion. In vulnerable versions, jQuery determined whether the input was HTML by looking for the '`<`' character anywhere in the string, giving attackers more flexibility when attempting to construct a malicious payload. In fixed versions, jQuery only deems the input to be HTML if it explicitly starts with the '`<`' character, limiting exploitability only to attackers who can control the beginning of a string, which is far less common.

[CWE-79 Improper Neutralization of Input During Web Page Generation \('Cross-site Scripting'\)](#)

Figura 7-8. Resultados del escaneo OWASP Dependency-Check

Como puedes ver, la herramienta ha informado de una vulnerabilidad en la biblioteca jquery-1.8.2. Hace referencia a la vulnerabilidad

publicada con el ID CVE-2012-6708 y explica que "jQuery antes de la versión 1.9.0 es vulnerable a ataques de Cross-site Scripting (XSS)", indicándonos que actualicemos adecuadamente la biblioteca

OWASP ZAP

OWASP Zed Attack Proxy (ZAP) es una herramienta de código abierto que realiza DAST en una aplicación implementada. Utiliza scripts automatizados preconfigurados como ataques a la aplicación para exponer un conjunto conocido de vulnerabilidades. ZAP funciona esencialmente como una herramienta "man-in-the-middle" entre el navegador y la aplicación; olfatea los mensajes intercambiados entre ellos para detectar vulnerabilidades conocidas y las modifica para realizar diversos ataques. Esto facilita que los equipos que se inician en los fundamentos de la seguridad encuentren fácilmente los problemas de seguridad. Además, ZAP dispone de una lista exhaustiva de configuraciones y complementos para soportar múltiples funcionalidades, lo que permite a los profesionales de la seguridad añadir scripts avanzados. Existe [una buena documentación](#) para explorar estas opciones. ZAP puede integrarse con otras herramientas como Selenium, facilitando también su ejecución en CI.

Pongámonos manos a la obra con ZAP.

Configurar

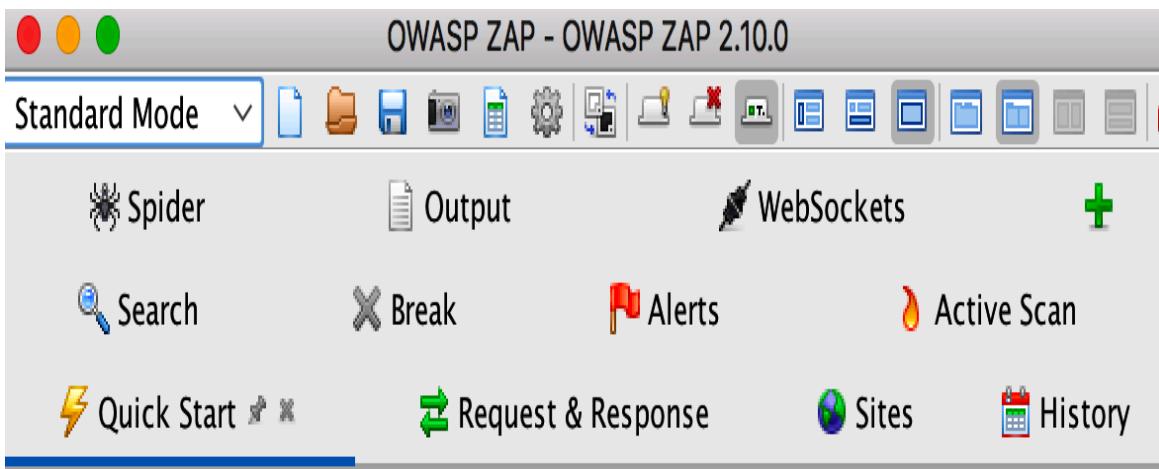
Para instalar ZAP en macOS, utiliza el siguiente comando :

```
$ brew install cask owasp-zap
```

Utiliza los binarios de instalación del [sitio web oficial](#) para otros SO.

Flujo de trabajo

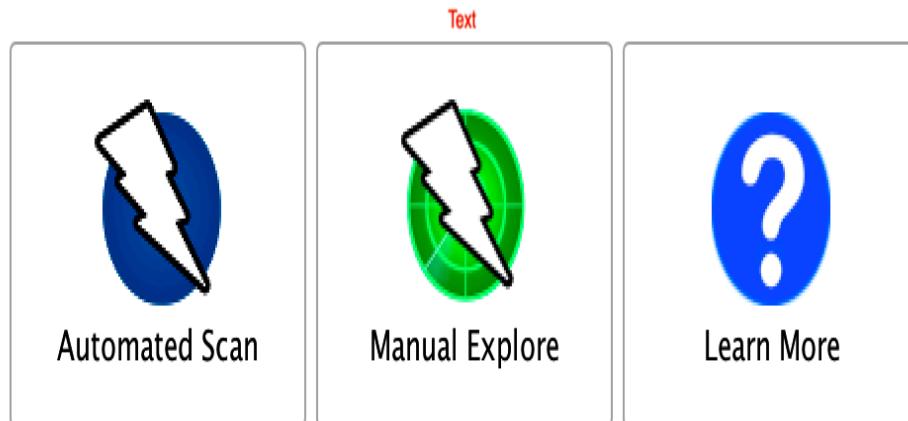
Una vez terminada la instalación, deberías poder abrir la interfaz de usuario de escritorio de ZAP, como se ve en [la Figura 7-9](#). (En un Mac, puede que tengas que dar permiso para abrir la aplicación, ya que no es de la App Store).



Welcome to OWASP ZAP

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

If you are new to ZAP then it is best to start with one of the options below.



News

Announcing the First Ever ZAPCon - Call for Papers

[Learn More](#)

X

Figura 7-9. La interfaz de escritorio de ZAP

El primer paso es educar a ZAP sobre las URL internas de la aplicación y los componentes de la interfaz de usuario para que pueda atacarlos más tarde. Puedes hacerlo de dos formas: utilizando la opción Exploración manual que se puede ver en [la Figura 7-9](#), o mediante utilizando la Araña ZAP.

Explorar manualmente

Para explorar manualmente una aplicación, haz clic en el botón Exploración manual de la interfaz de usuario de ZAP Desktop. Se abrirá una nueva pantalla, como se ve en la [Figura 7-10](#), en la que se te pedirá que introduzcas la URL de la aplicación.

ADVERTENCIA

No utilices esta herramienta en un sitio web público. Es ilegal realizar pruebas de seguridad en un sitio web sin autorización.

The screenshot shows the ZAP interface with the following elements:

- Top Bar:** Quick Start, Request & Response, Sites, History.
- Main Title:** Manual Explore
- Description:** This screen allows you to launch the browser of your choice so that you can explore your application while proxying through ZAP.
- Input Fields:**
 - URL to explore:
 - Enable HUD:
 - Explore your application:
- Note:** You can also use browsers that you don't launch from ZAP, but will need to configure them to proxy through ZAP and to import the ZAP root CA certificate.

Figura 7-10. Opción Exploración manual de ZAP

OWASP nos proporciona un sitio web de conejillos de indias, la [Tienda de Zumos OWASP](#), que podemos utilizar para aprender sobre las pruebas de seguridad. Yo he utilizado esta URL. ZAP nos permite explorar utilizando tanto Firefox como Chrome; elige uno. Una vez que el navegador abra la aplicación, recorre manualmente el flujo de usuario una vez. ZAP explorará la aplicación en segundo plano y registrará los detalles relevantes.

Fíjate en la casilla Activar HUD de la [Figura 7-10](#). Un Heads Up Display (HUD) es una superposición del navegador que se muestra encima de tu aplicación. Se trata de una disposición de ZAP para evitar la necesidad de alternar entre la interfaz de usuario de escritorio de ZAP y el navegador durante los ataques. Si lo has activado, verás el HUD encima del sitio web de Juice Shop, como en la [Figura 7-11](#) (los paneles de la izquierda y la derecha).



All Products

Out

Off

0

0

1

2

0

0

+



Apple Juice
(1000ml)

1.99¤

Sites

Start

Start

Start

Off

1

3

6

3

1

This website uses fruit cookies to ensure you get
the juiciest tracking experience. But me wait!

Me want it!

History 51

WebSockets 9



Figura 7-11. El HUD del sitio web de Juice Shop

Cuando explores el sitio manualmente, verás que el ícono Sitios del panel derecho recoge el árbol de sitios, y la pestaña Historial de la parte inferior enumera las URL visitadas. ZAP utilizará estos detalles más adelante, cuando ataque activamente la aplicación.

Araña ZAP

El ZAP Spider elimina la carga de explorar manualmente el sitio web . Rastrea automáticamente el sitio utilizando Selenium WebDriver y recopila todas las URL de la aplicación y los componentes de la interfaz de usuario. Es posible que la Araña simple (el ícono gris debajo de Sitios a la derecha) no pueda explorar componentes JavaScript, pero la **Araña AJAX** (el ícono rojo) sí. Puedes utilizar ambas para explorar completamente la aplicación.

Escaneando

Mientras la araña ZAP navega por la aplicación recopilando información, realiza un escaneo pasivo en segundo plano. ZAP también realiza otro tipo de escaneo llamado escaneo activo, ¡en el que ataca a la aplicación! Pruébalos ahora:

- El rastreo pasivo consiste en leer los mensajes intercambiados entre el navegador y la aplicación web e inspeccionarlos en busca de vulnerabilidades: los mensajes no se modifican (es decir, no se atacan). Este escaneo se realiza automáticamente, por lo que verás que se muestran alertas en los paneles derecho e izquierdo cuando la Araña ZAP está rastreando. Las alertas se priorizan en función de su gravedad (alta, media y baja) y se agrupan bajo banderas rojas, naranjas y amarillas, como se ve en la **Figura 7-11**. Al hacer clic en esas banderas, puedes ver los detalles de las respectivas vulnerabilidades. Además, la interfaz de usuario de ZAP Desktop tendrá registros detallados. Por ejemplo, un escaneo pasivo encontró una dirección IP privada

expuesta en el sitio web de Juice Shop, como se ve en la **Figura 7-12**.

Alerts Active Scan Spider Output WebSockets +

Alerts (24)

- > Example High-Level Notification
- > SQL Injection (2)
- > CSP: Wildcard Directive (38)
- > Cross-Domain Misconfiguration (1)
- > Example Medium-Level Notification
- > Session ID in URL Rewrite (200)
- > Vulnerable JS Library
- > X-Frame-Options Header Not Set
- > Absence of Anti-CSRF Tokens (6)
- > CSP: Notices (38)
- > Cookie Without Secure Flag (200)
- > Cross Site Scripting Weakness (Reflected) (1)
- > Cross-Domain JavaScript Source File (1)
- > Example Low-Level Notification
- > HUD Tutorial Site Alert
- > Incomplete or No Cache-control header (1)
- > Private IP Disclosure
- > X-Content-Type-Options Header
- > Base64 Disclosure in WebSocket message (1)
- > Example Informational Alert Notification
- > HUD Tutorial Page Alert
- > Information Disclosure - Sensitive (1)
- > Information Disclosure - Suspicious (1)
- > Timestamp Disclosure - Unix (78)

Private IP Disclosure

URL: <https://juice-shop.herokuapp.com/rest/admin/application-configuration>

Risk: Low

Confidence: Medium

Parameter:

Attack:

Evidence: 192.168.99.100:3000

CWE ID: 200

WASC ID: 13

Source: Passive (2 - Private IP Disclosure)

Description:

A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks.

Other Info:

192.168.99.100:3000

Solution:

Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.

Reference:

<https://tools.ietf.org/html/rfc1918>

Figura 7-12. Resultados del escaneo pasivo en la interfaz de usuario de ZAP Desktop

- Durante el escaneo activo, ZAP ataca la aplicación interceptando las peticiones, modificándolas y haciéndolas rebotar de un lado a otro, comprobando vulnerabilidades conocidas como la inyección SQL y muchas más. Haz clic en el cuarto ícono desde arriba en el panel lateral derecho del HUD (debajo de la araña roja) para activar un escaneo activo. Inmediatamente verás cómo ZAP navega por el sitio web página a página, imitando distintos ataques. El escaneo tarda un rato; una vez completado, puedes ver las vulnerabilidades encontradas bajo las banderas de colores. [La Figura 7-13](#) muestra una vulnerabilidad de inyección SQL encontrada en el sitio web Juice Shop.

Alerts Active Scan Spider Output WebSockets +

SQL Injection

URL: <https://juice-shop.herokuapp.com/rest/user/login>

Risk: High

Confidence: Medium

Parameter: email

Attack: EPWKFdlhjKluYwhAcKKpzlRq' OR '1='1' --

Evidence:

CWE ID: 89

WASC ID: 19

Source: Active (40018 - SQL Injection)

Description:
SQL injection may be possible.

Other Info:
The page results were successfully manipulated using the boolean conditions
[EPWKFdlhjKluYwhAcKKpzlRq' AND '1='1' --] and [EPWKFdlhjKluYwhAcKKpzlRq' OR
'1='1' --]

Solution:
Do not trust client side input, even if there is client side validation in place.
In general, type check all data on the server side.
If the application uses JDBC, use PreparedStatement or CallableStatement, with

Reference:
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

- Alerts
- Active Scan
- Spider
- Output
- WebSockets
- +

Alerts (24)

- > Example High-Level
- > SQL Injection (2)
- > CSP: Wildcard Directive
- > Cross-Domain Misconfig
- > Example Medium-Level
- > Session ID in URL Rewrite
- > Vulnerable JS Library
- > X-Frame-Options Header
- > Absence of Anti-CSRF
- > CSP: Notices (38)
- > Cookie Without Secure
- > Cross Site Scripting Vulnerability
- > Cross-Domain JavaScript
- > Example Low-Level Mutation
- > HUD Tutorial Site Alerts
- > Incomplete or No Cache Headers
- > Private IP Disclosure
- > X-Content-Type-Options
- > Base64 Disclosure in Headers
- > Example Information Disclosure
- > HUD Tutorial Page Alerts
- > Information Disclosure
- > Information Disclosure
- > Timestamp Disclosure

Figura 7-13. Una vulnerabilidad de inyección SQL encontrada durante una exploración activa

Así de sencillo hace ZAP el DAST para todos los equipos de software: basta con abrir la aplicación en la interfaz de usuario de ZAP, utilizar las arañas para realizar un escaneado pasivo, y activar un escaneado activo!

Integrar ZAP con CI

Ahora que ZAP te ha proporcionado una lista de todas las vulnerabilidades de la aplicación, tienes que analizarlas y decidir cuáles corregir. Esto requiere mucho tiempo y experiencia. Es aconsejable hacer este ejercicio continuamente en lugar de dejar que este trabajo se acumule al final: ¿qué mejor opción que integrarse con CI para obtener retroalimentación continua?

NOTA

Como se ha mencionado anteriormente, el análisis activo puede tardar mucho tiempo en completarse, dependiendo de tu aplicación (a veces incluso horas). En ese caso, puedes optar por integrarlo con CI como una etapa de regresión nocturna o incluir un activador manual que puedas utilizar para cada historia de usuario.

Para integrarse con CI, ZAP proporciona API como las siguientes:

- `zap.urlopen(target)` para abrir la aplicación
- `zap.spider.scan(target)` para activar la Araña ZAP y realizar una exploración pasiva
- `zap.ascan.scan(target)` para activar una exploración activa
- `zap.core.alerts()` para imprimir los resultados encontrados

Puedes utilizar estas API en un simple script JavaScript o Python para realizar los escaneos e integrarte con CI utilizando la **CLI** de ZAP.

Alternativamente, puedes incrustar las API de ZAP dentro de tus pruebas funcionales de Selenium WebDriver y ejecutarlas como las pruebas funcionales típicas en la canalización CI. WebDriver también puede ayudar a ZAP a iniciar sesión en el sitio web, lo que podría no ser capaz de hacer por sí solo. **El Ejemplo 7-3** muestra una prueba de ejemplo que escanea la aplicación y falla si hay vulnerabilidades. Ten en cuenta que debes añadir las esperas adecuadas, en función del tiempo de escaneado de tu aplicación.

Ejemplo 7-3. Escaneo ZAP como parte de las pruebas Selenium

```
@Test
public void testSecurityVulnerabilities() throws Exception {

    zapScanner = new ZAPProxyScanner(ZAP_PROXYHOST, ZAP_PROXYPORT,
ZAP_APIKEY);
    login.loginAsUser();

    // Step 1- Spider the app using ZAP API
    zapSpider.spider(BASE_URL)

    // Step 2 - Enable passive scanning
    zapScanner.setEnablePassiveScan(true);

    // Step 3 -Start Active scan. Add wait methods.
    zapScanner.scan(BASE_URL);

    // Step 4 - Log the alerts and assert the count of alerts
    List<Alert> alerts = filterAlerts(zapScanner.getAlerts());
    logAlerts(alerts);
    assertThat(alerts.size(), equalTo(0));
}
```

ZAP produce informes HTML de las vulnerabilidades, como se ve en la **Figura 7-14**, que pueden guardarse como artefactos de salida en CI.

High (Medium)	SQL Injection
Description	SQL injection may be possible.
URL	https://juice-shop.herokuapp.com/rest/user/login
Method	POST
Parameter	email
Attack	EPWKFdhjKluYwhAcKKpzlRq' OR '1='1' --
URL	https://juice-shop.herokuapp.com/rest/user/login
Method	POST
Parameter	email
Attack	VqqxCXFFxHhqC1xYYvCGioKa' OR '1='1' --
Instances	2
	<p>Do not trust client side input, even if there is client side validation in place.</p> <p>In general, type check all data on the server side.</p> <p>If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'.</p>

Figura 7-14. Informe ZAP HTML

Por último, si utilizas Acciones de GitHub para tu CI/CD, tienes una opción sencilla para integrar ZAP utilizando las Acciones predefinidas **OWASP ZAP Baseline Scan** y **OWASP ZAP Full Scan**. Como su nombre

indica, realizan el escaneo ZAP y también añaden las incidencias a GitHub.

Además de las que aquí se comentan, ZAP tiene muchas otras funciones útiles que permiten realizar diversos tipos de pruebas exploratorias de seguridad en la aplicación. Aquí se enumeran algunas:

- ZAP puede utilizar las especificaciones OpenAPI para realizar pruebas de seguridad en las API.
- Tiene una función llamada Breaks que te ayudará a insertar datos de prueba específicos en una solicitud y observar el comportamiento. Por ejemplo, para comprobar si la API valida los parámetros de entrada para la inyección SQL, puedes utilizar la función Romper.
- Permite reproducir una petición en el navegador.
- Existe una opción para resaltar determinadas palabras clave ocultas en el HTML.
- Tiene una función para revelar todos los campos de entrada ocultos de la aplicación.
- Hay complementos con guiones preescritos elaborados por expertos que pueden utilizarse para reproducir tipos específicos de ataques si es necesario.

En general, ZAP es una herramienta excelente para probar y puede enseñarte muchas cosas sobre seguridad.

Herramientas de comprobación adicionales

Aquí se comentan algunas herramientas más que ayudan en las pruebas de seguridad exploratorias manuales y SAST, para dar una perspectiva más amplia de las herramientas relacionadas con la

seguridad que pueden adoptarse durante el ciclo de entrega del software.

Complemento IDE de Snyk

El complemento Snyk JetBrains IDE combina las capacidades SCA y SAST. Es totalmente gratuito y puede utilizarse con cualquier de los IDE de JetBrains (por ejemplo, IntelliJ IDEA, WebStorm, PyCharm). La mayor ventaja es que está muy cerca del desarrollo y un destacado desplazamiento a la izquierda. Puedes activar escaneos para buscar vulnerabilidades tanto en el código de la aplicación como en sus dependencias mientras se desarrolla el código. La Figura 7-15 muestra los resultados de un escaneo de este tipo en el panel inferior del IDE IntelliJ. Puedes ver que Snyk ha resaltado una vulnerabilidad de "Revelación de información" en el código de la aplicación. También muestra opciones de corrección para solucionar las vulnerabilidades que encuentra, lo que facilita a los desarrolladores la incorporación de la seguridad.

The screenshot shows the Snyk IDE plugin interface. On the left, a tree view displays a scan for issue types (Severity: Critical, High, Medium, Low) across various projects and files. A specific vulnerability for 'org.codehaus.groovy:groovy@3.0.3: Information Disclosure' is highlighted in blue. The right side provides detailed information about this vulnerability, including its severity (High), module (org.codehaus.groovy:groovy), introduction through (io.rest-assured:rest-assured@4.3.1), fix (org.codehaus.groovy:groovy@2.4.21, @2.5.14, @3.0.7, @4.0.0-alpha-2), and exploit maturity (Not Defined). It also includes a 'Detailed paths' section and a 'Remediation' section suggesting an upgrade.

Vulnerability	Severity	Module	Introduced through	Fixed in	Exploit maturity
org.codehaus.groovy:groovy@3.0.3: Information Disclosure	High	org.codehaus.groovy:groovy	io.rest-assured:rest-assured@4.3.1	org.codehaus.groovy:groovy@2.4.21, @2.5.14, @3.0.7, @4.0.0-alpha-2	Not Defined

Figura 7-15. Ejemplo de resultados del escaneado del complemento IDE de Snyk

Snyk también viene como opción CLI, pero sólo con capacidades SCA para integrarse con CI. La empresa también ofrece un conjunto de otros servicios relacionados con la seguridad como opciones de pago.

Gancho de precompromiso Talismán

Talisman, una herramienta de código abierto, escanea el código de tu aplicación en busca de secretos e información sensible (como contraseñas, claves SSH, tokens, etc.) cuando haces commit en el sistema de control de versiones y emite alertas cuando los encuentra. Esto es muy útil para evitar que los equipos de desarrollo envíen secretos por accidente. Puedes configurarlo como gancho precommit o como gancho pre-push. El Ejemplo 7-4 muestra un ejemplo de resultado de escaneo al intentar confirmar código utilizando Git.

Ejemplo 7-4. Muestra de resultados de la exploración Talisman

```
$ git commit
Talisman Report:
+-----+
|     FILE      |          ERRORS          |
+-----+
| sampleCode.pem | The filename "sampleCode.pem"
|               | failed checks against the
|               | pattern ^.+\.pem$|
+-----+
| sampleCode.pem | Expected file not to contain hex-encoded texts such as:
|               | awsSecretKey=
|               | c99e0c79ddcf5ddb02f1274db2d973f363f4f553ab1692d8d203b4cc09692f79|
+-----+
```

Talisman ha identificado la presencia de awsSecretKey en el código de la aplicación. Con los bots rastreando los repositorios de GitHub en busca de secretos, como ya hemos comentado, ésta es una herramienta importante que debería añadir a su repertorio.

Chrome DevTools y Postman

Para realizar pruebas exploratorias manuales de seguridad en torno a casos de uso funcionales, como los distintos casos de prueba resultantes de un ejercicio de modelado de amenazas, Chrome DevTools y Postman son bastante prácticos. En el Capítulo 2 hablamos extensamente de las funciones de Postman. Postman también te permite realizar pruebas exploratorias relacionadas con la seguridad, como configurar tokens de autenticación como parte de

las solicitudes API, como se ve en la [Figura 7-16](#). Puedes utilizar esta función para probar escenarios como tokens de acceso manipulados y caducados.

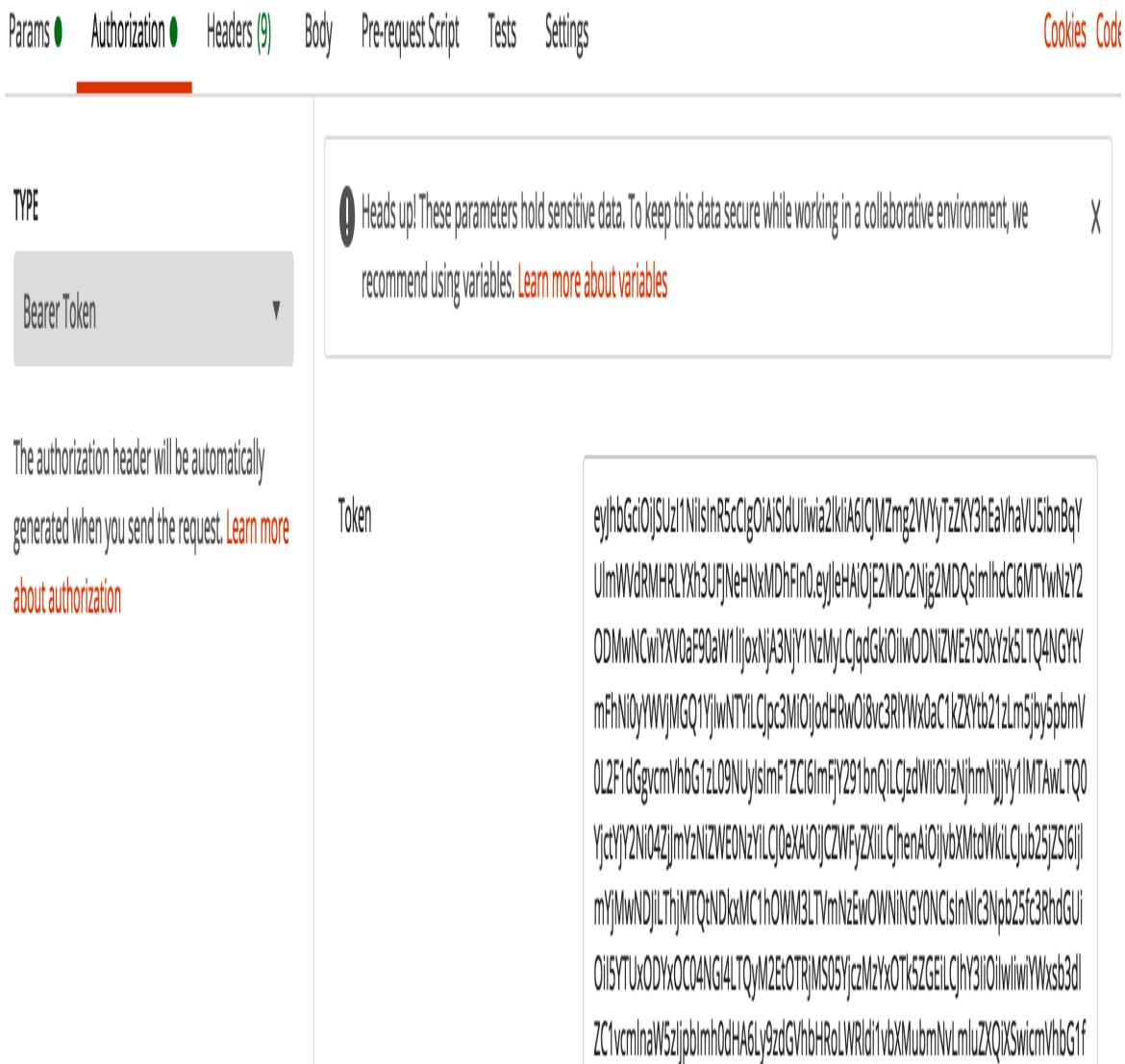


Figura 7-16. Configuración del token de acceso Postman

Del mismo modo, la pestaña Seguridad de Chrome DevTools, como se ve en la [Figura 7-17](#), te indica si la página se sirve correctamente a través de HTTPS o no. También resalta cuando los recursos de sitios de terceros no se sirven de forma segura, ya que esto puede conducir potencialmente a ataques man-in-the-middle.

The screenshot shows the Chrome DevTools interface with the "Security" tab selected. On the left, a sidebar lists network origins categorized into three groups: "Main origin (secure)", "Secure origins", and "Unknown / canceled". Each group contains a list of URLs with their respective SSL icons. On the right, the main panel displays the "Origin" section for the main secure origin, which includes a link to "View requests in Network Panel". Below this, a message states "No security information" and "No security details are available for this origin".

Category	Origin
Main origin (secure)	🔒 https://munchkin.marketo.net View requests in Network Panel
Secure origins	HTTPS https://www.thoughtworks.com HTTPS https://cdn.cookie-law.org HTTPS https://static.thoughtworks.com HTTPS chrome-extension://cjpalhdlnbpafiamejdnhcphjbkeiagm HTTPS https://dynamic.thoughtworks.com HTTPS https://fast.wistia.net
Unknown / canceled	?

Origin

?

<https://munchkin.marketo.net>

[View requests in Network Panel](#)

No security information

No security details are available for this origin.

Figura 7-17. La pestaña Seguridad en Chrome DevTools

Con las herramientas de pruebas de seguridad que hemos explorado aquí, ahora debería estar claro que las pruebas de seguridad pueden integrarse en el ciclo de desarrollo del software y no tienen por qué ser responsabilidad exclusiva de expertos en pruebas de penetración. Las pruebas de seguridad por turnos ayudarán a evitar que tu equipo se enfrente más adelante a graves problemas de seguridad.

Perspectivas: La seguridad es un hábito

Una de mis observaciones basadas en la experiencia es que, por mucho esfuerzo que dediquemos a las diversas actividades tratadas en este capítulo para intentar crear software seguro, a menos que convirtamos la seguridad en un hábito, podríamos dejar eslabones débiles inesperados en nuestras aplicaciones que podrían ser explotados. De hecho, varias prácticas habituales en los equipos de software podrían conducir fácilmente a un compromiso de la seguridad. Por ejemplo, ¿has pensado en los aspectos de seguridad de las herramientas que utilizas para ayudarte en el desarrollo y las pruebas? ¿Alguno de ellos almacena los datos del proyecto en la nube privada del proveedor? ¿Dejas el diagrama de arquitectura del proyecto con los detalles del entorno en un portal online? ¿Compartiste las credenciales del sistema de producción con todos los miembros de tu equipo? Si es así, ¿las compartiste en Slack en texto plano? Muchos de estos actos simples podrían fácilmente resultar en un compromiso.

Por tanto, hacer de la seguridad un hábito es la única forma de avanzar. Es igual que cuando miramos la comida antes de comerla o nos damos cuenta de que alguien nos sigue. Hacemos esas comprobaciones de forma inconsciente y natural. Del mismo modo, nosotros, como equipos de software, debemos entrenarnos para hacer de la seguridad un hábito inconsciente y natural. Podemos empezar por preguntarnos a diario si alguno de los actos sencillos,

involuntarios e inofensivos que realizamos podría dar lugar a una brecha de seguridad.

Puntos clave

Éstos son los puntos clave de este capítulo:

- Los ciberdelitos en la era digital son más frecuentes de lo que crees, y se espera que los ingresos anuales superen los 10 billones de dólares en pocos años.
- Ejemplos reales de ataques demuestran que se piratean todo tipo de plataformas digitales con la intención de robar dinero, datos privados, infraestructuras, etc. Así que, claramente, la seguridad ya no es sólo una característica "agradable de tener".
- Las medidas de seguridad deben incorporarse a la aplicación a lo largo de todo el ciclo de vida de desarrollo del software, desde el análisis hasta las pruebas, para lograr el objetivo de construir sistemas fuertes e impenetrables.
- El modelo STRIDE proporciona una lente estructurada a través de la cual explorar las amenazas a la seguridad de una aplicación, que puede aplicarse para realizar el modelado de amenazas.
- Los ejercicios de modelado de amenazas deben hacerse con frecuencia con todo el equipo para pequeñas partes de la funcionalidad de la aplicación, como una historia de usuario o una característica. El modelado de amenazas debe dar lugar a la creación de historias de usuario y casos de prueba relacionados con la seguridad.
- Implementación de estrategias de pruebas de seguridad por turnos utilizando diversos tipos de herramientas de pruebas de seguridad automatizadas (SAST, SCA, DAST, etc.) , así como pruebas manuales exploratorias y funcionales automatizadas.

- Con la disponibilidad de herramientas accesibles de pruebas de seguridad automatizadas, los equipos de software no tienen que esperar a las pruebas de penetración como única forma de obtener información sobre los problemas de seguridad.
- Y lo más importante, haz de la seguridad un hábito.

-
- 1 Para una excelente introducción a los principios y buenas prácticas para escribir software seguro, véase *Secure by Design* (Manning), de Daniel Deogun, Dan Bergh Johnsson y Daniel Sawano.
 - 2 Para más información sobre este tema, véase *Introducción a la Criptografía con Teoría de la Codificación*, 3^a Edición (Pearson), de Wade Trappe y Lawrence C. Washington.

Capítulo 8. Pruebas de rendimiento

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

¡El tiempo es oro!

-Benjamin Franklin

Todos lo hemos experimentado: a veces nuestros sitios web favoritos se vuelven de repente tan lentos como perezosos, dejándonos preguntándonos: "¿Hay algún problema con mi Internet?". ¿Recuerdas haber esperado una eternidad durante las rebajas del Cyber Monday a que se cargara un sitio web? ¿O mirar fijamente el icono de carga esperando a que aparezcan los billetes de tren cuando te mueres por reservar tus vacaciones de Navidad? ¿O quedarte colgado en la página de reserva de una película taquillera? Un rendimiento deficiente del sitio web en casos como estos puede hacer que, como clientes, sintamos una intensa frustración.

Si quieras salvar a los usuarios finales de tu aplicación de tales frustraciones, tienes que medir continuamente y trabajar para mejorar su rendimiento. Este capítulo pretende equiparte con los elementos esenciales que necesitas para medir o probar el rendimiento web; en concreto, trataremos temas como los KPI de rendimiento, las pruebas de rendimiento de la API, las pruebas de rendimiento del frontend y las pruebas de rendimiento de shift-left. También tendrás la oportunidad de probar de forma práctica las

pruebas de rendimiento del frontend y de la API como parte de los ejercicios de este capítulo.

Como las pruebas de rendimiento son un tema tan amplio y deben realizarse tanto en el backend como en el frontend, este capítulo está estructurado de forma un poco diferente a los anteriores. Aquí encontrarás todas las secciones conocidas, pero divididas en dos mitades. Empezaremos cubriendo todo lo que necesitas saber para ponerte al día con las pruebas de rendimiento del backend, incluyendo ejercicios y herramientas adicionales. Una vez hecho esto, nos centraremos en las pruebas de rendimiento del frontend. Hacia el final del capítulo se presenta una estrategia global para las pruebas de rendimiento del shift-left.

Bloques de construcción de las pruebas de rendimiento del backend

Veamos primero por qué el rendimiento es tan crítico para el éxito de una empresa. Aquí exploraremos los factores que afectan al rendimiento de una aplicación, los indicadores clave del rendimiento de una aplicación web y las formas de medirlos.

El rendimiento, las ventas y los fines de semana libres ¡están correlacionados!

Al principio del capítulo hemos hablado de que los clientes se sienten frustrados por el rendimiento de las aplicaciones en . Tenemos que entender a qué puede llevar eso. ¿Cuánto pueden importar realmente unos segundos de retraso? De hecho, existe un indicador cuantitativo que proporciona una medida del efecto de los tiempos de carga de la página en el comportamiento del cliente . La *tasa de rebote* es una medida del porcentaje de clientes que abandonan un sitio web tras ver una sola página.

Entre todos los factores potenciales que pueden aumentar la tasa de rebote, se ha demostrado que el rendimiento del sitio web es el que más contribuye. Las estadísticas publicadas por Google (ver Tabla 8-1) muestran la correlación entre el tiempo de carga de la página y la tasa de rebote de los usuarios. Los datos confirman que con cada segundo adicional de retraso, las empresas pierden clientes a favor de sus competidores.

Tabla 8-1. Estadísticas de Google que correlacionan el tiempo de carga de la página y las tasas de rebote

Tiempo de carga de la página	Aumenta la probabilidad de rebote en
1-3 segundos	32%
1-5 segundos	90%
1-6 segundos	106%
1-10 segundos	123%

Y aún hay más: los algoritmos de optimización de motores de búsqueda (SEO) de Google clasifican peor los sitios web más lentos, lo que significa que si tu sitio web no rinde lo suficiente, se hundirá aún más en el abismo! Para su propio sitio web, Google aspira a un tiempo de carga **inferior a medio segundo**, y recomienda 2 segundos como máximo para un rendimiento aceptable del sitio web.

Perder clientes se traduce en perder ventas, y las empresas pueden pagar un precio muy alto por los fallos de rendimiento. Por ejemplo, en 2018, Amazon se enfrentó a una pérdida estimada de **entre 72 y 99 millones de dólares** cuando su sitio web no pudo gestionar el tráfico de su evento Prime Day. Un rendimiento pésimo también puede conducir a una pérdida de reputación de la marca, especialmente en un mundo en el que, gracias a las redes sociales, las malas críticas pueden propagarse con tanta rapidez.

Por otro lado, un ligero aumento del rendimiento puede dar lugar a mejoras significativas en las ventas. Por ejemplo, en 2016, **Trainline**,

una empresa operadora de trenes del Reino Unido, redujo el tiempo medio de carga de su página en 0,3 s, y los ingresos aumentaron en 8 millones de libras (11 millones de dólares) al año. Del mismo modo, el proveedor de frontend como servicio **Mobify** observó que cada 100 ms de reducción del tiempo de carga de su página de inicio aumentaba las conversiones a un ritmo que se traducía en un aumento de los ingresos anuales de 380.000 \$. La correlación entre ventas y rendimiento deja claro que el primer paso para mejorar las ventas de un negocio online es examinar el rendimiento de su aplicación. Esto significa que nosotros, como equipos de software, tenemos que construir y probar el rendimiento pronto y con frecuencia, es decir, desplazar nuestras pruebas de rendimiento hacia la izquierda.

Una de mis principales motivaciones para centrarme pronto en el rendimiento del sitio web es sencilla: Me encantan mis fines de semana y quiero pasarlo relajado. Dado que los problemas de rendimiento pueden ser muy costosos, como habrás observado en los ejemplos anteriores, y afectar directamente a la reputación de la marca, cuando se producen en producción los equipos de desarrollo de software suelen verse sometidos a una gran presión para solucionarlos cuanto antes. Así que, si no incorporas las pruebas de rendimiento pronto y con frecuencia durante el ciclo de vida de desarrollo del software, ipuedes esperar pagarla más tarde trabajando los fines de semana (y muchas horas) para solucionar los problemas de rendimiento que surjan en !

Objetivos de rendimiento sencillos

El rendimiento, en términos sencillos, puede considerarse como la capacidad de una aplicación para servir a un gran número de usuarios simultáneos sin una degradación significativa de su comportamiento en comparación con cuando sirve a un solo usuario. Es decir, el rendimiento no debe degradarse más allá de un punto que sea aceptable para los usuarios finales. Por tanto, para

comprobar el rendimiento, primero tienes que determinar el número previsto de usuarios en hora punta para tu aplicación, y luego comprobar que el rendimiento de la aplicación bajo ese nivel de carga sigue siendo aceptable.

Lo que constituye un rendimiento aceptable depende en gran medida de los límites de la percepción humana. Según [el](#) investigador de usabilidad web e interacción persona-ordenador [Jakob Nielsen](#), cuando el tiempo de respuesta del sitio web es inferior a 0,1 segundos aproximadamente, el usuario siente que el comportamiento es instantáneo. Con tiempos de respuesta de 0,2 a 1 segundo, perciben el retraso pero siguen sintiendo que controlan la navegación por el sitio web. Más allá de esto, sienten que la interfaz de usuario es lenta y pierden la sensación de fluidez al realizar la tarea deseada. Como hemos visto antes, la investigación de Google muestra que con retrasos superiores a 3 segundos corres el riesgo de perder a la mayoría de tus clientes, y recomiendan mantener el tiempo de carga de la página en menos de 2 segundos.

Éstos son tus objetivos de rendimiento. Para conseguir tan buenos resultados, es necesario realizar muchos ajustes en la infraestructura y optimizar el código en muchas iteraciones antes de que tu aplicación entre en funcionamiento, otra razón más para adoptar una estrategia de pruebas de rendimiento por turnos!

Factores que afectan al rendimiento de la aplicación

Alcanzar los objetivos de rendimiento establecidos en la sección anterior no es tan sencillo: si lo fuera, las empresas no habrían perdido tanto dinero por problemas de rendimiento. Hay muchos factores en una aplicación que afectan al rendimiento, incluidos los enumerados aquí:

Diseño arquitectónico

El diseño de la arquitectura desempeña un papel vital en el rendimiento de un sitio web. Por ejemplo, cuando las responsabilidades de los servicios web no están debidamente compartimentadas, habrá que realizar numerosas llamadas a diferentes servicios desde la interfaz de usuario, lo que retrasará el tiempo de respuesta. Del mismo modo, cuando no se implementan mecanismos de almacenamiento en caché adecuados en los niveles correctos, el rendimiento del sitio web se verá afectado.

Elección de la pila tecnológica

Las diferentes capas de la aplicación necesitan diferentes conjuntos de herramientas. Estas herramientas pueden no trabajar juntas de forma coherente, afectando al rendimiento general. Por poner sólo un ejemplo de los tipos de interacciones que debes tener en cuenta, la elección del lenguaje (por ejemplo, Java, Ruby, Go, Python) puede tener un impacto perceptible en **el tiempo de inicio en frío de AWS Lambda**.

Complejidad del código

El código complejo o mal escrito (piensa en algoritmos complicados, operaciones largas, validaciones que faltan o están duplicadas en , etc.) suele provocar problemas de rendimiento. Considera el caso en el que se realiza una búsqueda con una cadena vacía. Lo óptimo sería que el punto final de la búsqueda realizara alguna validación sencilla de los datos de entrada y fallara la petición rápidamente. En caso contrario, el servicio buscará en la base de datos y devolverá un error, retrasando innecesariamente el tiempo de respuesta.

Elección y diseño de la base de datos

Las bases de datos desempeñan un papel clave en el rendimiento de la aplicación . Existen varios tipos de bases de datos, como se

explica en el Capítulo 5. Si tu aplicación requiere un rendimiento muy alto, será fundamental elegir un tipo de base de datos adecuado y organizar correctamente los datos dentro de ella. Por ejemplo, almacenar los detalles de un único pedido de compra en varias tablas requerirá consolidación y retrasará la recuperación del pedido final. Es esencial estructurar los datos adecuadamente teniendo en cuenta el rendimiento.

Latencia de la red

El sistema nervioso central de cualquier aplicación es la red. Todos los componentes de una aplicación se comunican internamente a través de algún tipo de red. Por tanto, garantizar una buena conectividad entre los componentes es crucial, ya sea dentro del mismo centro de datos o a través de varios centros de datos. Además, los usuarios finales de todo el mundo interactuarán con la aplicación utilizando sus propias redes (2G, 3G, 4G, WiFi). La calidad de esas redes está fuera del control de los equipos de software, pero diseñar la aplicación para atender a los usuarios con una conectividad de red débil sí está dentro de su ámbito. Un buen diseño de UX que evite imágenes pesadas y transferencias de datos sustanciales es importante para aumentar el rendimiento de la aplicación para todos los usuarios.

Geolocalización de la aplicación y de los usuarios

Si los usuarios de tu sitio web sólo proceden de una región concreta, tener el sitio web alojado físicamente cerca de esa región reducirá el número de saltos de red y, por tanto, la latencia. Por ejemplo, si el sitio web es para clientes europeos pero está alojado en Singapur, conectarse al sistema requerirá múltiples saltos de red de ida y vuelta; alojarlo en algún lugar de Europa mejorará el rendimiento para los usuarios finales. Por el contrario, si el sitio web pretende servir a clientes de todo el mundo, debe haber una estrategia para replicarlo en diferentes ubicaciones de alojamiento (o utilizar redes de distribución de

contenidos [CDN]). Si utilizas infraestructura en la nube, debes acordarte de solicitar máquinas que estén físicamente más cerca de los clientes previstos: un error común es utilizar infraestructura que esté más cerca de la ubicación del equipo de desarrollo.

Infraestructura

La infraestructura es el esqueleto que soporta todos los músculos de un sistema. La potencia de tu infraestructura, en términos de CPU, memoria, etc., repercutirá directamente en la capacidad del sistema para soportar la carga. Diseñar la infraestructura para ofrecer un sistema de alto rendimiento es un arte en sí mismo. Los ingenieros de infraestructura recogen continuamente los resultados de las pruebas de rendimiento como uno de los parámetros para planificar las necesidades de infraestructura de la aplicación.

Integraciones de terceros

Cuando hay integraciones con componentes de terceros de , la aplicación depende del rendimiento de esos componentes. Cualquier latencia en un componente de terceros acabará sumándose a la latencia de la propia aplicación. Por ejemplo, como se explica en [el Capítulo 3](#), una aplicación típica de comercio minorista se integra con muchos servicios externos, como los sistemas de gestión de información de productos de los proveedores, los sistemas de gestión de almacenes, etc., y en estos casos, elegir componentes de alto rendimiento es vital.

Durante las pruebas de rendimiento, debes tener en cuenta todos estos factores para simular casos de prueba del mundo real. Por ejemplo, debes configurar un entorno de pruebas de rendimiento que sea muy similar al entorno de producción en cuanto a red, infraestructura, geolocalización, etc. De lo contrario, ipuedes no tener una medida precisa del rendimiento!

Indicadores clave de rendimiento

Medir o probar el rendimiento de una aplicación implica capturar un conjunto de indicadores clave de rendimiento (KPI) cuantitativos. Medirlos continuamente a lo largo del ciclo de desarrollo ayudará al equipo a corregir el rumbo antes y con menos esfuerzo. Por regla general, los KPI que debes monitorear son:

Tiempo de respuesta

El tiempo de respuesta se refiere al tiempo que tarda la aplicación en responder a una consulta del usuario; por ejemplo, el tiempo exacto que tarda en mostrar al cliente los resultados de una consulta de búsqueda de productos. Como hemos visto antes, el tiempo de respuesta esperado para las aplicaciones web es, como máximo, de 3 s; más allá de esto, corren el riesgo de perder la mayoría de los clientes. Ten en cuenta que 3 s es el retraso que experimenta el usuario final, y por tanto incluye tanto el tiempo de respuesta de la API como el tiempo que tarda el frontend en cargar completamente la página.

Concurrencia/rendimiento

A los sitios web pueden acceder numerosos usuarios de todo el mundo en un momento dado. De hecho, algunas aplicaciones de alta velocidad, como los sitios bursátiles, admiten millones de transacciones por segundo. Establecer que la aplicación puede soportar un determinado volumen de usuarios dentro de los límites aceptables en un momento dado se denomina medir *la concurrencia*. Por ejemplo, podrías querer validar que la aplicación puede responder en 3 segundos a 500 usuarios concurrentes.

Aunque "usuarios concurrentes" es un término utilizado habitualmente por las empresas y los equipos de software, si pensamos desde la perspectiva del sistema, éste recibe diversas peticiones de los usuarios finales y otros componentes, que se

ponen en cola y se seleccionan para ser procesadas una tras otra por hilos paralelos. Por lo tanto, desde su perspectiva, utilizar el número de usuarios concurrentes como indicador no le sienta bien. En su lugar, un indicador mejor para medir es el *rendimiento*. El rendimiento mide el número de solicitudes que el sistema puede soportar durante un intervalo de tiempo.

Para entenderlo mejor, considera la analogía de los coches que cruzan un puente muy corto sobre un río. Supongamos que hay cuatro carriles para coches. Suponiendo que el tráfico fluya sin problemas, cada coche podrá cruzar el puente en unos cientos de milisegundos. Así, en un segundo, el número total de coches que cruzarán el puente será de 30 a 40. Este valor de 30-40 coches por segundo es el rendimiento.

Tanto la concurrencia como el rendimiento son útiles en la planificación de la capacidad del servidor y suelen utilizarse en contextos diferentes para tomar decisiones impactantes.

Disponibilidad

La disponibilidad es una medida de la capacidad del sistema para responder a los usuarios finales dentro de los mismos límites aceptables durante un periodo continuo determinado.

Normalmente, se espera que los sitios web estén disponibles 24 horas al día, 7 días a la semana, excepto en caso de mantenimiento planificado. La disponibilidad es un criterio esencial que hay que probar porque una aplicación puede funcionar bien durante la primera media hora, pero las respuestas podrían degradarse con el tiempo debido a fugas de memoria, consumo excesivo de la capacidad de la infraestructura por trabajos por lotes paralelos, y muchas otras razones impredecibles.

Ahora que hemos hablado de los KPI, veamos cómo medirlos.

Tipos de pruebas de rendimiento

Para medir los KPI de , tienes que diseñar específicamente tus pruebas de rendimiento de una determinada manera. La siguiente lista describe tres tipos comunes de pruebas de rendimiento:

Pruebas de carga/volumen

Como ya se ha dicho, la concurrencia o rendimiento se mide para validar que la aplicación puede servir al volumen previsto de usuarios en un tiempo aceptable. Por ejemplo, supongamos que quieras que la funcionalidad de búsqueda responda en 2 segundos para un volumen de 300 usuarios. Una prueba de rendimiento para simular este volumen de usuarios y validar si la aplicación cumple el tiempo de respuesta objetivo previsto se denomina *prueba de volumen* o *prueba de carga*. Es posible que tengas que repetir estas pruebas varias veces para observar la coherencia y medir la media para comparar la aplicación.

Pruebas de estrés

Un comportamiento comúnmente observado es que el rendimiento de una aplicación empieza a degradarse a medida que más usuarios la estresan. Por ejemplo, puede rendir dentro de límites aceptables para X usuarios, pero más allá de X usuarios empieza a responder con retrasos y, finalmente, a X+n usuarios, responde con errores. Necesitas la medida exacta de estas cifras. Esta medida se utilizará en la planificación de la infraestructura cuando se escala la aplicación a nuevas regiones, o durante eventos como las rebajas. La prueba de rendimiento se diseñará para aumentar lentamente la carga de la aplicación en pequeños pasos más allá de los límites de la prueba de volumen, para determinar con precisión el punto en el que responde con errores. Este proceso de estresar el sistema para encontrar el punto de ruptura se denomina *prueba de estrés*.

Pruebas de remojo

Cuando la aplicación funciona con el volumen de usuarios previsto en durante un tiempo, puede producirse una degradación del tiempo de respuesta debido a problemas de infraestructura, fugas de memoria u otros problemas. Las pruebas *de rendimiento* diseñadas para mantener la aplicación bajo un volumen constante de carga durante un periodo prolongado y observar el comportamiento se denominan *pruebas de remojo*.

Al diseñar todas estas pruebas, un punto importante es mantenerlas realistas y evitar sobrecargar la aplicación con situaciones extremas que quizá nunca se produzcan. Por ejemplo, no todos los usuarios se conectarán a la aplicación en el mismo instante. Un caso de uso más realista será que los usuarios se conecten gradualmente, con intervalos de unos milisegundos entre ellos. Este retraso entre el inicio de la prueba y el momento en que se considera que todos los usuarios virtuales están conectados se denomina *tiempo de rampa*. Tus casos de prueba deben incluir un diseño práctico de este tipo; por ejemplo, podrías planificar el aumento de 100 usuarios en 1 minuto.

Además, los usuarios no son robots capaces de iniciar sesión, buscar un producto y completar una compra en milisegundos, pero los casos de prueba de rendimiento pueden diseñarse así involuntariamente. En realidad, los usuarios tardan al menos unos segundos en pensar entre una acción y otra, y suelen tardar minutos en completar una transacción como la compra de un producto después de iniciar sesión. Esto se denomina *tiempo de reflexión* en términos de pruebas de rendimiento. Tienes que incluir el tiempo de reflexión adecuado en tus casos de prueba y espaciar las acciones del usuario unos segundos o minutos. Relacionado con el tiempo de reflexión hay otro concepto llamado *ritmo*, que define el tiempo entre transacciones (no entre acciones del usuario). En la vida real, los usuarios podrían volver a iniciar transacciones pasado cierto

tiempo. Así que, si esperas 1.000 transacciones por hora durante las horas punta de ventas, puedes repartir las transacciones a lo largo de la hora configurando el tiempo de ritmo. Estos tres atributos deben ajustarse sabiamente para medir de forma realista el rendimiento de una aplicación.

Tipos de patrones de carga

En la sección anterior hablamos de los distintos tipos de pruebas de rendimiento que se utilizan para medir los KPI. Estas pruebas de rendimiento se traducen en la generación de diferentes patrones de carga en la aplicación, utilizando los atributos que acabamos de comentar: el tiempo de aceleración, el tiempo de reflexión, el número de usuarios concurrentes y el ritmo. Discutiremos algunos patrones de carga comúnmente probados en esta sección:

Patrón de aceleración constante

En el patrón de aumento constante (ilustrado en la [Figura 8-1](#)), los usuarios aumentan de forma constante en un periodo determinado, y luego se mantiene la carga de forma constante durante un periodo sostenido para medir el rendimiento. Se trata de un patrón muy común en escenarios del mundo real -por ejemplo, las rebajas del Viernes Negro- en los que los usuarios llegan a la aplicación de forma gradual pero constante y permanecen en ella durante un tiempo antes de abandonarla de forma constante.

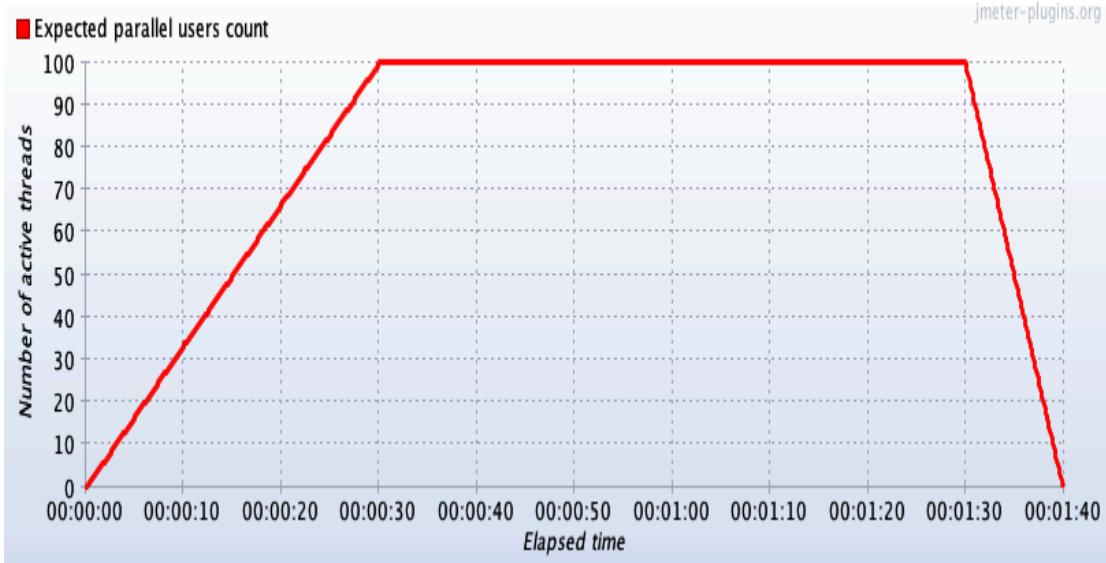


Figura 8-1. Aumento constante de usuarios

Patrón de rampa escalonada

Con el patrón de aumento escalonado([Figura 8-2](#)), los usuarios son aumentados en lotes periódicamente; por ejemplo, 100 usuarios cada 2 minutos. Observar y medir el rendimiento de la aplicación para cada paso de usuarios ayudará a evaluar la aplicación para diferentes cargas. El patrón de aumento escalonado es útil para ajustar el rendimiento y planificar la capacidad de la infraestructura.

NOTA

La evaluación comparativa consiste en medir el tiempo medio de respuesta de a partir de ejecuciones repetidas.

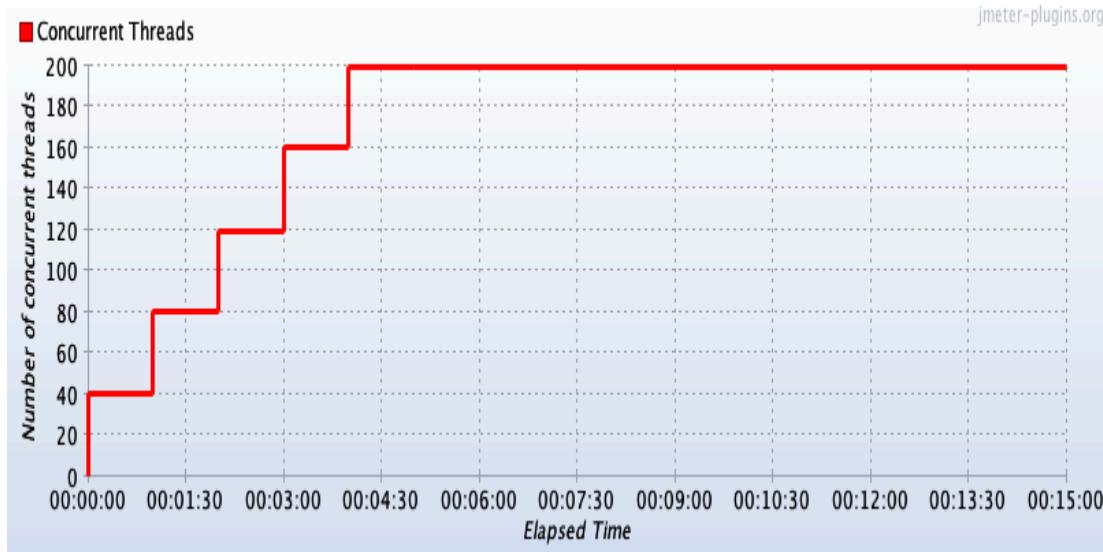


Figura 8-2. Aumento escalonado de usuarios

Patrón pico-descanso

El patrón pico-descanso([Figura 8-3](#)) es cuando el sistema se eleva hasta alcanzar el pico de carga y luego se reduce hasta el descanso completo en ciclos repetidos. Este escenario puede observarse en algunas aplicaciones como las redes sociales, donde el pico va y viene en ciclos a lo largo de un día.



Figura 8-3. Patrón de carga pico-descanso

Las herramientas de pruebas de rendimiento ayudan a generar estos patrones fácilmente, como veremos más adelante en el capítulo.

Pasos de las pruebas de rendimiento

Ahora que hemos hablado de los KPI, los tipos de pruebas de rendimiento y los patrones de carga, el siguiente paso es recorrer los pasos de un ejercicio de pruebas de rendimiento. Esto te ayudará a planificar el tiempo y la capacidad necesarios para las pruebas de rendimiento en tu proyecto.

Paso 1: Definir los KPI objetivo

El primer paso es definir los KPI objetivo en función de las necesidades de la empresa. La mejor forma de empezar a pensar en las cifras objetivo es considerarlas cualitativamente, y luego traducirlas en cifras.¹ Por ejemplo, el pensamiento cualitativo sobre el rendimiento podría conducir a objetivos como

- La aplicación debe poder ampliarse a un nuevo país más.
- La aplicación debe funcionar mejor que su competidor X.
- La nueva versión de la aplicación debería funcionar mejor que la anterior.

Estos objetivos cualitativos conducen naturalmente a los pasos siguientes. Si el objetivo es hacerlo mejor que la última versión de la aplicación, tienes que medir el rendimiento de la versión anterior y ver si tus cifras actuales son mejores. Del mismo modo, si conoces las cifras de rendimiento de la competencia, tienes que validar que tus cifras son mejores que las suyas.

NOTA

La gente de negocios tiende a dar cifras de rendimiento que pueden no reflejar el patrón de uso real. Obtén siempre los KPI objetivo a partir de los datos:

- Si existe una aplicación, analiza los datos de producción para llegar a los KPI y a los patrones de carga.
- Si estás creando una nueva aplicación, pide los datos de la competencia.
- Si la aplicación es completamente nueva y no tiene datos de referencia, utiliza datos sobre el uso de Internet en todo el país, la duración probable de los picos, etc., para calcular tus KPI objetivo.

Paso 2: Definir los casos de prueba

El segundo paso es describir los casos de prueba utilizando los patrones de carga y la semántica del tipo de prueba de rendimiento. Tus casos de prueba deben cubrir obligatoriamente la medición de la disponibilidad, el rendimiento y el tiempo de respuesta de todos los puntos finales críticos de la aplicación. Los casos de prueba de rendimiento revelarán posteriormente la configuración de los datos de prueba necesarios para ejecutar los casos de prueba. Al final, puede que sólo necesites un puñado de casos de prueba de rendimiento, a diferencia de los casos de prueba funcionales.

Paso 3: Preparar el entorno de pruebas de rendimiento

Como ya se ha dicho, el entorno de pruebas de rendimiento debe ser lo más parecido posible al entorno de producción, para que puedas obtener resultados realistas. Esto también te ayudará a identificar cualquier cuello de botella de rendimiento en las configuraciones del entorno.

Aquí tienes un ejemplo de lista de control para lograr este objetivo, que puedes adaptar a tus circunstancias:

- Los niveles/componentes respectivos deben desplegarse de forma similar.
- Las configuraciones de las máquinas (número de CPU, capacidad de memoria, versión del SO, etc.) deben ser similares.
- Las máquinas deben estar alojadas en la misma geolocalización en la nube.
- El ancho de banda de la red entre máquinas debe ser similar.
- Las configuraciones de aplicación, como la limitación de la dosis, deben ser exactamente las mismas.
- Si va a haber trabajos por lotes ejecutándose en segundo plano, deben estar preparados. Si hay que enviar correos electrónicos, esos sistemas también deben estar preparados.
- Los equilibradores de carga, si los hay, deben estar instalados.
- El software de terceros debe estar disponible, al menos en versión de prueba.

Configurar un entorno similar al de producción para las pruebas suele ser complicado debido a los costes adicionales que conlleva, aunque las provisiones en la nube son más baratas. Puede que tengas que mantener una conversación sobre coste frente a valor con las partes interesadas de la empresa. Si no ganas esa batalla, prepárate para hacer concesiones significativas en ciertas partes de la configuración del entorno de rendimiento y deja claro a las partes interesadas respectivas que, debido a esas concesiones, las cifras de rendimiento medidas podrían no ser infalibles.

NOTA

Una buena práctica es solicitar que el entorno de pruebas de rendimiento se configure junto con el entorno de control de calidad justo al principio del proyecto, para que esté disponible cuando lo necesites.

Aparte del entorno de pruebas de rendimiento, también necesitas una máquina independiente que sea el ejecutor de pruebas, es decir, que ejecute las pruebas de rendimiento. Planifica tener ejecutores de pruebas individuales alojados en diferentes geolocalizaciones (esto es posible con los proveedores de la nube) para observar los respectivos comportamientos de rendimiento con latencias de red de múltiples países, si tu aplicación es destinada a servir a una audiencia global.

Paso 4: Preparar los datos de prueba

Del mismo modo que el entorno de pruebas de rendimiento debe ser lo más similar posible al entorno de producción, los datos de prueba deben reflejar lo más posible los datos de producción. Las cifras de rendimiento que medirás dependerán en gran medida de la calidad de los datos de prueba, por lo que éste es un paso crítico. Una situación ideal sería utilizar datos de producción reales tras anonimizar cualquier información sensible de los usuarios, ya que reflejarán el tamaño real de la base de datos y la complejidad de los datos. Sin embargo, esto puede no ser posible por motivos de seguridad en determinadas situaciones. En tales casos, prepara datos de prueba que imiten fielmente los datos de producción.

Algunos consejos para crear datos similares a los de producción son:

- Calcula el tamaño de la base de datos de producción (por ejemplo, 1 GB o 1 TB) y crea guiones para llenar los datos de prueba. Puede ser necesario limpiar y repoblar los datos de prueba en cada ejecución de prueba, por lo que disponer de los

scripts de creación y limpieza de los datos de prueba será crucial.

- Crea una variedad de datos de prueba similares a los que se observan en producción. En lugar de "Camisa1", "Camisa2", etc., utiliza valores reales similares a los de producción, como "Camiseta de cuello en V verde oliva Van Heusen".
- Rellena una buena parte de valores erróneos, como direcciones con faltas de ortografía, espacios en blanco, etc., que podrían representar entradas reales del usuario.
- Tener una distribución similar de los datos en función de factores como la edad, el país, etc.
- Dependiendo de los casos de prueba, puede que tengas que crear muchos datos únicos, como números de tarjeta de crédito únicos, credenciales de inicio de sesión, etc., para ejecutar pruebas de volumen con usuarios concurrentes.

Sí, preparar los datos de prueba puede ser un trabajo tedioso! Estas actividades deben planificarse con mucha antelación en el ciclo de lanzamiento. Es imposible incluirlas a posteriori, y si lo intentas, los datos de prueba podrían no ser de buena calidad, lo que daría lugar a cifras de rendimiento inexactas.

Paso 5: Integrar las herramientas APM

El siguiente paso es integrar la aplicación herramientas de monitoreo del rendimiento (APM) (por ejemplo, New Relic, Dynatrace, Datadog) para que puedas ver cómo se comportó el sistema durante las pruebas de rendimiento. Estas herramientas ayudan mucho a depurar cualquier problema de rendimiento. Por ejemplo, las solicitudes pueden fallar durante la ejecución de las pruebas de rendimiento debido a que no hay suficiente memoria en la máquina, y las herramientas APM sacarán a la luz estos problemas fácilmente.

Paso 6: Guión y ejecución de las pruebas de rendimiento mediante herramientas

El último paso es guionizar los casos de prueba de rendimiento utilizando herramientas y ejecutarlos en el entorno de pruebas de rendimiento. Hay muchas herramientas que puedes utilizar para guionizar y ejecutar tus casos de prueba de rendimiento con un solo clic, y también integrarlas con CI para ayudarte a desplazarte a la izquierda. JMeter, Gatling, k6 y Apache Benchmark (ab) son algunos de los niños populares en este patio de recreo. Además de estas herramientas de código abierto, también hay herramientas comerciales alojadas en la nube, como BlazeMeter, NeoLoad y otras. Algunas de estas herramientas proporcionan interfaces de usuario sencillas para configurar las pruebas de rendimiento y no requieren codificación. Puedes obtener informes de ejecución de pruebas con gráficos, mientras que las herramientas comerciales ofrecen incluso una vista de panel de control. En la siguiente sección se incluye un ejercicio para crear scripts de prueba utilizando JMeter e integrarlos con CI.

CONSEJO

Las pruebas de rendimiento pueden llevar a desde unos minutos hasta unas horas, dependiendo de la prueba. Para hacerte una idea de cuánto durará la tuya, quizás quieras hacer un simulacro de los scripts con un número de usuarios menor antes de iniciar la prueba completa.

Ésos son los seis pasos de las pruebas de rendimiento: los aplicaremos como parte de un ejercicio en la siguiente sección. La clave para ejecutar con éxito todos los pasos de tu proyecto es planificar adecuadamente la capacidad para ellos, como se ha mencionado antes. Al planificar, incluye también tiempo y capacidad para recopilar informes de ejecución de pruebas, depurar y solucionar problemas de rendimiento, y hacer ajustes de capacidad

del servidor. ¡Así se completará todo el ciclo de pruebas de rendimiento!

Ejercicios

Ahora, tomaremos el ejemplo de una aplicación de gestión de bibliotecas en línea y navegaremos por los pasos de las pruebas de rendimiento. Por comodidad, mantendremos sencillas las características de la aplicación de gestión de bibliotecas. Tiene dos tipos de usuarios: los administradores, que pueden añadir y eliminar libros, y los clientes, que pueden ver todos los libros y buscar un libro por su ID. Las respectivas API REST son /addBook, /deleteBooks, /books, y /viewBookByID.

Paso 1: Definir los KPI objetivo

Para llegar a los KPI objetivo de la aplicación de la biblioteca , supongamos que hemos obtenido los siguientes datos de la empresa y del equipo interno de marketing:

- Están haciendo una campaña agresiva para su lanzamiento en dos ciudades europeas y esperan que se unan 100.000 usuarios únicos en el primer año.
- Tienen un estudio que dice que los usuarios pasan una media de 10 minutos buscando libros, viendo libros similares, etc., en una sola sesión.
- El estudio también dice que un usuario típico puede pedir prestado un libro dos veces al mes por término medio. Por tanto, esperan que los usuarios accedan al sitio dos veces al mes.
- En Europa, los usuarios están activos en Internet entre las 10 de la mañana y las 10 de la noche (12 horas) todos los días.

Con esos datos, podemos calcular lo siguiente:

- Total de usuarios que acceden al sitio mensualmente = 100.000 usuarios * 2 accesos al mes = 200.000 usuarios mensuales
- Usuarios medios diarios = 200.000 usuarios mensuales ÷ 30 días al mes = 6.667 usuarios diarios. (Ten en cuenta que podría haber más usuarios los fines de semana que entre semana, pero estamos calculando la media de usuarios diarios).
- Usuarios medios por hora = 6.667 usuarios medios diarios ÷ 12 horas al día = 555 usuarios por hora. (Del mismo modo, podría haber más usuarios por hora en algunos momentos del día que en otros, como al mediodía o por la noche).
- Para tener en cuenta los picos, podemos ser generosos y redondear a 1.000 usuarios por hora.
- Cada usuario utiliza el sitio web durante un tiempo de sesión de 10 minutos, lo que equivale a 0,166667 horas.
- Número de usuarios concurrentes = 1.000 usuarios pico por hora * 0,166 = 166 usuarios concurrentes.
- Suponiendo que cada usuario haga al menos 5 peticiones (buscar libros y ver la lista de libros) en una sesión de 10 minutos, el sistema tendrá que soportar 5 * 1.000 usuarios por hora = 5.000 peticiones por hora.

Según los cálculos, estos son nuestros KPI objetivo:

- Para 166 usuarios simultáneos, el sistema debe responder en 3 segundos.
- El rendimiento del sistema tiene que soportar 5.000 peticiones por hora.

Deberíamos llegar a un consenso con el equipo directivo del cliente sobre estas cifras antes de seguir adelante. También podemos

sondear el negocio para pensar más allá del primer año y comprobar de nuevo las cifras objetivo.

NOTA

Esto es sólo un ejemplo de cálculo para dar una idea de cómo calcular los KPI objetivo. Como ya se ha dicho, el primer lugar donde hay que indagar es en los datos de producción de la aplicación existente o en los datos de la competencia, que darán una imagen más precisa de los KPI y los patrones de carga.

Paso 2: Definir los casos de prueba

Ahora que conocemos los KPI objetivo, podemos definir casos de prueba de rendimiento apropiados basados en las características de la aplicación de la biblioteca. Recordando los factores que hemos discutido antes, los casos de prueba para nuestra aplicación podrían incluir:

- Compara los tiempos de respuesta de los cuatro puntos finales: /addBook, /deleteBooks, /viewBookById, y /books.
- Prueba en volumen los puntos finales de cara al cliente con 166-200 usuarios simultáneos, es decir, los puntos finales /viewBookById y /books deberían responder en menos de 3 segundos con 166 usuarios simultáneos. (Ten en cuenta que 3 segundos incluye el rendimiento del frontend, por lo que tendrás que afirmar con un valor límite inferior específico de tu aplicación para los puntos finales). Sólo los administradores acceden a los otros dos puntos finales; por lo tanto, puede que no sea necesario realizar pruebas de volumen para ellos.
- Realiza una prueba de estrés de los puntos finales de cara al cliente con pasos ascendentes de 100 usuarios y encuentra los puntos de ruptura.

- Valida el rendimiento de 5.000 solicitudes por hora. El flujo de usuario para este caso de prueba podría ser ver la lista de libros, seleccionar un libro y hojear su descripción, luego volver a la página de la lista de libros, seleccionar otro libro y leer su descripción, y volver de nuevo a la página de la lista de libros - en total, haciendo cinco peticiones por flujo de usuario-. Incluye un tiempo de reflexión de, digamos, 30 segundos entre cada una de estas acciones, y asume que 45 usuarios pueden seguir haciendo este flujo de usuario durante una hora. Aumenta el número de usuarios lentamente durante los primeros 10 minutos.
- Prueba de remojo durante 12 horas seguidas para validar que el sistema está disponible para los usuarios de forma continua. Podríamos reutilizar el diseño de la prueba de rendimiento anterior para ejecutarla también durante 12 horas, si tiene éxito.

Pasos 3-5: Preparar los datos, el entorno y las herramientas

Para este ejercicio, he desarrollado una aplicación de biblioteca de ejemplo y la he alojado en Heroku. Para completar el ejercicio tú mismo, puedes crear un stub (consulta "[WireMock](#)" para más detalles al respecto) en tu máquina local para el endpoint /books, como se muestra en el [Ejemplo 8-1](#), y configurarlo para que devuelva 50 libros. Pruébalo una vez después de configurarlo.

ADVERTENCIA

Realizar pruebas de carga de gran volumen en API públicas puede considerarse un ataque DDoS; de ahí la necesidad de crear stubs para el ejercicio. Como alternativa, las distintas herramientas de pruebas de rendimiento (como JMeter y Gatling) proporcionan sitios de prueba que puedes utilizar para practicar con pruebas de rendimiento. Consulta sus respectivos sitios oficiales para obtener las URL de los sitios de prueba y ejecútalos sólo con la carga mínima prescrita.

Ejemplo 8-1. punto final /libros

GET: /books

Response:

Status Code: 200
Body:
[
{ "id": 1,
 "name": "Man's search for meaning",
 "author": "Victor Frankl",
 "Language": "English",
 "isbn": "ABCD1234"
,
{ "id": 2,
 "name": "Thinking Fast and Slow",
 "author": "Daniel Kahneman",
 "Language": "English",
 "isbn": "UFGH1234"
}]

Paso 6: Escribe los casos de prueba y ejecútalos con JMeter

JMeter es una popular herramienta de pruebas de rendimiento . Es totalmente de código abierto y puede integrarse con CI y generar bonitos informes gráficos. Se integra con BlazeMeter, una

herramienta de análisis del rendimiento alojada en la nube, si quieras liberarte de las tareas de gestión de la infraestructura. JMeter se basa en Java, y existe una comunidad de desarrolladores activos que contribuyen con distintos plug-ins valiosos. Las figuras de "[Tipos de patrones de carga](#)" se crearon utilizando uno de estos plug-ins. También hay buena [documentación](#) y tutoriales sobre muchos casos de uso para principiantes. Instalemos la herramienta y escribamos algunos scripts de prueba para nuestra aplicación de biblioteca.

Configurar

Sigue estos pasos para configurar JMeter:

1. Descarga el archivo ZIP del [sitio oficial](#) e instálalo en . Asegúrate de que tu versión local de Java es compatible. Asegúrate también de que la variable JAVA_HOME está configurada en el *bash_profile* de tu entorno.
2. Para abrir la GUI de JMeter, ejecuta el script de shell *jmeter.sh* dentro de la carpeta */apache-JMeter-version/bin* desde tu terminal.
3. También utilizaremos plug-ins de JMeter. Puedes descargar el Gestor de Plugins del [sitio oficial](#) y colocar el JAR en */apache-JMeter-version/lib/ext*.
4. Reinicia JMeter. Entonces deberías ver el Gestor de Plugins en el menú Opciones.

Flujo de trabajo

Utiliza los pasos descritos aquí para configurar un esqueleto de prueba JMeter básico y añade una prueba sencilla para comparar el tiempo de respuesta del punto final /books:

1. Crea un grupo de hilos desde la interfaz gráfica de JMeter haciendo clic con el botón derecho en Plan de pruebas en el

panel izquierdo y seleccionando Añadir → Hilos (Usuarios) → Grupo de hilos. Nombra el grupo de hilos *ViewBooks*. Configura los parámetros como se muestra en la Figura 8-4 (*Número de hilos = 1, Periodo de aceleración = 0, Recuento de bucles = 10*) para registrar los tiempos de respuesta del punto final 10 veces diferentes y calcular su promedio.

Thread Group

Name:

Comments:

Action to be taken after a Sampler error

Continue Start Next Thread Loop Stop Thread Stop Test Stop Test Now

Thread Properties

Number of Threads (users):

Ramp-up period (seconds):

Loop Count: Infinite

Figura 8-4. Configuración del grupo de hilos para ejecutar una petición 10 veces

2. Añade el muestreador Solicitud HTTP para configurar los parámetros de la API. Haz clic con el botón derecho del ratón en el grupo de hilos que acabas de añadir en el panel izquierdo y selecciona Añadir → Muestreador → Solicitud HTTP. Introduce el nombre del servidor web, el tipo de solicitud HTTP y la ruta (consulta [la Figura 8-5](#)). Nombra al muestreador *viewBooksRequest*.

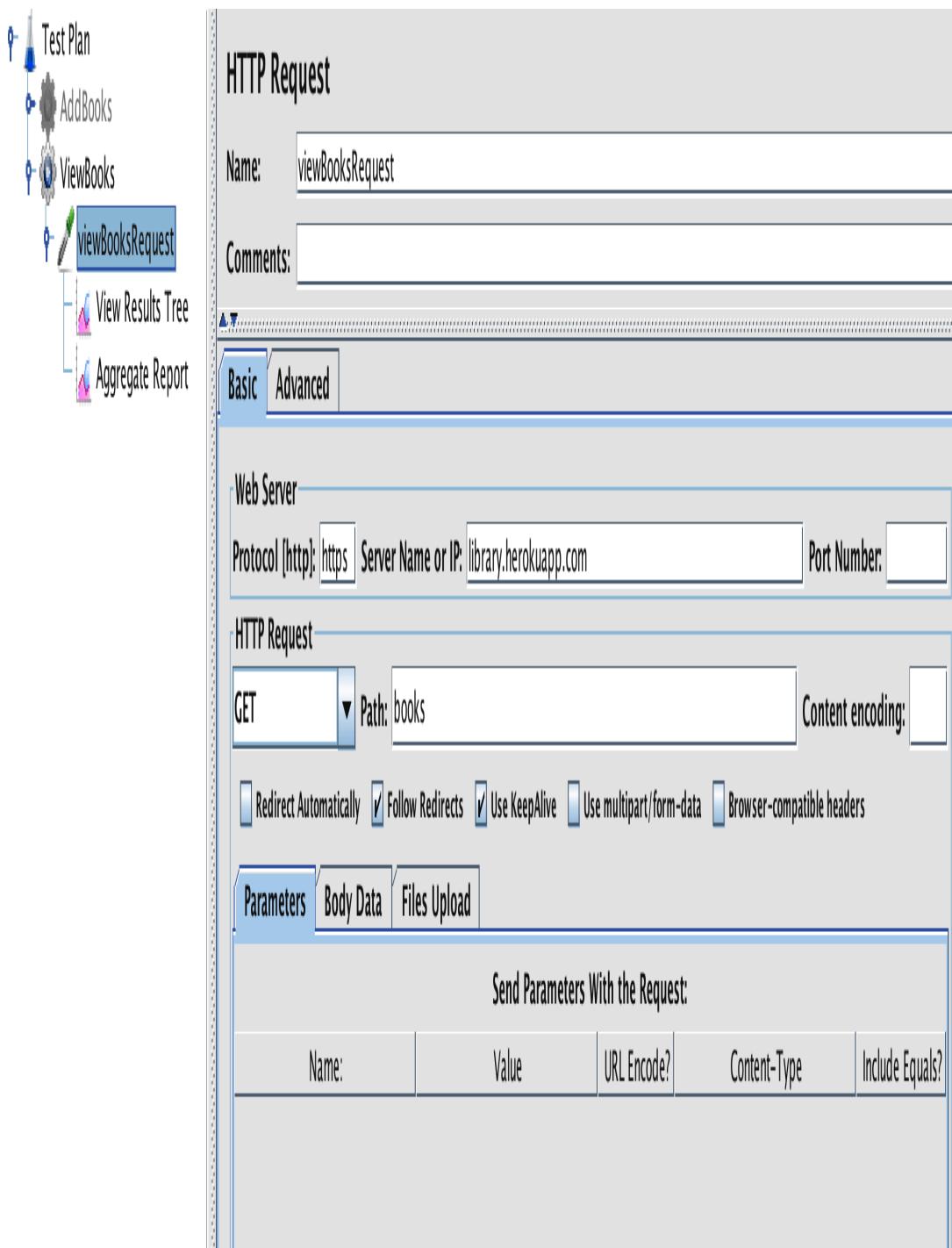


Figura 8-5. Configuración de la petición HTTP viewBooksRequest

3. Añade oyentes, que grabarán en cada solicitud y respuesta durante la ejecución de la prueba. Haz clic con el botón derecho en el muestreador viewBooksRequest y selecciona Añadir →

Escuchas → Ver árbol de resultados, luego repite el proceso pero esta vez selecciona la escucha Informe agregado .

4. Guarda el esqueleto básico de la prueba. A continuación, para medir el tiempo de respuesta, haz clic en el botón Ejecutar. Los resultados estarán disponibles en las secciones de los dos oyentes.

Haz clic en Ver árbol de resultados en el panel izquierdo para ver la salida de este oyente. Verás la lista de solicitudes individuales realizadas por JMeter, con una indicación de éxito o fracaso para cada una. JMeter considera que el código de estado de respuesta 200 significa éxito; de lo contrario, considera que la solicitud ha fallado. Un punto a tener en cuenta es que puede haber situaciones en tu aplicación en las que el servicio devuelva un código de estado 200 para indicar que la operación se ha ejecutado, pero puede no haber producido los resultados previstos. Por ejemplo, el punto final /addBook podría devolver un código de estado 200 para libros duplicados con un mensaje indicando que se trata de un duplicado. En tales casos, tienes que añadir aserciones explícitas sobre los resultados (las aserciones, como los escuchadores, también son componentes de JMeter). La vista Ver árbol de resultados también mostrará los datos de solicitud y respuesta al hacer clic en cada solicitud para una mayor depuración, como se muestra en la **Figura 8-6**.

View Results Tree

Name: View Results Tree

Comments:

Write results to file / Read from file

Filename

Browse...

Log/Display Only: Errors Successes Configuration

Search:

Case sensitive

Regular exp.

Search

Reset

Text

Sampler result

Request

Response data

- viewBooksRequest

Response Body

Response headers

Find

Case sensitive

Regular exp.

```
[{"id":1}, {"id":2}, {"id":3}, {"id":4}, {"id":5}, {"id":6}, {"id":7}, {"id":8}, {"id":9}, {"id":10}, {"id":11}, {"id":12}, {"id":13}, {"id":14}, {"id":15}, {"name": "Games People Play", "author": "Erin", "language": "English", "isbn": "ABCD12354"}, {"name": "Shoe Dog", "author": "Phil Knight", "language": "English", "isbn": "ABCD12345", "id": 16}, {"name": "Thinking Fast and Slow", "author": "Daniel Kehman", "language": "English", "isbn": "ABCD12352", "id": 17}, {"name": "Ark", "author": "Veronica Roth", "language": "English", "isbn": "ABCD12350", "id": 18}, {"name": "Surely You're Joking Mr. Feynman", "author": "Richard Feynman", "language": "English", "isbn": "ABCD12351", "id": 19}, {"name": "Prisoner's of Geography", "author": "Tim Marshall", "language": "English", "isbn": "ABCD12353", "id": 20}, {"name": "Emergency Skin", "author": "Jemisin", "language": "English", "isbn": "ABCD12357", "id": 21}, {"name": "Prisoner's of Geography", "author": "Tim Marshall", "language": "English", "isbn": "ABCD12353", "id": 22}, {"name": "Ark", "author": "Veronica
```

Figura 8-6. Salida del oyente Ver Árbol de Resultados

Del mismo modo, cuando hagas clic en Informe agregado, verás una tabla con métricas como la media, la mediana, el rendimiento, etc. Para el endpoint /books, el tiempo medio de respuesta para 10 muestras es de 379 ms (ver [Figura 8-7](#)), lo que sugiere que el tiempo de respuesta en el mejor de los casos es de 379 ms cuando la aplicación no está bajo carga.

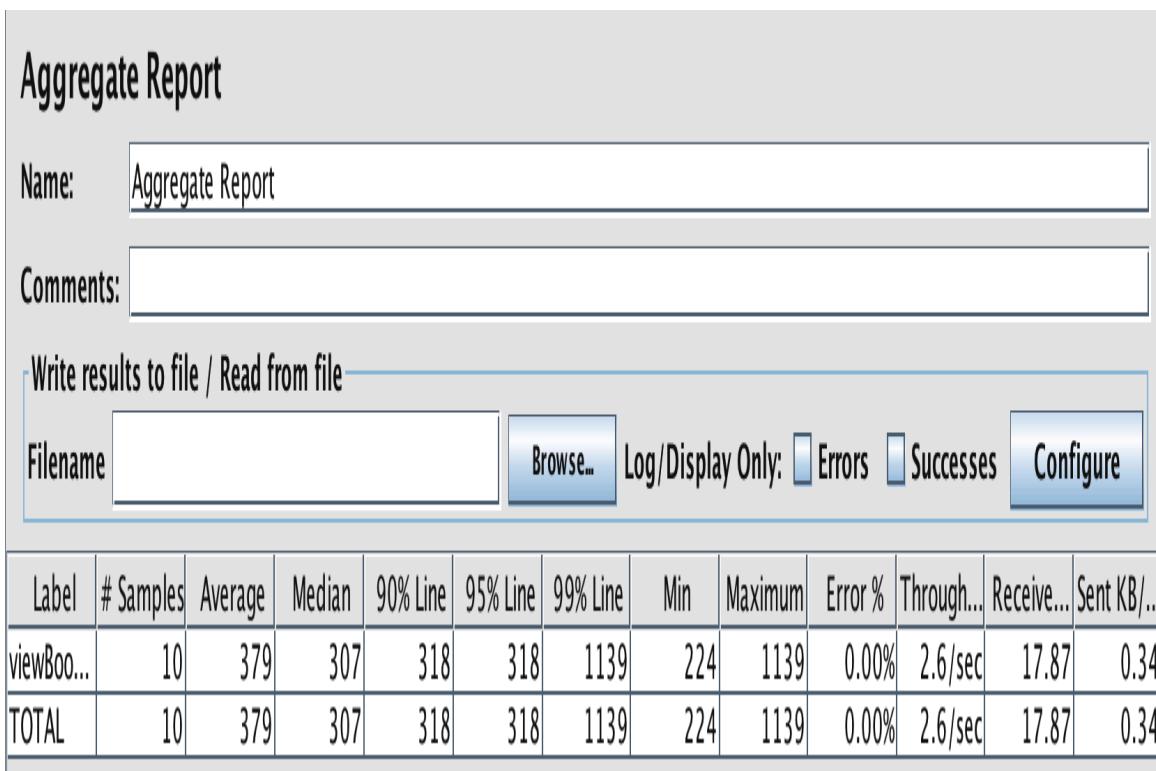


Figura 8-7. Vista del Informe Agregado del tiempo de respuesta del punto final /books

El siguiente paso es realizar pruebas de carga en el endpoint /books con 166 usuarios simultáneos y comprobar el tiempo de respuesta de . JMeter ofrece muchas formas de configurar diferentes patrones de carga. Aquí veremos tres opciones sencillas para configurar la carga en el endpoint /books.

Como hemos visto antes, el grupo de hilos es un elemento básico de JMeter bajo el que puedes colocar diferentes escuchadores y

controladores. También puede utilizarse para configurar parámetros de carga, como el número de hilos paralelos, la duración del periodo de aceleración y el número de veces que debe repetirse la prueba. Anteriormente, configuraste el grupo de hilos *ViewBooks* para ejecutar la petición /books en un bucle 10 veces con el fin de comparar su tiempo de respuesta. Ahora, para realizar una prueba de volumen, puedes cambiar los parámetros a Number of Threads = 166, Ramp-up period = 0, Loop Count = 5. JMeter pondrá en marcha 166 subprocessos simultáneos sin tiempo de aceleración y repetirá el bucle 5 veces para obtener el tiempo medio de respuesta.

También hay un práctico complemento que te da acceso a tipos adicionales de grupos de hilos que son útiles para configurar distintos patrones de carga, como el patrón de rampa escalonada. Aquí te mostraré cómo utilizar el Grupo de Hilos de Concurrencia y el Grupo de Hilos Último. Empezaremos con el Grupo de Hilos de Concurrencia, que te proporciona un controlador de concurrencia para pruebas de volumen:

1. Selecciona Opciones → Gestor de complementos. Busca "Grupos de hilos personalizados" en la pestaña Plugins disponibles e instálalo.
2. Reinicia JMeter para que los nuevos tipos de grupos de hilos estén disponibles.
3. Haz clic con el botón derecho en Plan de pruebas en el panel izquierdo y selecciona Añadir → Hilos (Usuarios) → bzm → Grupo de hilos de concurrencia.
4. Configura los parámetros de carga como se muestra en [la Figura 8-8](#) (Target Concurrency = 166, Ramp Up Time = 0.5, Hold Target Rate Time = 2). Esto indica a JMeter que cargue 166 usuarios en 30 segundos y mantenga a cada uno de ellos durante 2 minutos en el sistema.

5. Añade el muestreador de solicitudes HTTP como antes en este grupo de hilos, ejecuta la prueba y visualiza los resultados en los oyentes.

The screenshot shows the configuration for a 'Concurrency Thread Group' named 'bzm - Concurrency Thread Group'. The 'Name' field contains 'Volume Test ViewBooks'. Under 'Action to be taken after a Sampler error', the 'Continue' option is selected. The 'Target Concurrency' is set to 166, 'Ramp Up Time (min)' is 0.5, 'Ramp-Up Steps Count' is 1, and 'Hold Target Rate Time (min)' is 2.

Name:	Volume Test ViewBooks
Comments:	
Action to be taken after a Sampler error	<input checked="" type="radio"/> Continue <input type="radio"/> Start Next Thread Loop <input type="radio"/> Stop Thread <input type="radio"/> Stop Test <input type="radio"/> Stop Test Now
Target Concurrency:	166
Ramp Up Time (min):	0.5
Ramp-Up Steps Count:	1
Hold Target Rate Time (min):	2

Figura 8-8. Grupo de hilos de concurrencia para probar el volumen del punto final /books

El complemento Grupos de hilos personalizados también proporciona un tipo de Grupo de hilos Ultimate con funciones adicionales. Por ejemplo, te permite adaptar tu patrón de carga configurando el retardo inicial antes de la ejecución de la prueba, el tiempo de

apagado después de la ejecución de la prueba, etc. Para utilizar un Grupo de hilos Ultimate para pruebas de volumen:

1. Haz clic con el botón derecho en Plan de pruebas y selecciona Añadir → Temas (Usuarios) → jp@gc Ultimate Thread Group.
2. Configura los parámetros de carga como se ve en [la Figura 8-9](#) (Start Threads Count = 166, Initial Delay = 0, Startup Time = 10, Hold Load For = 60, Shutdown Time = 10). Esto indica a JMeter que inicie 166 peticiones simultáneas en 10 segundos y mantenga la carga durante 1 minuto, tras lo cual reducirá los usuarios en 10 segundos. Aquí también puedes añadir más filas según convenga para generar el patrón de pico-descanso.
3. Añade el muestreador de solicitudes HTTP como antes, ejecuta la prueba y visualiza los resultados.

jp@gc - Ultimate Thread Group

Name:	Volume Test ViewBooks										
Comments:											
Action to be taken after a Sampler error											
<input checked="" type="radio"/> Continue <input type="radio"/> Start Next Thread Loop <input type="radio"/> Stop Thread <input type="radio"/> Stop Test <input type="radio"/> Stop Test Now											
Threads Schedule <table border="1"> <thead> <tr> <th>Start Threads Count</th> <th>Initial Delay, sec</th> <th>Startup Time, sec</th> <th>Hold Load For, sec</th> <th>Shutdown Time</th> </tr> </thead> <tbody> <tr> <td>166</td> <td>0</td> <td>10</td> <td>60</td> <td>10</td> </tr> </tbody> </table>		Start Threads Count	Initial Delay, sec	Startup Time, sec	Hold Load For, sec	Shutdown Time	166	0	10	60	10
Start Threads Count	Initial Delay, sec	Startup Time, sec	Hold Load For, sec	Shutdown Time							
166	0	10	60	10							
Add Row Copy Row Delete Row											

Figura 8-9. Grupo de hilos Ultimate para probar el volumen del punto final /books

La Figura 8-10 muestra los resultados utilizando la opción de grupo de hilos simple (la primera opción) con 166 usuarios concurrentes y 0 tiempo de aceleración, con un promedio de 5 recuentos de bucles:

Average = 801 ms, 90% Line = 1499 ms. En otras palabras, el 90% de los 166 usuarios concurrentes obtienen su respuesta en ~1,5 s, y de media, los 166 usuarios concurrentes obtienen su respuesta en 0,8 s. La media es inferior porque, como podemos ver en la tabla, el tiempo mínimo para que algunos usuarios obtuvieran una respuesta fue de sólo 216 ms.

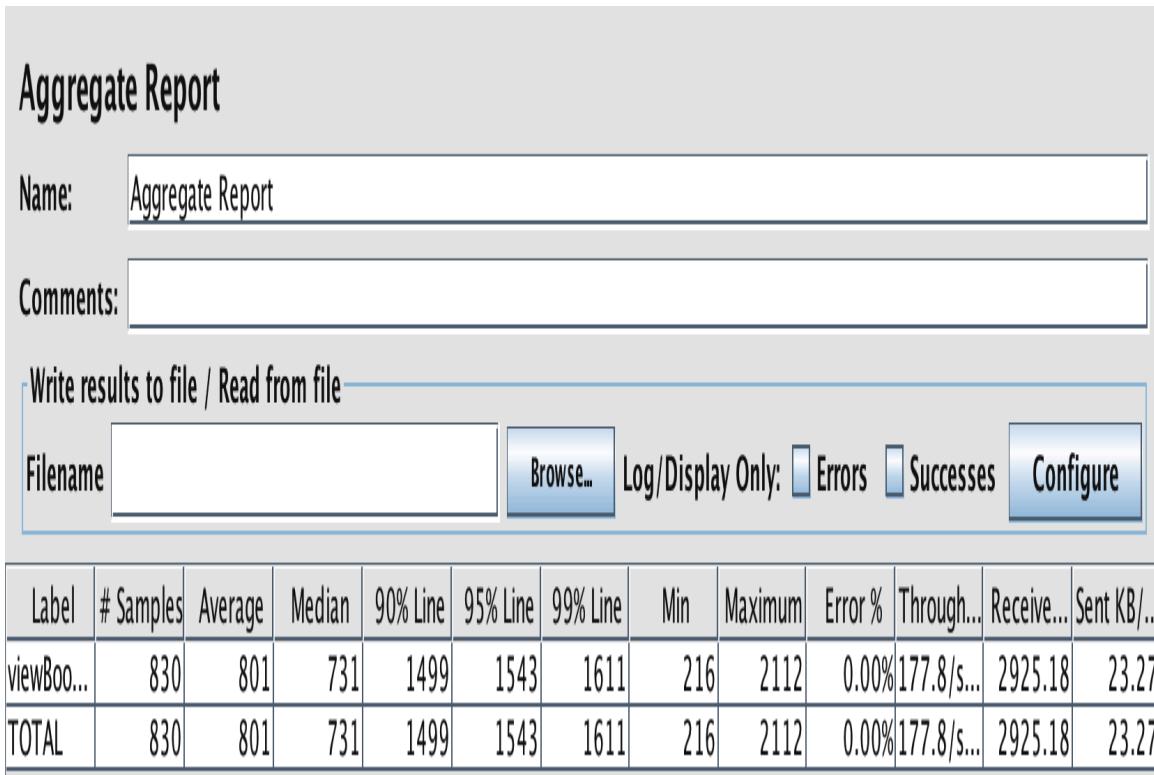


Figura 8-10. Resultados de la prueba de volumen para el punto final /books

Diseñar otros casos de pruebas de rendimiento

En la sección anterior, exploraste diferentes formas de distribuir la carga utilizando JMeter. Esta experiencia de uso de la herramienta para pruebas de volumen es un gran comienzo y debería permitirte simular otros casos de pruebas de rendimiento, como pruebas de estrés, pruebas de remoho, validación de rendimiento, etc. Para realizar pruebas de estrés, puedes utilizar un Grupo de Hilos de Concurrencia para introducir la carga en pasos de x usuarios hasta un límite máximo, ejecutando cada paso durante un tiempo determinado. El objetivo es encontrar la carga a partir de la cual el

tiempo de respuesta se ralentiza y, en última instancia, se producen errores.

Para hacer soak testing, puedes simular una carga constante durante un tiempo prolongado utilizando un Grupo de Hilos Ultimate. Para validar el rendimiento por hora, utiliza el **complemento Controlador Paralelo** para ejecutar varias solicitudes HTTP en paralelo, haciendo una pausa entre las solicitudes mediante componentes Temporizador, por ejemplo, para fijar el tiempo de reflexión. También hay un temporizador de Rendimiento constante, que puede utilizarse para fijar el rendimiento en un valor constante y validar si la aplicación funciona como se espera; ralentiza automáticamente el número de peticiones realizadas al servidor por JMeter si supera el valor de rendimiento establecido.

Hay muchos más componentes en JMeter para ayudar a modelar casos de uso específicos de la aplicación. Los controladores If, Loop y Random te permiten incluir condiciones en las pruebas. También hay disposiciones para introducir credenciales de usuario, si la aplicación requiere un inicio de sesión, desde una fuente externa como un archivo CSV para realizar pruebas de volumen. Esto se denomina *prueba de rendimiento basada en datos*. También puedes utilizar esta función de JMeter para configurar los datos de la prueba al principio de la prueba. A continuación veremos un ejemplo.

Pruebas de rendimiento basadas en datos

Digamos que el endpoint /addBook en la aplicación de la biblioteca toma un cuerpo de petición con el nombre del libro, el autor, el idioma y el ISBN. Para crear carga en este endpoint, necesitas añadir libros únicos con cada solicitud. Para ello, puedes utilizar las capacidades de pruebas de rendimiento basadas en datos de JMeter, como se indica a continuación:

1. Crea un archivo CSV con name, author, language, y isbn como claves. JMeter hace referencia a estas claves al definir las

variables de entrada. Añade filas para 50 libros. (Puedes hacerlo en Google Sheets y descargarlo como archivo CSV).

2. En JMeter, añade un grupo de hilos con un muestreador de solicitudes HTTP para el endpoint /addBook y establece el Recuento de bucles en 50.
3. Para conectar el archivo CSV al muestreador de peticiones HTTP, haz clic con el botón derecho del ratón en Grupo de hilos y selecciona Añadir → Elemento de configuración → Config. conjunto de datos CSV. En la ventana Config. conjunto de datos CSV (ver **Figura 8-11**), especifica la ruta del archivo CSV y las variables que se leerán del archivo.

CSV Data Set Config

Name:	CSV Data Set Config
Comments:	
Configure the CSV Data Source	
Filename:	/pathToInputFile/BooksTestData - Sheet1.csv
File encoding:	
Variable Names (comma-delimited):	name,author,language,isbn
Ignore first line (only used if Variable Names is not empty):	False
Delimiter (use '\t' for tab):	,
Allow quoted data?:	False
Recycle on EOF ?:	True
Stop thread on EOF ?:	False
Sharing mode:	All threads

Figura 8-11. Configuración de la entrada del conjunto de datos CSV para la prueba basada en datos

4. En el cuerpo de la solicitud HTTP del endpoint /addBook, utiliza las variables como \${variable_name} como se ve en la Figura 8-12. Se puede hacer referencia a estas variables utilizando la misma \${variable_name} siempre que sea necesario en las pruebas de JMeter.

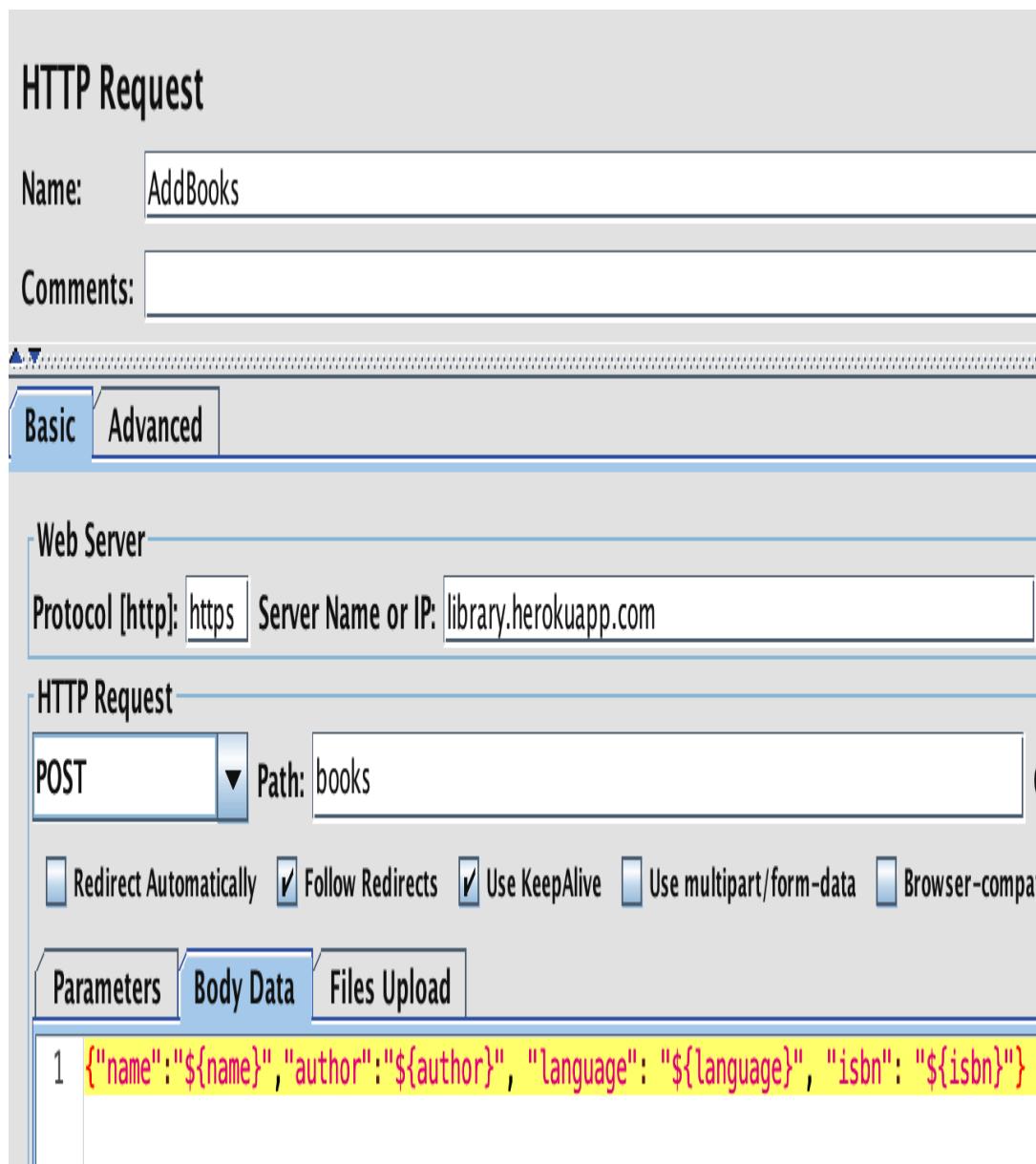


Figura 8-12. Referencia a variables del archivo CSV

Esta prueba puede ejecutarse para crear los datos de prueba antes de iniciar las pruebas de rendimiento.

Integración en la IC

El último paso consiste en integrar las pruebas de JMeter en tu canalización CI como un trabajo independiente y desplazar las pruebas de rendimiento a la izquierda. Es importante asegurarse de que las pruebas de rendimiento se ejecutan completamente aisladas

para obtener las métricas correctas. Para integrar las pruebas con CI, guárdalas, localiza los archivos *.jmx* guardados y ejecuta el siguiente comando:

```
$ jmeter -n -t <library.jmx> -l <log file> -e -o <Path to output folder>
```

También puedes configurar JMeter para que proporcione **informes exhaustivos del panel de control**, según necesites, con otras extensiones.

Como puedes ver, JMeter facilita las pruebas de rendimiento con una sencilla interfaz gráfica de usuario para configurar y ejecutar casos de pruebas de rendimiento.

Herramientas de comprobación adicionales

Existen otras herramientas de pruebas de rendimiento que pueden ayudarte a guionizar tus casos de pruebas de rendimiento. En esencia, estas herramientas proporcionan variadas asas para configurar los cuatro parámetros clave para diseñar patrones de carga (tiempo de rampa, tiempo de reflexión, número de usuarios concurrentes y ritmo). Por ejemplo, como vimos, JMeter ofrece una GUI, mientras que Gatling proporciona un lenguaje específico del dominio y Apache Benchmark (ab) utiliza simples argumentos de línea de comandos. Conozcamos también brevemente Gatling y ab.

Gatling

Gatling proporciona un DSL basado en Scala para configurar el patrón de carga. Es una herramienta de código abierto con la opción de grabar flujos de usuarios. Las pruebas pueden integrarse con pipelines CI. Si te apetece explorar Scala, ésta es una herramienta robusta para simular patrones de carga matizados. Puedes ver un

ejemplo de script Scala que demuestra cómo inducir la carga con think time en la API /books de nuestra aplicación de gestión de bibliotecas en [el Ejemplo 8-2](#).

Ejemplo 8-2. Ejemplo de script Scala para pruebas de carga

```
package perfTest

import scala.concurrent.duration._

import io.gatling.core.Predef._
import io.gatling.http.Predef._

class BasicSimulation extends Simulation {

    // Defining the HTTP request
    val httpProtocol = http
        .baseUrl("https://library.herokuapp.com/")

    .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;
q=0.8")
        .doNotTrackHeader("1")
        .acceptLanguageHeader("en-US,en;q=0.5")
        .acceptEncodingHeader("gzip, deflate")
        .userAgentHeader("Mozilla/5.0 (Windows NT 5.1; rv:31.0)
Gecko/20100101
Firefox/31.0")

    // Defining a single user flow with think time
    val scn = scenario("BasicSimulation")
        .exec(http("request_1")
            .get("/books"))
        .pause(5) // Think time

    // Configuring load of 166 concurrent users to do the above user flow
    setUp(
        scn.inject(atOnceUsers(166))
        .protocols(httpProtocol)
    )
}
```

Apache Benchmark

Si sólo quieres obtener rápidamente algunas cifras sobre el rendimiento de tu aplicación, **ab** es una gran elección. Es una sencilla herramienta de línea de comandos de código abierto . Si utilizas un Mac, ab forma parte del sistema operativo, por lo que ni siquiera tienes que preocuparte de instalarlo. Para obtener cifras de rendimiento para probar la carga del punto final /books con 200 usuarios simultáneos, puedes ejecutar el siguiente comando desde tu terminal:

```
$ ab -n 200 -c 200 https://library.herokuapp.com/books
```

Los resultados serán los siguientes:

```
Concurrency Level:        200
Time taken for tests:   5.218 seconds
Complete requests:      200
Failed requests:         0
Total transferred:      1389400 bytes
HTML transferred:       1340800 bytes
Requests per second:    38.33 [#/sec] (mean)
Time per request:       5217.609 [ms] (mean)
Time per request:       26.088 [ms] (mean, across all concurrent
                         requests)
Transfer rate:          260.05 [Kbytes/sec] received
```

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	869	2074	97.6	2064
Processing:	249	1324	299.4	1303
Waiting:	249	1324	299.5	1303
Total:	1192	3398	354.3	3370

Percentage of the requests served within a certain time (ms)

50%	3370
66%	3483
75%	3711

80%	3776
90%	3863
95%	3889
98%	4016
99%	4022
100%	4027 (longest request)

Ahora ya tienes una idea de cómo las distintas herramientas pueden ayudarte a programar casos de pruebas de rendimiento y a medir los KPI del servidor. Recuerda, sin embargo, que éste no es el final del ciclo de pruebas de rendimiento. Si encuentras problemas de rendimiento, tendrás que depurar, ajustar y volver a probar.

Aquí hemos tratado con cierta profundidad las pruebas de rendimiento del backend, pero aún no hemos terminado. A continuación, nos centraremos en las pruebas de rendimiento del frontend.

Bloques de construcción de las pruebas de rendimiento del frontend

Aunque las herramientas de pruebas de rendimiento permiten a imitar el comportamiento de la aplicación en horas punta, existe un desfase entre las cifras de rendimiento medidas y el rendimiento real experimentado por el usuario. Esto se debe a que las herramientas no son navegadores reales, y no hacen todas las tareas que hace un navegador típico!

Para entender este vacío, exploremos un poco el comportamiento del navegador. Como vimos en el [Capítulo 6](#), hay tres partes en el código del frontend que se muestra en el navegador:

- el código HTML, que es la estructura básica del sitio web
- Código CSS, que da estilo a la página
- Scripts para crear lógica en la página

Un navegador típico descarga primero todo el código HTML del servidor, luego descarga la hoja de estilos, las imágenes, etc. y comienza a ejecutar los scripts, según la secuencia del HTML. Existe paralelización hasta cierto punto, como cuando descarga imágenes de diferentes hosts. Pero el navegador detiene por completo el procesamiento paralelo cuando ejecuta un script, ya que es posible que el script cambie por completo la forma en que se hace visible la página. Como puede haber scripts al final del HTML, la página sólo se hace visible para el usuario cuando se ha ejecutado completamente todo el documento.

Las herramientas de pruebas de rendimiento no realizan la mayoría de estas tareas. Golpean directamente la página y obtienen el código HTML, pero no renderizan la página mientras ejecutan las pruebas de rendimiento. Por tanto, aunque hayas medido el tiempo de respuesta de los servicios en milisegundos, el usuario final sólo verá aparecer la página tras un retraso adicional debido a las tareas de renderizado adicionales que realiza el navegador. Se estima que esta renderización del frontend supone **el 80-90%** de todo el tiempo de carga de la página, lo que resulta chocante, ¿no?

Por ejemplo, si navegas a la [página de inicio de la CNN](#), el navegador realizará 90 tareas antes de que te aparezca la página. La [Figura 8-13](#) muestra las 33 primeras de estas tareas. Si pensabas que optimizar sólo el tiempo de respuesta del servicio web tendría un impacto significativo en el rendimiento del sitio web, ihe aquí una prueba que puede cambiar esa opinión!

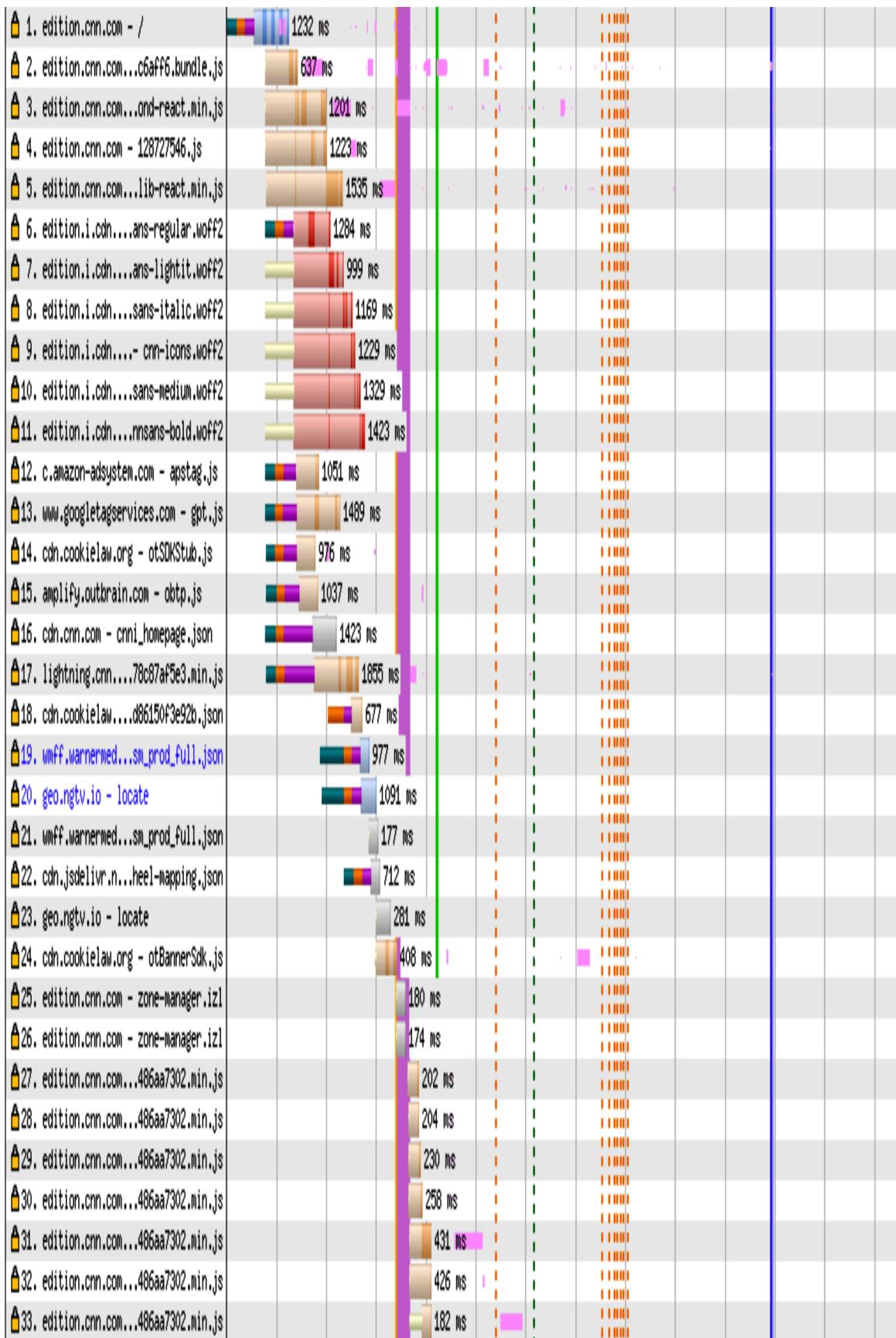


Figura 8-13. Tareas del navegador frontend de CNN durante la carga de la página

Sin embargo, los KPI descritos y medidos como parte de los ejercicios realizados anteriormente siguen siendo relevantes y críticos. Son vitales para planificar la capacidad del sistema y solucionar problemas de rendimiento. En otras palabras, te ayudan a responder a preguntas como "¿Soportará la aplicación un pico de carga de 5.000 transacciones durante las rebajas del Viernes Negro?". Pero si los KPI indican que el tiempo de respuesta máximo de tu aplicación es de ~1,5 segundos, puede que eso no refleje la experiencia real de un usuario final. Para comprenderlo, debes evaluar también las métricas de rendimiento del frontend. Eso es lo que trataremos en esta sección.

Para empezar, vamos a entender los factores que afectan específicamente al rendimiento del frontend y las métricas que hay que medir para cuantificarlo. Más adelante, te pondrás manos a la obra para medirlos realmente.

Factores que afectan al rendimiento del Frontend

Hay varios factores que contribuyen al rendimiento del frontend:

Complejidad del código frontend

Si no se aplican buenas prácticas como minificar el JavaScript, reducir el número de solicitudes HTTP realizadas por página e implementar técnicas adecuadas de almacenamiento en caché, el rendimiento será menor. Por ejemplo, el servidor tarda al menos unos milisegundos en responder a cada solicitud HTTP, y si la página tiene que hacer muchas de esas solicitudes, los retrasos se acumularán.

Redes de distribución de contenidos (CDN)

Una CDN es un conjunto de servidores alojados en múltiples ubicaciones cuyo objetivo es entregar contenidos web, como

imágenes, a los usuarios de forma más eficiente. Como hemos comentado antes, la geolocalización del servidor y del usuario influye en el rendimiento de la aplicación, debido a la latencia de la red. Para reducir la latencia de la red, el contenido se almacena en CDNs y se sirve desde el servidor que está físicamente más cerca del usuario. Esto es mucho más sencillo que replicar la aplicación en diferentes geolocalizaciones; sin embargo, el rendimiento de la propia CDN afectará al tiempo de carga de la página.

Consultas DNS

Normalmente, un navegador tarda entre 20 y 120 ms en buscar la dirección IP de un nombre de host determinado. Esto se conoce como resolución del Servicio de Nombres de Dominio (DNS). Una vez hecho esto la primera vez, el navegador y el sistema operativo almacenan en caché la dirección IP, reduciendo el tiempo de carga de la página en visitas posteriores. Los proveedores de servicios de Internet (ISP) también almacenan en caché las direcciones IP durante un tiempo, lo que contribuye a mejorar el rendimiento. Sin embargo, la experiencia del usuario por primera vez se ve afectada por el tiempo de búsqueda del DNS.

Latencia de la red

El ancho de banda de la red del usuario tiene un gran impacto en el tiempo total de carga de la página. Como vimos en el [Capítulo 6](#), los datos de uso global indican que el uso de móviles supera al de ordenadores de sobremesa hoy en día, y el ancho de banda de la red móvil puede ser muy bajo en ocasiones, tanto en zonas urbanas como rurales. Algunos sitios superan esto sirviendo una versión "lite" de su sitio web cuando identifican que el ancho de banda es bajo. Sin embargo, los usuarios que suelen operar con poco ancho de banda (como 3G) tienden a acostumbrarse a la

lentitud y a no quejarse a menos que el rendimiento sea escandalosamente malo.

Caché del navegador

Además de las direcciones IP, el navegador almacena en caché muchos otros contenidos (imágenes, cookies, etc.) después de la primera visita. En consecuencia, el tiempo de carga de la página suele variar significativamente entre la primera vez que se visualiza y las siguientes. El almacenamiento en caché del navegador puede hacerse intencionado mediante código para mejorar los tiempos de carga de la página.

Transferencias de datos

Si se transfieren grandes volúmenes de datos de un lado a otro entre el usuario y la aplicación, obviamente esto afectará al rendimiento general del frontend debido a la latencia de la red.

Viendo todos estos factores, podrías pensar que está fuera del control del equipo siquiera pensar en optimizarlos, lo que te deja preguntándote por dónde empezar. Mucha gente de la industria del software también ha sentido el dolor de enfrentarse a este reto. Ahí es donde entra en juego el modelo RAIL.

Modelo RAIL

El **modelo RAIL** es una forma de estructurar en el proceso de pensamiento en torno al rendimiento del frontend. Está diseñado con el principio rector de mantener la experiencia del usuario final en el centro del rendimiento del frontend, y cuantifica los objetivos para el rendimiento del frontend. Puede ser útil ver el rendimiento del frontend a través de esta lente e integrar los objetivos como parte de tus pruebas.

El modelo RAIL desglosa la experiencia de un usuario en un sitio web en cuatro áreas clave:

Respuesta

¿Alguna vez has tenido la experiencia de pulsar un botón pero no ver ninguna indicación visual inmediata de que lo has hecho, lo que te hace preguntarte si te habías imaginado pulsándolo en primer lugar? Este retraso se conoce como *latencia de entrada*. El aspecto "respuesta" de RAIL define los objetivos de la latencia de entrada. Cuando un usuario realiza una acción en un sitio web, como pulsar un botón, alternar un elemento, seleccionar una casilla de verificación, etc., RAIL prescribe que el tiempo de respuesta para esa acción debe ser inferior a 100 ms; de lo contrario, el usuario percibirá el retraso!

Animación

Del mismo modo, el usuario percibirá un retraso en los efectos de animación (por ejemplo, indicadores de carga, desplazamiento, arrastrar y soltar, etc.) cuando cada fotograma no se complete en 16 ms (el mínimo para alcanzar una velocidad de fotogramas de 60 FPS).

Ralentí

Un patrón general de diseño del frontend es para agrupar tareas no críticas, como el envío de datos analíticos, el arranque de un cuadro de comentarios, etc., y realizarlas más tarde, cuando el navegador esté inactivo. Lo ideal es agrupar estas tareas en bloques que tarden unos 50 ms en completarse, de modo que cuando el usuario vuelva a interactuar, puedas responder dentro de la ventana de 100 ms.

Carga

Un sitio web de alto rendimiento debería tener como objetivo empezar a mostrar la página en 1 segundo, ya que sólo entonces

los usuarios sentirán que controlan completamente la navegación (según la investigación mencionada anteriormente).

Como puedes ver, el modelo RAIL nos guía a la hora de pensar qué debemos comprobar desde el punto de vista del rendimiento del frontend. También proporciona un lenguaje concreto para la comunicación dentro de los equipos, en lugar de expresar sentimientos vagos como "la página parece lenta"!

Métricas de rendimiento del frontend

En la práctica, los objetivos de alto nivel establecidos por el modelo RAIL se desglosan en métricas más pequeñas para afinar la depuración de los problemas de rendimiento. Un conjunto de métricas estándar de rendimiento del frontend adoptadas en la industria son las siguientes:

Primera pintura de contenido

Se refiere al tiempo que tarda el navegador en renderizar el primer elemento del DOM (imágenes, elementos no blancos, SVG, etc.). Esto nos ayuda a comprender cuánto tiempo tiene que esperar el usuario para ver alguna acción en el sitio web después de abrirlo.

Hora de interactuar

Es el tiempo que tarda la página en volverse interactiva. Con las prisas por hacer que la página rinda, los elementos podrían hacerse visibles rápidamente, pero podrían no responder a las acciones del usuario, lo que provocaría frustración. Por tanto, paralelamente a la medición del tiempo que se tarda en ver el primer contenido de la página, esta métrica nos ayuda a comprender si la información presentada es útil o sólo ruido.

Pintura de mayor contenido

Es el tiempo que tarda en hacerse visible el elemento más destacado de la página web, como una gran mancha de texto o una imagen.

Cambio de disposición acumulativo

¿Te has encontrado alguna vez con sitios en los que has empezado a leer un artículo y luego la página se ha desplazado automáticamente hacia abajo a medida que se cargaba contenido adicional, haciéndote perder la pista de lo que estabas leyendo? Es frustrante, ¿verdad? Esta métrica pretende medir la estabilidad visual de la página y cuantifica la frecuencia con la que el usuario se enfrenta a un cambio inesperado en el diseño de la página. Cuanto menor sea el número, mejor será el rendimiento.

Primer retardo de entrada

Entre el primer pintado del contenido y el tiempo hasta la interacción, cuando el usuario hace clic en un enlace o realiza cualquier interacción con la página web, habrá un retraso mayor que el habitual porque la página aún se está cargando. Esta métrica da ese tiempo de retardo para la primera interacción.

Retardo máximo potencial de la primera entrada

Representa el peor escenario posible del retraso de la primera entrada. Mide el tiempo que tarda la tarea más prolongada que se produce entre el primer pintado de contenido y el tiempo de interactivo en completarse.

Google clasifica la mayor pintura de contenido, el retardo de la primera entrada y el desplazamiento acumulativo del diseño como los *principales indicadores vitales de la web* para ayudar a la gente de negocios a entender el rendimiento de un sitio en términos sencillos. La mayoría de las herramientas de comprobación del

rendimiento del frontend capturan específicamente estas tres métricas. Podemos utilizar dichas herramientas para medir continuamente estas métricas como parte de CI y, por tanto, desplazar las pruebas de rendimiento del frontend hacia la izquierda. A continuación veremos cómo hacerlo.

Ejercicios

Como se desprende del modelo RAIL, el rendimiento del frontend tiene que ver con la experiencia del usuario final. Por tanto, para medir las métricas de rendimiento del frontend de tu aplicación, primero tienes que definir un conjunto de casos de prueba que abarquen todas las experiencias de tus usuarios finales objetivo en diferentes grupos demográficos. Por ejemplo:

- Ten en cuenta a los usuarios con distintos tipos de dispositivos (ordenador de sobremesa, móvil, tableta). Recopila también información sobre los fabricantes de dispositivos que son actores importantes en la región a la que tu aplicación quiere servir. Esto es importante porque cada dispositivo tendrá su propia CPU, batería y capacidad de memoria, lo que afecta a la experiencia del usuario final.
- Ten en cuenta a los usuarios con distintos anchos de banda de red: WiFi, 3G, 4G, etc. Además, ten en cuenta que las velocidades medias de los móviles y la banda ancha son diferentes en los distintos países. Según [los datos del World Population Review](#), por ejemplo, en 2021 Mónaco tenía la velocidad media de banda ancha más rápida, con 261,8 Mbps, frente a los 203,8 Mbps de EE.UU., los 102,2 Mbps del Reino Unido y los 13,8 Mbps de Pakistán.
- Ten en cuenta la distribución de tus usuarios objetivo. Como sugiere el punto anterior (aunque hay otros factores que también contribuyen), habrá que probar específicamente el

rendimiento del frontend experimentado en diferentes geolocalizaciones.

En Internet hay disponible mucha investigación sobre este tipo de datos de uso. Alternativamente, si tienes una aplicación en vivo, Google Analytics te dará la información de uso del sitio en tiempo real. Una vez que tengas los casos de prueba, puedes utilizar las herramientas descritas aquí para medir las métricas de rendimiento del frontend y también añadir esas pruebas a tu canal de CI.

Vamos a definir un caso de prueba de ejemplo para hacer aquí los ejercicios prácticos: "Un usuario de Milán, que tiene un Samsung Galaxy S5, accede a la página de inicio de Amazon utilizando una conexión de red 4G". Ahora, veamos cómo herramientas como WebPageTest y Lighthouse pueden ayudar a medir el rendimiento del frontend.

WebPageTest

WebPageTest es una herramienta online gratuita para evaluar el rendimiento del frontend de un sitio web. Es una herramienta potente, ya que incluye disposiciones para elegir la geolocalización desde la que se accede al sitio web y recopila las métricas de rendimiento del frontend renderizando el sitio web en navegadores web y móviles reales. Una herramienta no puede estar mucho más cerca de replicar el comportamiento de un usuario final real.

Flujo de trabajo

Utilizar la herramienta es sencillo, como muestran los pasos que se indican a continuación:

1. Introduce la URL de Amazon en el campo de entrada, como se ve en [la Figura 8-14](#).
2. Elige la ubicación del usuario final, el tipo de navegador, el tipo de dispositivo móvil y el ancho de banda de la red, según el

caso de prueba de ejemplo. Consulta [la Figura 8-14](#).

3. Establece el parámetro Número de pruebas a ejecutar en 3. Los resultados de una sola ronda de evaluación pueden ser defectuosos debido a fallos en el ancho de banda de la red, por lo que es una buena idea ejecutar el caso de prueba varias veces para observar la media.
4. Establece el parámetro Vista repetida en "Primera vista y Vista repetida". Esto capturará las métricas de rendimiento por separado para la primera visita y las visitas posteriores. Como recordarás, las métricas podrían variar para las visitas posteriores debido al almacenamiento en caché.
5. Ejecuta la prueba y visualiza los informes con las métricas.

<https://www.amazon.com/>

Start Test →

Test Location

Milan, Italy - EC2 (Chrome,Firefox)

 Select from Map

Browser

Samsung Galaxy S5

Advanced Settings ▾

Test Settings

Advanced

Chromium

Script

Block

SPOF

Custom

Connection

4G (9 Mbps, 170ms RTT)

Desktop Browser Dimensions

default (1366x768)

Number of Tests to Run

Up to 9

3

Repeat View

First View and Repeat View

First View Only

Capture Video



Figura 8-14. Configuración de WebPageTest

Como WebPageTest es una herramienta gratuita y disponible públicamente, es posible que tengas que esperar en la cola unos minutos para ver el informe. Para evitar la espera, puedes optar por configurarlo de forma privada en un entorno de prueba local pagando una tarifa.

El informe tiene muchas secciones valiosas que permiten una depuración detallada. Cada informe puede recuperarse utilizando un ID único durante 30 días. Analicemos un par de secciones importantes del informe generado por WebPageTest para nuestro caso de prueba de ejemplo.

La tabla de métricas de rendimiento (ver [Figura 8-15](#)) tiene los valores vitales de la web para la primera vista y la vista repetida de las tres ejecuciones de la prueba. Para evaluar el tiempo de carga de la página en este caso de prueba, puedes tomar la mediana del tiempo de documento completo de todas las ejecuciones. Observa que el tiempo completo del documento de la primera vista es de 3,134 s y el de la mayor pintura de contenido es de 2,105 s, lo que nos indica que la experiencia del usuario está dentro de unos límites aceptables. El tiempo de carga completa de la tabla incluye el tiempo empleado en cargar todo el contenido secundario, es decir, las tareas aplazadas por el evento de carga. Aunque es considerable (~14 s con 230 peticiones), es poco probable que afecte a la experiencia del usuario final.

Performance Results (Median Run - SpeedIndex)

	First Byte	Start Render	First Contentful Paint	Speed Index	Web Vitals			Document Complete			Fully Loaded			
					Largest Contentful Paint	Cumulative Layout Shift	Total Blocking Time	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View (Run 1)	0.918s	2.000s	1.994s	2.505s	2.105s	0.156	0.162s	3.134s	38	406 KB	14.615s	230	1,154 KB	\$\$\$-
Repeat View (Run 1)	1.156s	2.100s	2.085s	2.577s	2.316s	0.142	0.050s	3.048s	9	116 KB	13.502s	127	127 KB	

Figura 8-15. Tabla de métricas de rendimiento del informe WebPageTest

La vista en cascada, que se muestra en la Figura 8-16, muestra una colorida vista cronológica del tiempo que tarda cada tarea -como la resolución DNS, el inicio de la conexión, la descarga de HTML e imágenes, el tiempo de ejecución de los scripts, etc.-, lo que nos da una pista sobre las vías para una mayor optimización.

Waterfall View

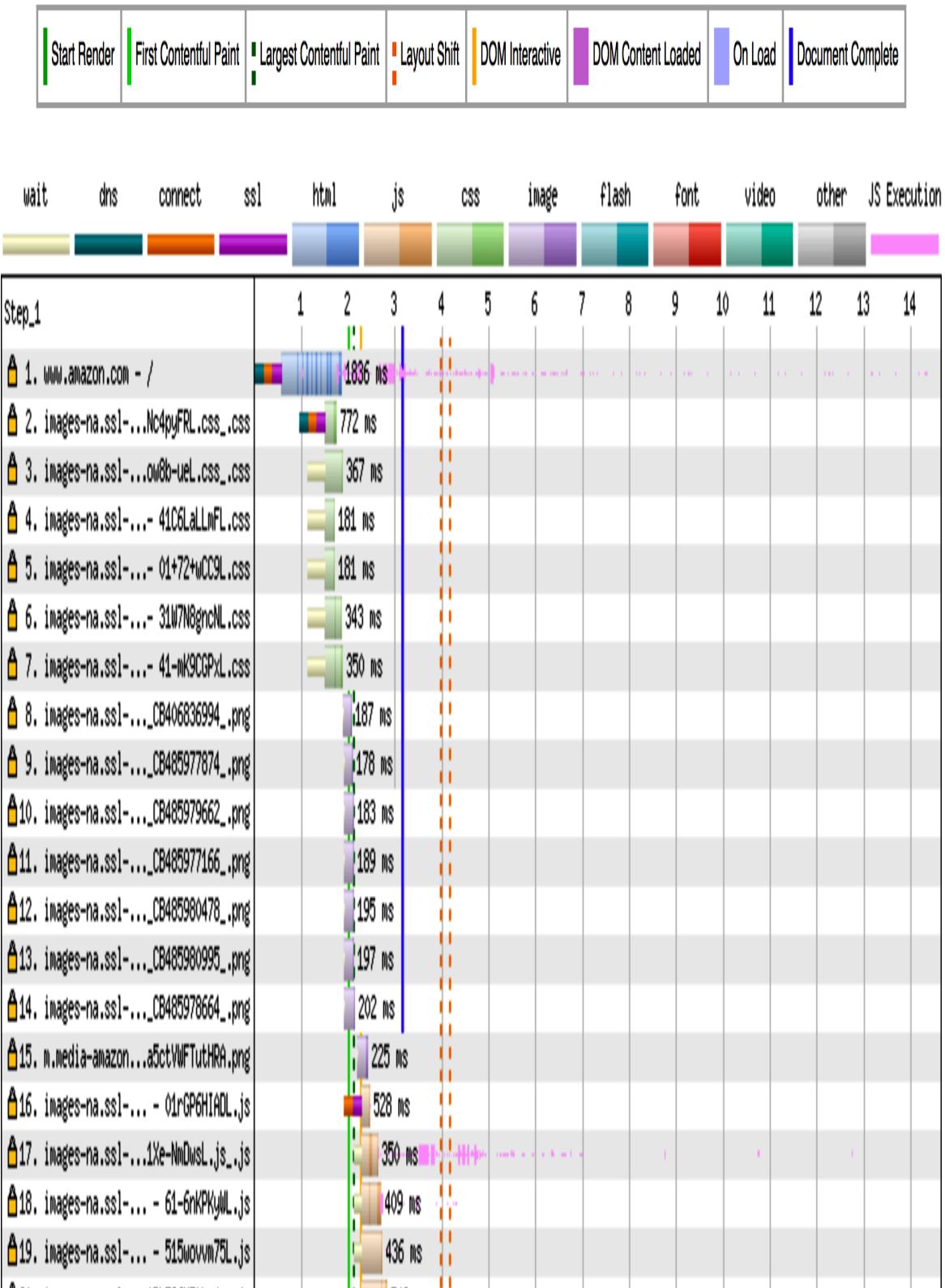


Figura 8-16. Vista en cascada del informe WebPageTest

WebPageTest también permite pasar datos de autenticación, pero ten en cuenta que las credenciales de prueba que proporcionas serán visibles para cualquiera que tenga acceso al informe de ejecución de la prueba, ya que está alojado públicamente.

WebPageTest también expone API para obtener estos informes mediante programación, y existe una variante del módulo Node.js para ejecutar las pruebas también directamente desde la línea de comandos. Estas dos opciones permiten la integración con canalizaciones CI. Ambas requieren una clave API que está disponible previo pago. Si decides adquirir una, consulta los Ejemplos 8-3 y 8-4 para obtener información detallada sobre los comandos CLI y el uso de la API, respectivamente.

Ejemplo 8-3. Comandos CLI de WebPageTest para instalar, ejecutar casos de prueba y ver los resultados

```
// Step 1: Install using npm

npm install webpagetest -g

// Step 2: Run a sample test case via the command line

webpagetest test http://www.example.com --key API_KEY --
location
ec2-eu-south-1:Chrome --connectivity 4G --device Samsung
Galaxy S5 --runs 3 --first --video --label
"Using WebPageTest" --timeline

// Step 3: Read test results from the report ID generated by the above
command

webpagetest results 2345678
```

Ejemplo 8-4. API para ejecutar casos de prueba WebPageTest y ver los resultados

```
// Step 1: Run a sample test case using WebPageTest's API

http://www.webpagetest.org/runtest.php?url=http%3A%2F%2Fwww.example.com&k=API\_KEY&location=ec2-eu-south-1%3AChrome&connectivity=4G&runs=3&fonly=1&video=1&label=Using%20WebPagetest&timeline=1&f=json

// Step 2: Read test results from the report ID returned as the response
// by the above API

http://www.webpagetest.org/jsonResult.php?test=2345678
```

Faro

Lighthouse forma parte de Google Chrome, y también está disponible como extensión para Firefox. Audita tu sitio web en múltiples dimensiones, como la seguridad, la accesibilidad y el rendimiento del frontend. El informe de auditoría de rendimiento generado por Lighthouse incluye una puntuación global y todas las métricas detalladas de rendimiento del frontend.

Una de las ventajas de Lighthouse es que no está alojado públicamente, por lo que no hay colas ni tiempos de espera. Como se ejecuta en tu navegador local, tampoco hay problemas de seguridad, aunque eso también significa que no puedes configurar la geolocalización del usuario final (accederás al sitio web desde tu ubicación real). No obstante, puedes estrangular tu red y CPU y cambiar el tamaño a diferentes resoluciones de navegador móvil en Chrome para simular diferentes casos de prueba y obtener las métricas respectivas.

Lighthouse también está disponible como herramienta CLI, lo que facilita la integración con CI y la obtención de feedback continuo. Zalando, una importante cadena minorista europea, ha declarado

que redujo el tiempo de retroalimentación del rendimiento de su frontend de 1 día a **15 minutos** con Lighthouse CI. La herramienta es totalmente gratuita y de código abierto.

Flujo de trabajo

Puedes seguir estos sencillos pasos para explorar Lighthouse:

1. Abre el sitio web de Amazon en Chrome.
2. Abre Chrome DevTools utilizando el atajo de teclado Cmd-Opción-J en macOS o Mayúsculas-Ctrl-J en Windows/Linux, o elige la opción Inspeccionar en el menú contextual de Chrome.
3. Elige las preferencias de estrangulamiento de red en la pestaña Red. Selecciona "3G lento" para el caso de prueba de ejemplo.
4. Elige las preferencias de ralentización de la CPU en la pestaña Rendimiento. Las opciones por defecto son ralentización 4x y ralentización 6x para dispositivos móviles de nivel medio y bajo. Elige ralentización 4x para el caso de prueba de ejemplo.
5. Elige el tamaño de la ventana en el desplegable de respuesta. La opción Galaxy S5 del caso de prueba de ejemplo también está disponible, como se ve en la **Figura 8-17**.
6. En la pestaña Lighthouse(**Figura 8-17**), selecciona la categoría Rendimiento y haz clic en "Generar informe".

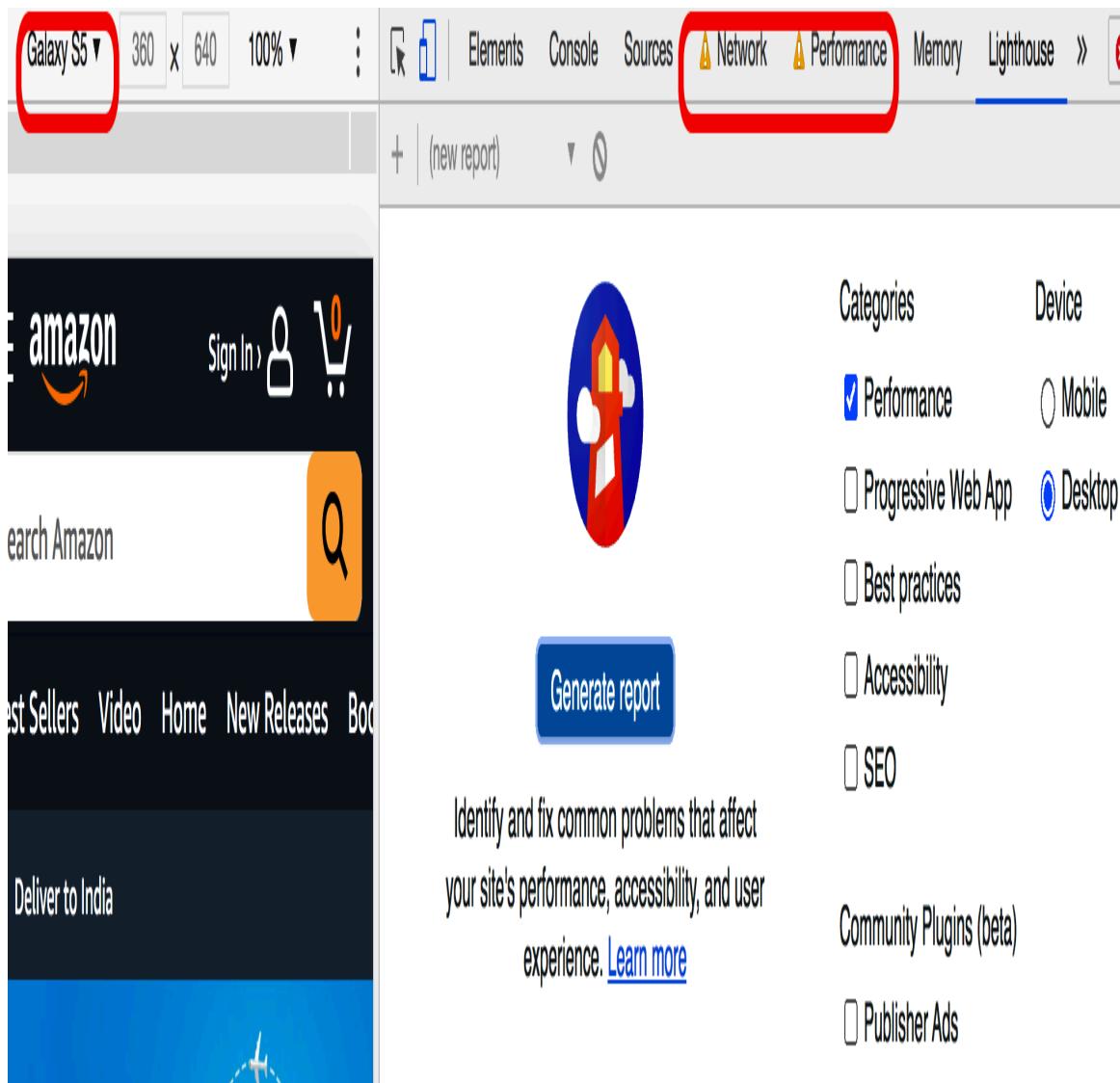


Figura 8-17. Ventana de Lighthouse con las configuraciones de red, CPU y resolución

Como se ve en la Figura 8-18, los resultados nos indican que Amazon hace un trabajo bastante bueno. El tiempo hasta la métrica interactiva, incluso en condiciones tan sesgadas, es de i3,8 s!



Performance

Metrics



■ First Contentful Paint	1.1 s	■ Time to Interactive	3.8 s
--------------------------	-------	-----------------------	-------

▲ Speed Index	3.6 s	▲ Total Blocking Time	420 ms
---------------	-------	-----------------------	--------

▲ Largest Contentful Paint	3.8 s	● Cumulative Layout Shift	0.036
----------------------------	-------	---------------------------	-------

Values are estimated and may vary. The [performance score is calculated](#) directly from these metrics. [See calculator.](#)

[View Original Trace](#)



Figura 8-18. Informe de rendimiento de Lighthouse

Puedes utilizar Lighthouse para probar varios casos de pruebas de rendimiento del frontend ya durante el propio desarrollo. Para integrarlo con CI, puedes utilizar el módulo Lighthouse Node.js. Para instalarlo, ejecuta el siguiente comando desde tu terminal:

```
$ npm install -g lighthouse
```

Para ejecutar una auditoría de rendimiento, ejecuta el siguiente comando:

```
$ lighthouse https://www.example.com/ --only-categories=performance
```

Puedes enviar **parámetros opcionales** junto con este comando para establecer los valores de estrangulamiento de la red y la CPU y seleccionar los tamaños de pantalla de los dispositivos. El informe de auditoría estará disponible en el directorio actual. Puedes escribir una envoltura para que falle la canalización si la puntuación de rendimiento es inferior a un umbral; por ejemplo, puede que quieras que falle la compilación si la puntuación es inferior a 90. También puedes definir presupuestos de rendimiento (valores umbral superiores) para cada uno de los vitales de la web mediante la función **LightWallet**. Esto comparará los resultados de rendimiento de Lighthouse con los valores umbral definidos para cada métrica y emitirá alertas cuando se superen.

Otra forma de integrarse con CI es mediante la herramienta **cypress-audit**. Integra Lighthouse con Cypress, mediante la cual puedes ejecutar las auditorías de rendimiento como parte de tus pruebas funcionales en el CI.

Herramientas de comprobación adicionales

Hay un par de herramientas más que ayudan de distintas formas a medir y depurar el rendimiento del frontend. En esta sección exploraremos algunas de las características de PageSpeed Insights y Chrome DevTools.

PageSpeed Insights

Las herramientas que utilizamos en los ejercicios anteriores nos permiten simular casos de prueba como en un laboratorio, donde establecemos condiciones previas y observamos los resultados. Pero en la vida real puede haber muchas variaciones matizadas debidas a pequeñas diferencias en el ancho de banda de la red de los usuarios, las configuraciones de los dispositivos, etc., que no pueden predecirse ni medirse. La única forma de saber cómo experimentan realmente los distintos usuarios el rendimiento del sitio web es mediante el monitoreo del usuario real (RUM) después de que la aplicación haya entrado en funcionamiento. Google proporciona servicios gratuitos de monitoreo que registran los datos vitales básicos de la web junto con otras métricas a medida que los usuarios de todo el mundo acceden a la aplicación en directo. Estos datos se denominan *datos de campo* o *datos RUM*.

La herramienta PageSpeed Insights intenta ofrecer una visión holística del rendimiento del frontend presentando los datos de RUM junto con los datos de laboratorio producidos por Lighthouse, como se ve en la [Figura 8-19](#). Prueba esta herramienta introduciendo la URL de tu aplicación en tiempo real en la [página de inicio de PageSpeed Insights](#).

Field Data – Over the previous 28-day collection period, [field data](#) shows that this page **passes** the [Core Web Vitals](#) assessment.

● First Contentful Paint (FCP) 1.5 s ● First Input Delay (FID) 29 ms



● Largest Contentful Paint (LCP) 1.7 s ● Cumulative Layout Shift (CLS) 0.01



[Show Origin Summary](#)

Lab Data



■ First Contentful Paint 2.5 s ▲ Time to Interactive 12.7 s

▲ Speed Index 9.4 s ■ Total Blocking Time 370 ms

■ Largest Contentful Paint 3.3 s ■ Cumulative Layout Shift 0.121

Figura 8-19. Un informe de datos de campo y datos de laboratorio de PageSpeed Insights

PageSpeed Insights también expone API para monitorear y alertar constantemente.

Chrome DevTools

Otra herramienta útil para depurar el rendimiento del frontend es el **perfilador de rendimiento** disponible en la pestaña Rendimiento de Chrome DevTools. Proporciona informes de análisis detallados sobre la pila de red , la velocidad de fotogramas de las animaciones, el consumo de la GPU, la memoria, el tiempo de ejecución de los scripts, etc., para que los desarrolladores puedan ahorrar preciosos milisegundos. El perfilador también te permite estrangular la red y la CPU mientras depuras. Y como está integrado en el propio navegador, es fácil de usar para los desarrolladores.

Funciona así. Supongamos que quieras averiguar cómo funciona el desplegable autocompletado de la interfaz de usuario de tu aplicación. Puedes grabar la acción de introducir texto en el desplegable utilizando la disposición de la pestaña Rendimiento; una vez detenida la grabación, se mostrarán los informes del análisis de rendimiento en la misma pestaña, como se ve en la **Figura 8-20**.

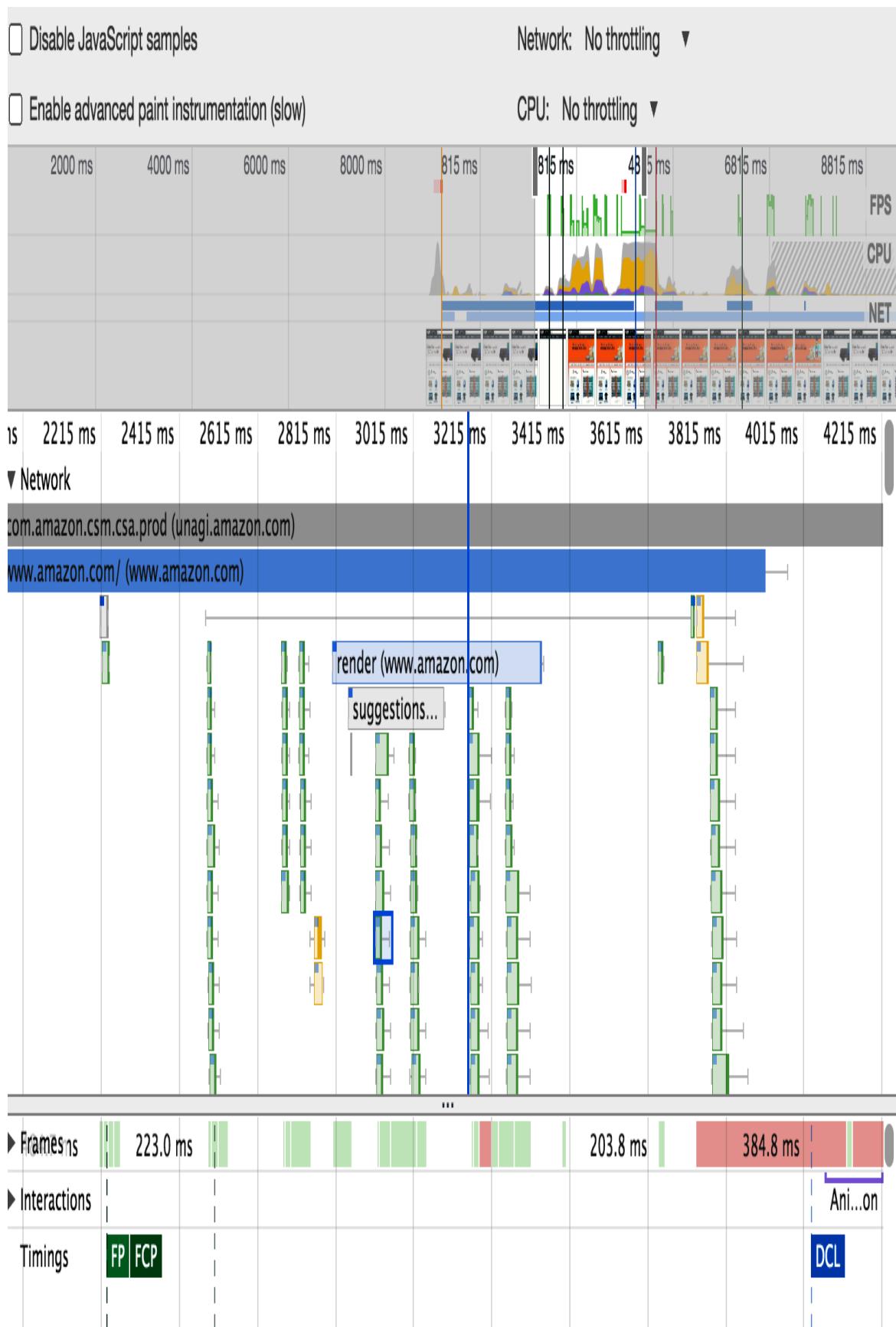


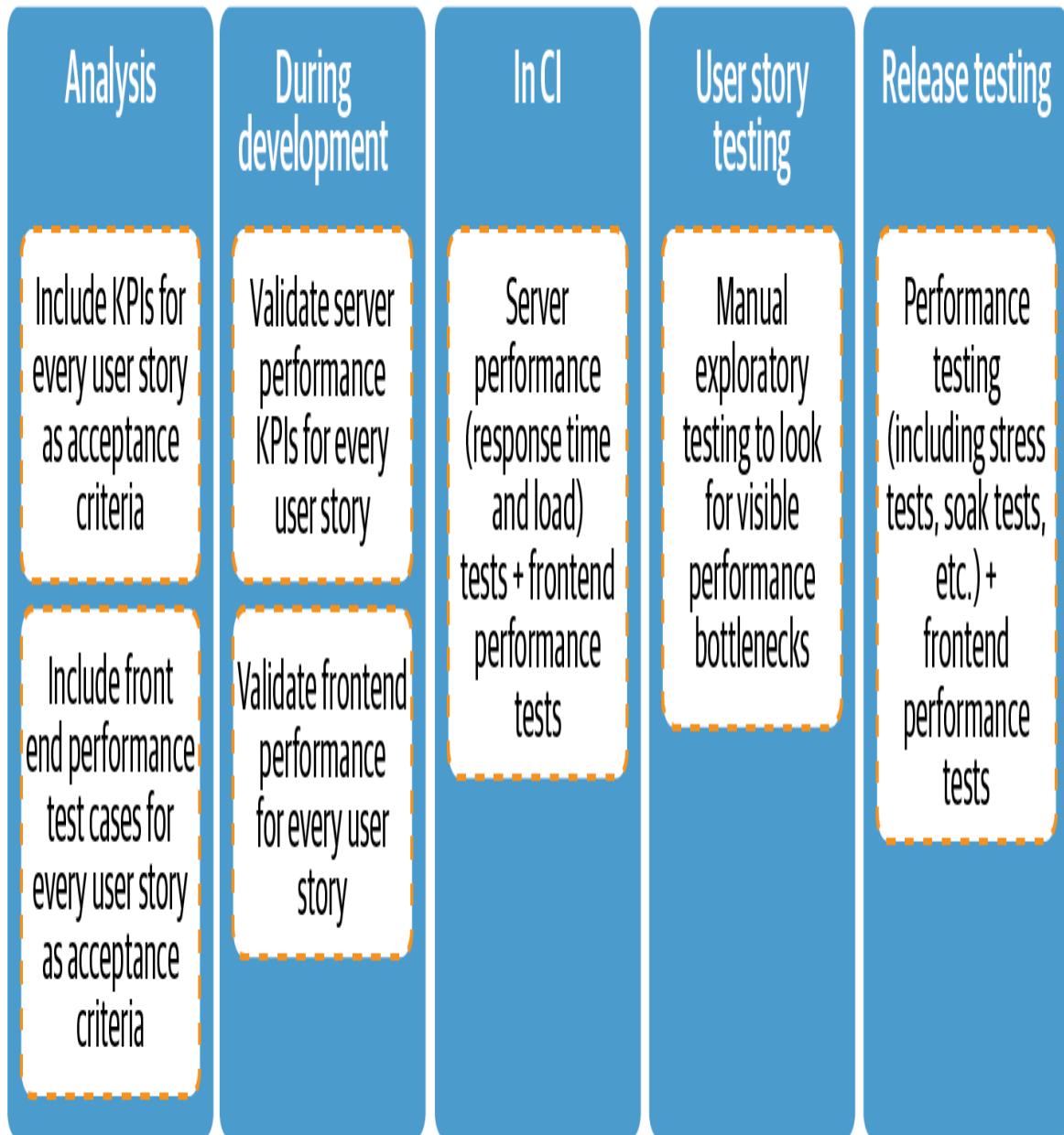
Figura 8-20. Un ejemplo de informe del perfilador de rendimiento de Chrome DevTools

Con esto, deberías estar preparado para empezar con las pruebas de rendimiento de extremo a extremo de tu aplicación. La última parte es reunir todo esto para formar una estrategia de pruebas de rendimiento, de modo que pueda planificar el tiempo y la capacidad necesarios con suficiente antelación.

Estrategia de pruebas de rendimiento

Como se ha mencionado varias veces a lo largo del capítulo, el cambio a la izquierda debe ser el principio rector de tu estrategia de pruebas de rendimiento . El desplazamiento a la izquierda debe empezar por diseñar la arquitectura de forma que se aadecue a las cifras de rendimiento esperadas y extenderse hasta integrar las pruebas de rendimiento en tus canalizaciones de CI para obtener una retroalimentación frecuente y continua, recordando cómo esto será rentable para la empresa, agradable para los usuarios finales y favorable para tus planes de fin de semana. **La Figura 8-21** muestra una visión general de una estrategia de pruebas de rendimiento por turnos que aplica los fundamentos tratados en este capítulo.

Shift-left performance testing



Path to production

Figura 8-21. Una estrategia de pruebas de rendimiento con desplazamiento a la

izquierda

Recorramos las distintas fases de en las pruebas de rendimiento de cambio-izquierda. En la fase de planificación:

- Llega a un consenso sobre los KPI de rendimiento con todas las partes interesadas en la aplicación, incluidos los responsables de negocio, marketing y técnicos, antes de que comience el proyecto. Diseña la arquitectura y elige la pila tecnológica y otros detalles basándote en estas cifras.
- Establece un entorno de pruebas de rendimiento al principio del proyecto. Si no puede ser similar al entorno de producción, al menos asegúrate de que tienes un entorno en el que empezar las pruebas.
- Incluye varios casos de prueba de rendimiento del frontend (como condiciones de red, geolocalización, etc.) como parte de los criterios de aceptación de cada historia de usuario.
- Incluye los KPI previstos (tiempo de respuesta, concurrencia y disponibilidad) de las API como parte de los criterios de aceptación de cada historia de usuario.

Durante el desarrollo:

- Valida los respectivos KPI del servidor (mediante tiempo de respuesta y pruebas de carga de los puntos finales) para cada historia de usuario.
- Valida los casos de prueba de rendimiento del frontend para cada historia de usuario.

En CI:

- Ejecuta todas las pruebas de validación del tiempo de respuesta para cada commit. Dependiendo del tiempo que tardes en ejecutar las pruebas de carga, ejecútalas en cada confirmación o como regresiones nocturnas para detectar los problemas de

rendimiento con antelación. Esto también te ayudará a ver cómo se degrada gradualmente el rendimiento a medida que añades más funciones y te ayudará a depurar más adelante.

- Incluye pruebas de rendimiento frontales para las páginas más visitadas como parte de tu canal de CI.

Durante la prueba de la historia de usuario:

- Busca cuellos de botella visibles en el rendimiento durante las pruebas exploratorias de diferentes casos de prueba.
- Asegúrate de que se cumplen los criterios de aceptación relacionados con el rendimiento, se automatizan y se integran con CI antes de marcar una historia de usuario como completa.

Y, por último, durante la fase de pruebas de lanzamiento:

- Completa las pruebas de rendimiento de la aplicación de extremo a extremo, incluidas las pruebas de estrés y las pruebas de remojo, así como las actividades de depuración. Antes de esta fase, esfuérzate por configurar un entorno de pruebas de rendimiento similar al de producción.

Como ya te habrás dado cuenta, las pruebas de rendimiento exigen un esfuerzo considerable y no pueden introducirse bruscamente en tu ciclo de publicación de como una ocurrencia tardía sin alterar los plazos.

Puntos clave

Éstos son los puntos clave de este capítulo:

- Un rendimiento web deficiente puede tener un grave impacto financiero en una empresa. Por el contrario, mejorar el rendimiento puede aumentar significativamente las conversiones y los ingresos.

- Diversos factores, como la arquitectura, el rendimiento de los servicios de terceros, el ancho de banda de la red, la geolocalización del usuario y otros, influyen en el rendimiento de una aplicación. Estos factores van cambiando a lo largo del ciclo de entrega del software y, a veces, optimizar uno sólo puede hacerse a expensas de otro, lo que plantea un duro reto a los equipos de software.
- Medir los KPI (disponibilidad, concurrencia/rendimiento y tiempo de respuesta) continuamente desde el principio del ciclo de entrega del software ayudará a prevenir problemas importantes de rendimiento en producción.
- Varias herramientas, como JMeter, Gatling y Apache Benchmark, están disponibles para realizar pruebas de rendimiento de shift-left.
- Centrarse por separado en el rendimiento del frontend es esencial, ya que el código del frontend es responsable del 80% o más del tiempo medio de carga de esa aplicación.
- El modelo RAIL de Google proporciona un marco útil para definir las métricas de rendimiento de tu frontend.
- Diseña tus casos de prueba de rendimiento del frontend teniendo en cuenta la experiencia del usuario final. Incluye distintas variables del usuario final, como el ancho de banda de la red, la geolocalización y las capacidades del dispositivo.
- Incluye pruebas de rendimiento tanto de la API como del frontend en tu canal CI, iy evita a tu equipo grandes sorpresas de rendimiento!

¹ Para más información sobre esta idea, consulta el libro blanco de Scott Barber "[Get Performance Requirements Right-Think Like a User](#)".

Capítulo 9. Pruebas de accesibilidad

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

Accesibilidad: esencial para algunos, útil para todos.

-W3C WAI

La web es un aspecto esencial de nuestras vidas en muchos sentidos: la utilizamos para comprar productos cotidianos y recibirlos en la puerta de casa, para comunicarnos con amigos y familiares, para aprender nuevas habilidades y mantenernos al día de las noticias mundiales. No puedo imaginar cuánto más difícil habría sido superar la pandemia del Covid-19 sin la conectividad, productividad e información que proporciona la web. Poner un bien tan vital a disposición de todos los usuarios con discapacidades permanentes, temporales o situacionales se denomina *accesibilidad web*. Esto incluye a las personas con deficiencias visuales, mayores, con lagunas literarias, que conducen automóviles o que se enfrentan a otras dificultades para acceder a la web. La accesibilidad es un subconjunto de la usabilidad, en términos de desarrollo web. Es un subconjunto de la inclusividad en términos humanitarios.

Aunque el objetivo principal es permitir que las personas con problemas de accesibilidad aprovechen los servicios de la web, en realidad mejora la vida de todos. Me encanta el eslogan "Esencial para algunos, útil para todos", acuñado por la **Iniciativa de Accesibilidad Web (WAI)** del W3C; hace hincapié en cómo las funciones de accesibilidad web son útiles para todos los usuarios,

independientemente de sus discapacidades u otros obstáculos. Por ejemplo, todos preferimos un diseño claro y estructurado en el que las distintas partes de la página sean fáciles de localizar e identificar y el sitio sea fácil de navegar. Del mismo modo, disponer de mensajes de error e instrucciones sencillas y comprensibles es una necesidad fundamental para todos los usuarios, y las aplicaciones habilitadas por voz están experimentando una rápida adopción en todos los segmentos de usuarios por la facilidad que proporcionan en este mundo tan cambiante.

NOTA

W3C son las siglas de World Wide Web Consortium. Es una comunidad internacional dirigida por Tim Berners-Lee, inventor de la World Wide Web, que trabaja con organizaciones miembro y usuarios públicos para establecer normas para la web. La WAI del W3C ha establecido normas globales para la accesibilidad web, de las que hablaremos en este capítulo.

Cambiando el enfoque al punto de vista empresarial, puede decirse que la comunidad de discapacitados constituye la **tercera economía** mundial en términos de poder adquisitivo, ya que 1 de cada 5 personas de la población mundial tiene algún tipo de discapacidad. Esto se traduce en un argumento empresarial concreto para invertir en funciones de accesibilidad web.

Además, una web accesible es a menudo un requisito legal . Según la Convención de las Naciones Unidas sobre los Derechos de las Personas con Discapacidad (CDPD), el acceso a las tecnologías de la información y la comunicación, incluida la web, es un **derecho humano fundamental**. Muchos países tienen ahora **políticas** legales de accesibilidad web basadas en ella, y en los últimos años **han aumentado las demandas** contra empresas por violación de esas políticas. El primer caso se ganó en 2017, cuando una persona con

discapacidad visual demandó a Winn-Dixie, una cadena de supermercados de EE.UU., porque el sitio web de la empresa no admitía lectores de pantalla (aunque esa decisión ha sido revocada desde entonces). Así que, por todas estas razones y más, si aún no lo estás haciendo, es hora de que los equipos de desarrollo de software y las empresas empiecen a prestar más atención a las funciones de accesibilidad web.

Este capítulo te proporcionará una amplia introducción a las pruebas y herramientas de accesibilidad web. Tendrás una visión general de los personajes de la accesibilidad, el ecosistema de herramientas y tecnologías, el funcionamiento interno de los lectores de pantalla y las directrices de accesibilidad web que exigen muchos gobiernos de todo el mundo. También aprenderás sobre los marcos de desarrollo web que admiten la accesibilidad, y una estrategia de pruebas de accesibilidad por turnos. Por último, hay ejercicios en los que se presentan herramientas automatizadas de auditoría de la accesibilidad que puedes incorporar a tu estrategia de pruebas continuas para que tu equipo pueda ofrecer continuamente un sitio web accesible.

NOTA

Las herramientas de comprobación de la accesibilidad móvil se tratan en [el Capítulo 11](#).

Bloques de construcción

Empecemos por conocer a las personas usuarias de la accesibilidad y sus necesidades específicas. Tras el debate sobre las personas usuarias, se ofrecerá una visión general del ecosistema de la accesibilidad y de las directrices de accesibilidad web.

Accesibilidad Personas usuarias

Para recordar, un personaje de usuario es un personaje que representa a un subconjunto del público más amplio con atributos similares. En los proyectos de software creamos usuarios-persona para comprender sus necesidades específicas y asimilarlas a lo largo de las fases de desarrollo del software, empezando por el diseño. La [Figura 9-1](#) muestra un conjunto de personajes de usuario específicos de la accesibilidad.

Accessibility user personas



Figura 9-1. Personas usuarias de accesibilidad

Estas personas podrían definirse del siguiente modo:

- Matt, un profesional de 30 años, se ha roto el brazo recientemente. Como le cuesta manejar el ratón, necesita acceder al sitio web sólo con el teclado.
- Helen es una profesora jubilada de 80 años cuya sensibilidad al color ha disminuido. Para acceder a la web, necesita contraste de color en la interfaz de usuario, es decir, que se distingan los elementos de fondo y primer plano, como imágenes, enlaces, botones, etc. Este requisito también se aplica a los usuarios daltónicos.
- Abbie es una adolescente con discapacidad cognitiva. Como tarda en aprender cosas nuevas, necesita un diseño web limpio, con encabezamientos adecuados, barras de navegación y estructuras de navegación coherentes para acceder a la web. Fred (no aparece en la foto), que quiere encontrar una gasolinera cercana mientras conduce, también necesita una disposición clara de la información para tomar una decisión rápidamente.
- Connie es ciega y directora de una tienda independiente. Necesita ayuda de texto a voz y reconocimiento de voz para acceder a la web.
- Laxmi tiene un bebé al que lleva encima la mayor parte del día. También necesita la conversión de voz a texto para enviar mensajes.
- Maya es una profesional del software que tiene la destreza reducida, y necesita textos, botones y controles grandes para acceder a la web. Los usuarios con dislexia y baja visión también tienen este requisito.
- Philip es sordo y aficionado a la cocina. Necesita subtítulos para entender los videos de recetas que le gusta ver.

- Xiao es propietario de una tienda de habla china. Lleva aprendiendo inglés sólo un par de meses. Xiao necesita instrucciones sencillas y contenidos comprensibles que no incluyan jerga ni palabras y frases complejas para acceder a la web. Los usuarios con dificultades cognitivas y de aprendizaje también se beneficiarán de esta función.

En conjunto, nuestros usuarios presentan dificultades visuales (totales o parciales), auditivas, cognitivas y musculares, así como restricciones temporales de accesibilidad. El objetivo es que todos ellos puedan percibir, comprender, navegar e interactuar por igual, como cualquier otro usuario.

Ecosistema de accesibilidad

Para crear funciones web accesibles, tenemos que comprender todo el ecosistema de la accesibilidad. Esto engloba las distintas herramientas y tecnologías (más allá de la tecnología web) que interactúan y se combinan para ofrecer contenidos a usuarios con discapacidades temporales y permanentes. Por ejemplo, una persona usuaria como Connie, que es ciega, utiliza texto a voz y comandos de voz para interactuar con la web. Para ello, la tecnología de lectura de texto y de comandos de voz cooperan. Algunos de nuestros otros personajes necesitan dispositivos de asistencia integrados en el ordenador. Así que, para que podamos pensar en distintos casos de uso de la accesibilidad que construir, tenemos que entender estos distintos componentes e integraciones, al menos a un alto nivel. Entre los elementos del ecosistema de accesibilidad que debemos tener en cuenta se incluyen:

Herramientas y prácticas de desarrollo web

Obviamente, las herramientas de desarrollo web como HTML, CSS, etc. deben tener las facilidades necesarias para que la web sea accesible. Por ejemplo, para transmitir información sobre los elementos de una página al lector de pantalla, debe haber

disposiciones en los marcos de desarrollo web que los mencionen explícitamente.

Agentes de usuario

Son las herramientas que renderizan el contenido web, como los navegadores y los reproductores multimedia. Estos agentes de usuario deben entender que el contenido web está habilitado con funciones relacionadas con la accesibilidad e integrarse con otras herramientas, como los lectores de pantalla, para ofrecer el contenido.

Tecnologías de apoyo

Las tecnologías de apoyo son los dispositivos y tecnologías adicionales de que hablan con el navegador y transmiten información al usuario y desde él; por ejemplo, lectores de pantalla, teclados alternativos, interruptores, etc.

Como puedes ver, el ecosistema de la accesibilidad comprende un vasto conjunto de herramientas y tecnologías. Disponer de disposiciones de accesibilidad en todos estos componentes permite a todas nuestras personas interactuar con la web. Algunos pueden proporcionar funciones más avanzadas que otros, lo que puede dar lugar a que sea necesario dar más rodeos en un área o a que nuestras personas usuarias carezcan de algunas funciones.

Para garantizar que todos estos componentes tengan características de accesibilidad normalizadas, la WAI del W3C ha establecido normas internacionales para cada uno de ellos, que se enumeran aquí:

- *Las Directrices de Accesibilidad para Herramientas de Autor* (ATAG) establecen las normas para herramientas de autoría de contenidos como editores de HTML.

- *Las Pautas de Accesibilidad al Contenido en la Web* (WCAG) definen las normas de contenido web y es a las que debemos prestar atención durante el desarrollo.
- *Las Pautas de Accesibilidad para el Agente de Usuario* (UAAG) abordan las normas para navegadores web y reproductores multimedia, incluidos algunos aspectos de las tecnologías de asistencia.

Todas estas normas se detallan en el [sitio de la WAI](#). Como equipos de desarrollo web, profundizaremos en las WCAG en la siguiente sección, concretamente en la WCAG 2.0, que enumera las especificaciones de diversos aspectos del contenido web (texto, imágenes, colores, medios, etc.) para hacerlo accesible. Muchos países han elaborado políticas para que los sectores gubernamental, público y privado impongan las normas WCAG 2.0, como ya se ha mencionado.

Ejemplo: Lectores de pantalla

Para entender por qué las WCAG 2.0 prescriben directrices específicas, debemos comprender cómo funcionan las tecnologías de asistencia. Consideremos el ejemplo de los lectores de pantalla, utilizados por nuestros usuarios con problemas visuales : se trata de una tecnología de asistencia común cuya compatibilidad debemos asegurarnos de comprobar.

Como su nombre indica, los lectores de pantalla leen en voz alta el contenido de la página para el usuario, que interactúa con el sitio web mediante un teclado. Así, mientras oye el contenido, el usuario pulsa atajos de teclado como Tabulador, Tab+Mayús, Intro, etc., para interactuar con el sitio.

El lector de pantalla recita el contenido de la página en el orden del *árbol de accesibilidad* de la página . Se trata de una estructura similar al DOM con los elementos de la página, junto con atributos

como roles, ID, etc., definidos explícitamente en una secuencia que representa un flujo significativo. Por ejemplo, considera un sitio de reservas con campos de entrada de texto Para y Desde y un botón Buscar en la página de inicio. El árbol de accesibilidad se estructurará para representar el flujo de usuario de las "entradas de búsqueda", es decir, introducir primero la ubicación De, luego la ubicación A y, por último, pulsar el botón Buscar. Podemos codificar determinados elementos de la página web para que queden ocultos en el árbol de accesibilidad si es necesario.

Para relacionarte mejor con las funciones de accesibilidad, es una buena idea que experimentes tú mismo el uso de un lector de pantalla. Google Chrome proporciona un lector de pantalla basado en el navegador como [extensión](#). ¡Pruébalo! También hay sitios web de demostración, como [el ejemplo de reserva de la Figura 9-2](#), que dan una idea de cómo pueden experimentar la web las personas con discapacidad visual; el contenido está intencionadamente borroso, y puedes hacer una reserva en utilizando un lector de pantalla y tu teclado.

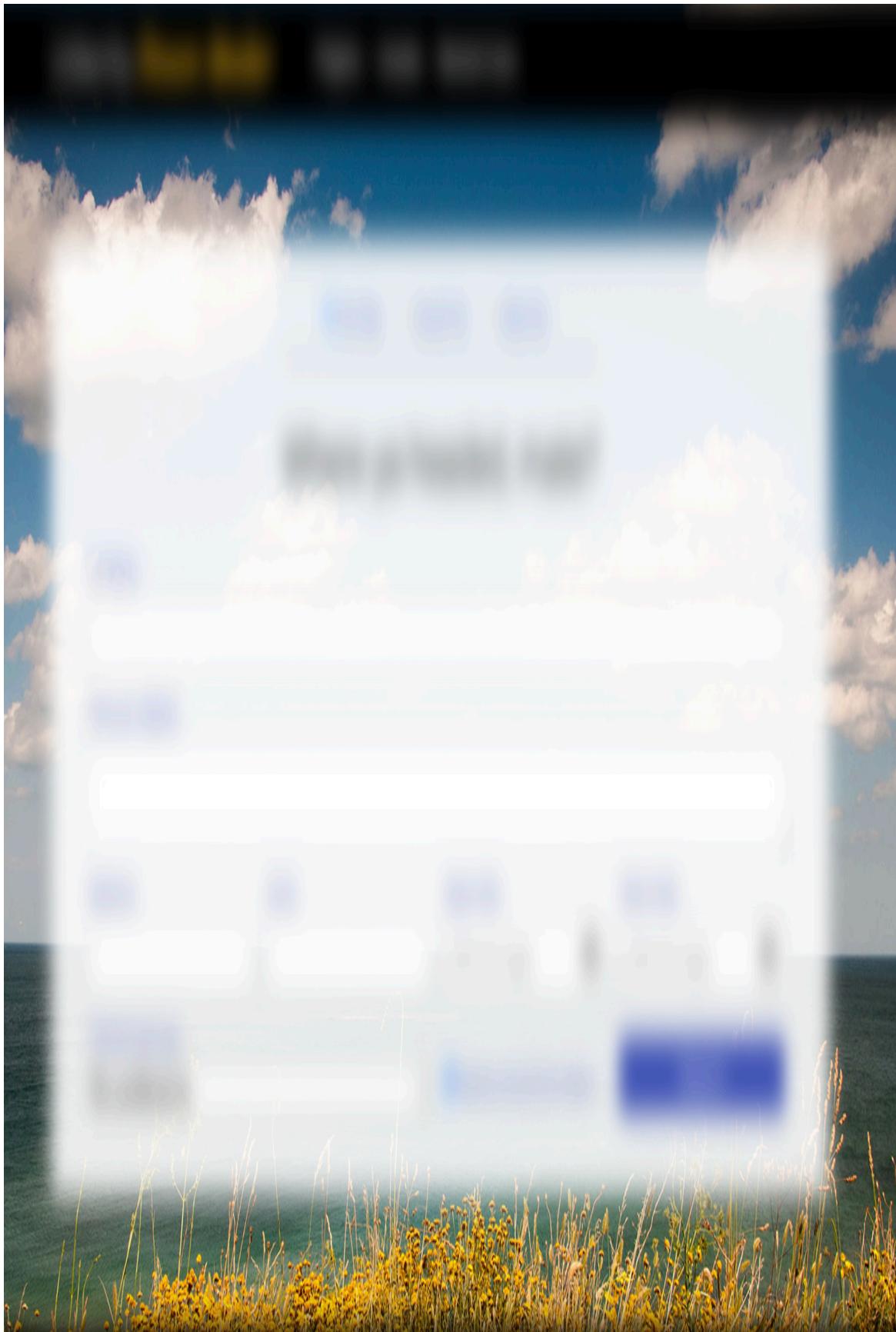


Figura 9-2. Captura de pantalla de un sitio web de demostración, intencionadamente borrosa, para simular la experiencia del lector de pantalla

WCAG 2.0: Principios rectores y niveles

Si has tenido la oportunidad de probar el lector de pantalla, ¡genial! Si no, espero que te hayas hecho una idea de cómo funcionan las cosas en la sección anterior. Exploraremos ahora en detalle las WCAG 2.0.

WCAG 2.0 establece cuatro principios rectores que hay que recordar al diseñar contenidos web: el contenido debe ser *perceptible, manejable, comprensible y sólido*. La norma también define tres niveles de conformidad basados en el grado en que el contenido cumple los criterios de éxito definidos para cada una de las directrices definidas:

Nivel A

Éste es el nivel mínimo de conformidad con , que proporciona un soporte esencial sin el cual el sitio es inaccesible. Por ejemplo, el contenido de audio o vídeo debe tener subtítulos, toda la funcionalidad debe ser accesible mediante el teclado y no debe utilizarse el color como único medio de transmitir información. La conformidad con el nivel A permitirá a todas nuestras personas usuarias navegar por la web.

Nivel AA

Abarca todos los requisitos del Nivel A más otros más estrictos, como las relaciones de contraste de color limitadas en todo el emplazamiento. Algunas políticas legales recomiendan que los sitios alcancen este nivel de conformidad.

Nivel AAA

Este nivel subsume todos los requisitos de los dos niveles anteriores y exige requisitos adicionales mejorados para que la

web sea realmente accesible para todos los usuarios. Un ejemplo de requisito de este nivel sería disponer de interpretaciones en lengua de signos para los contenidos de vídeo. Cuando intentas alcanzar este nivel de conformidad, demuestras a los usuarios que realmente te preocupas por ellos.

Las organizaciones deben determinar el nivel al que deben ajustarse en función de los requisitos legales, pero pueden optar por implantar un nivel de conformidad superior para dar servicio a más usuarios.

Normas de conformidad de nivel A

Echemos un vistazo a los requisitos WCAG 2.0 de nivel A, el nivel mínimo que deben cumplir los sitios web. Podemos trasladar estos requisitos directamente a nuestros casos de prueba de accesibilidad.

NOTA

Los detalles que se ofrecen aquí sólo pretenden dar una visión general, para ayudar al lector a comprender los requisitos de las pruebas de accesibilidad. La [documentación oficial](#) está disponible en el sitio WAI del W3C.

Perceptible

El primer principio es hacer que el contenido de la web esté disponible cómodamente para todas nuestras personas usuarias. Sólo cuando puedan percibir el contenido podrán seguir operando con él. Por tanto, debemos pensar en todos los posibles escenarios que podrían obstaculizar este requisito esencial desde la fase de diseño web, y evitarlos.

Las WCAG 2.0 elaboran este principio con requisitos detallados para que empecemos por él, como se indica a continuación:

- Todos los contenidos no textuales, como las imágenes, deben tener un texto alternativo que los describa para que los usuarios con problemas de visión puedan entender el contenido utilizando lectores de pantalla.
- El contenido de audio o vídeo debe tener transcripciones de texto y subtítulos (sincronizados con los medios) en como alternativas, con disposiciones para pausar, parar y controlar el volumen.
- Cualquier audio que comience a reproducirse automáticamente al cargar la página debe tener mecanismos de control de audio como pausa, repetición y controles de volumen.
- La información y la estructura de la página web deben diseñarse para que tengan una jerarquía, como un título de página por encima de todos los encabezamientos de , etiquetas apropiadas de título y encabezamiento de página, etc. Esto ayuda a los usuarios con lectores de pantalla a tener un flujo significativo.
- Las instrucciones para navegar por el sitio web no deben basarse únicamente en las características sensoriales de los componentes, como la forma, el color, el tamaño, la ubicación visual, la orientación o el sonido. Por ejemplo, evita una instrucción que diga "espera a que el botón se ponga verde" o "espera a oír un pitido".
- Los colores no deben ser la única forma de indicar una acción, pedir una respuesta o distinguir elementos en la pantalla. Hazlo intuitivo mediante texto para los usuarios daltónicos.
- La página debe tener contraste de color entre los elementos de fondo y de primer plano para ayudar a los usuarios con menor sensibilidad al color. Hay una proporción fija prescrita para ello.

Operable

Una vez que el contenido de la web se ha hecho perceptible, en debemos considerar formas de permitir que los usuarios manejen la web cómodamente, como pulsar un botón utilizando atajos de teclado. Las WCAG 2.0 incluyen requisitos específicos en este sentido, como se indica a continuación:

- Proporcionar un soporte de navegación mediante teclado que permita a los usuarios manejar todo el sitio web. Al utilizar la navegación con teclado, el enfoque de los elementos debe ser claro y tener un contraste de color adecuado.
- Añade disposiciones para avanzar, retroceder y salir de un área mediante atajos de teclado; por ejemplo, teclas para salir de una ventana modal.
- Proporciona tiempo suficiente para que los usuarios lean el contenido por completo.
- Evita los contenidos que parpadeen en la pantalla y tengan muchas animaciones, ya que podrían provocar reacciones físicas como convulsiones.
- Ofrecer la posibilidad de saltarse contenidos repetitivos.
- Oculta el contenido fuera de pantalla para los lectores de pantalla. Por ejemplo, si un enlace sólo aparece en una selección concreta, ocúltalo en el flujo del lector de pantalla.
- Proporciona un texto elaborado y significativo para los enlaces.

Comprensible

Un sitio web puede requerir muchos elementos y flujos de usuario para completar una acción. Por ejemplo, para reservar un billete de avión, puede haber que seguir varios pasos e instrucciones. Estos contenidos y flujos de usuario deben elaborarse cuidadosamente para que sean sencillos y directos para todas las personas usuarias.

Una vez más, las WCAG 2.0 establecen requisitos sólidos en este sentido:

- Evita la jerga y los términos técnicos; en su lugar, presenta contenidos sencillos y significativos. Por ejemplo, evita mensajes de error técnicos como "034506451988 no es válido" y proporciona un texto comprensible como "Formato de fecha incorrecto".
- Proporciona ampliaciones y abreviaturas cuando sea necesario.
- Evita los cambios bruscos de contexto (por ejemplo, abrir varias ventanas), ya que afectarán a la navegación con el teclado.
- Evita los cambios de contexto cuando el usuario tenga ajustes diferentes, como una fuente más grande.
- Proporciona un texto de etiqueta claro y procesable a los elementos para ayudar a los usuarios a realizar la acción correcta. Por ejemplo, un campo de entrada de dirección de correo electrónico debe tener la etiqueta Correo electrónico y un valor de ejemplo como *example@xyz.com*.

Robusto

Por último, debemos hacer que el contenido web sea robusto, compatible con distintos tipos de agentes de usuario y tecnologías de asistencia. Los lectores de pantalla no son las únicas tecnologías de asistencia; muchas otras requerirán una integración adecuada. Las WCAG 2.0 establecen los siguientes requisitos para este principio:

- El contenido del lenguaje de marcado debe seguir unas normas, como tener etiquetas de apertura y cierre, no tener duplicados, ID únicos, etc., para que sea fácilmente analizable por múltiples tecnologías de asistencia.

- El nombre, la función y el estado de cada elemento, incluidos los generados por scripts, deben estar disponibles para las tecnologías de asistencia (por ejemplo, `role="checkbox"` y `aria-checked="true|false"`). Proporciona el estado actualizado de elementos como las casillas de verificación al lector de pantalla tras la selección.

APLICACIONES RICAS DE INTERNET ACCESIBLES DE LA WAI (WAI-ARIA)

Antes hemos visto cómo un lector de pantalla lee los elementos de la página y describe las acciones a realizar sobre ellos basándose en el árbol de accesibilidad de la página web. A veces, cuando se desarrollan elementos personalizados para enriquecer la interacción del usuario, las tecnologías de asistencia no podrán identificar los elementos. Tales elementos deben llevar **atributos adicionales** que indiquen su tipo, estados y comportamientos para que las tecnologías de asistencia puedan entenderlos. Por ejemplo, la definición de un elemento HTML estándar, digamos `<input type="checkbox">`, se traducirá automáticamente como una casilla de verificación, y se indicará correctamente al usuario final que realice una acción de clic (pulsar) sobre él. Sin embargo, cuando se hace que un elemento de lista (``) parezca una casilla de verificación utilizando CSS, hay que aumentarlo con nuevos atributos para que las tecnologías de asistencia lo entiendan correctamente.

WAI-ARIA proporciona especificaciones para estos atributos (por ejemplo, `roles`, `aria-checked`, etc.) que deben respetarse durante el desarrollo web. Estos atributos ARIA se añaden al árbol de accesibilidad, haciéndolo amigable para todas las tecnologías de asistencia.

Éstos son los requisitos esenciales del Nivel A. También existe una versión actualizada de la norma, la WCAG 2.1, que incluye algunos

requisitos más para atender mejor a un determinado conjunto de personas usuarias; puedes explorarlos en la [documentación oficial](#) si tu organización opta por cumplir esta versión.

Marcos de desarrollo accesibles

Para construir las características mencionadas anteriormente, muchos marcos de desarrollo proporcionan un elaborado soporte de accesibilidad . Por ejemplo, React es totalmente compatible con la creación de sitios web accesibles, a menudo utilizando técnicas HTML estándar. Del mismo modo, el equipo de Angular mantiene una biblioteca Angular Material que proporciona un conjunto de componentes de interfaz de usuario reutilizables que pretenden ser totalmente accesibles. Vue.js también tiene soporte para crear componentes accesibles. También existen herramientas automatizadas de auditoría de accesibilidad, como verás en la siguiente sección, que alertan si faltan etiquetas estándar relacionadas con la accesibilidad en el HTML. Así que, no te preocunes, ¡tienes apoyo! Tu equipo también puede conseguirlo sin mucho esfuerzo adicional.

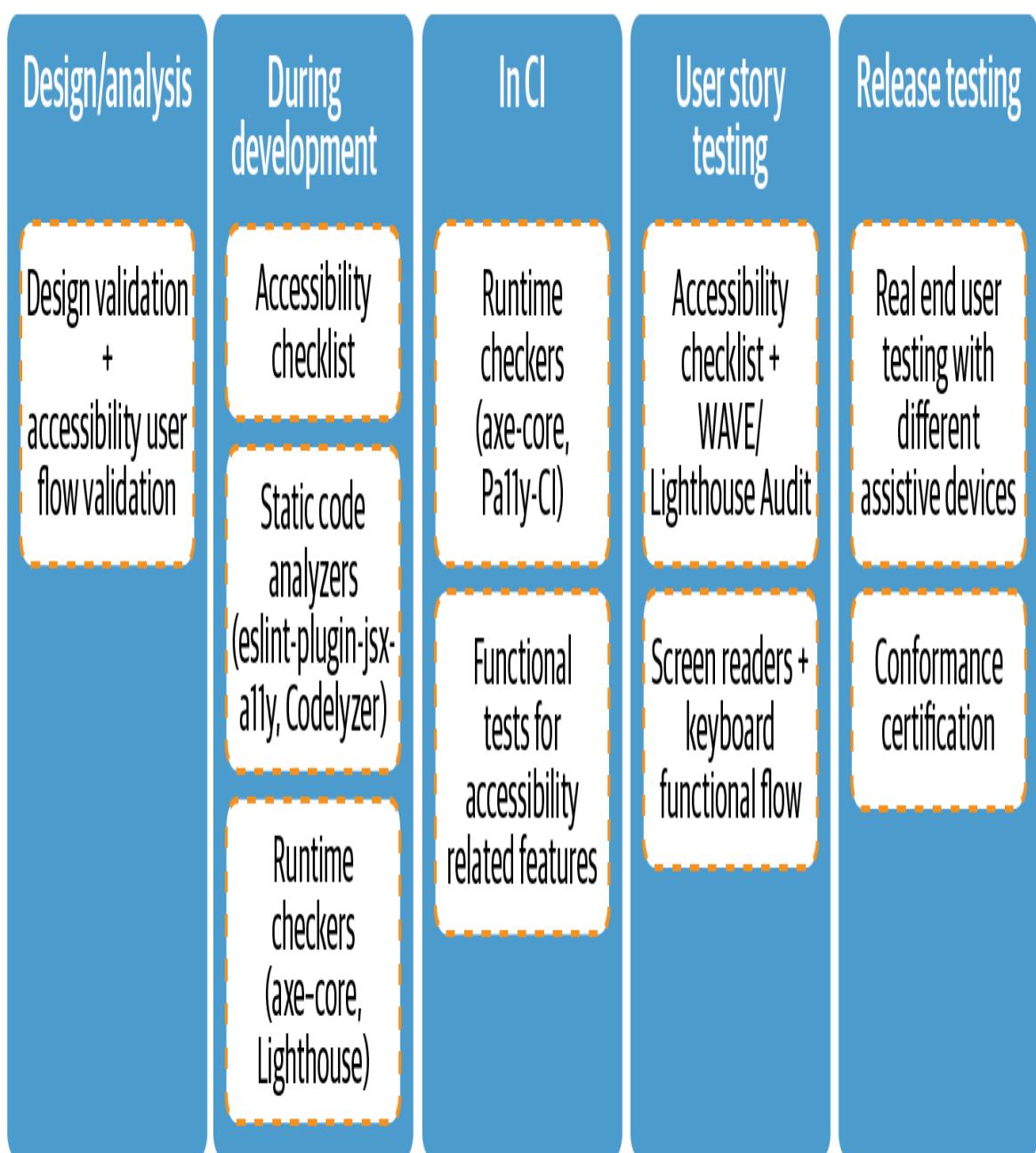
Estrategia de pruebas de accesibilidad

De las secciones anteriores debería resultar evidente que la mayoría de los requisitos de accesibilidad tienen que ser pensados desde el principio del proyecto y apoyados continuamente a lo largo del proceso de desarrollo, en lugar de adaptarlos después de la fase de pruebas. Por ejemplo, para ajustarte al Nivel A, debes incorporar una navegación sencilla y coherente en todo el sitio, subtítulos para los vídeos, mensajes de error significativos, imágenes con contraste de color, etc., durante el diseño del producto y no durante el desarrollo o las pruebas. Así que, como primer paso para apoyar la accesibilidad, los equipos deben definir los personajes de usuario accesibles de la aplicación, de forma similar a lo que comentamos al

principio del capítulo, y adaptar las historias de usuario a cada uno de esos personajes de usuario. Después, cuando tu equipo discuta las características del producto, valida colectivamente si los flujos de accesibilidad están incluidos en el alcance. ¡Ese será el paso principal para desplazar las pruebas de accesibilidad hacia la izquierda!

La Figura 9-3 muestra la implementación de las pruebas de accesibilidad a lo largo del ciclo de vida de desarrollo del software. Nos sumergiremos en algunos de estos elementos en el resto de esta sección y en los ejercicios que siguen.

Shift-left accessibility testing



Path to production

Figura 9-3. Estrategia de comprobación de la accesibilidad con shift-izquierda

Lista de comprobación de accesibilidad en las historias de usuario

Las directrices WCAG 2.0 incluyen varios requisitos universales que abarcan todas las páginas del sitio web, como añadir texto alternativo, compatibilidad con la navegación mediante teclado, títulos de página, etc. Por tanto, adjuntar una lista de comprobación de accesibilidad a todas las historias de usuario ayudará a los desarrolladores y probadores a revisar meticulosamente estos requisitos. La siguiente es una lista de comprobación genérica que puedes utilizar en tu equipo, después de añadir los elementos específicos de la aplicación.

LISTA DE CONTROL DE ACCESIBILIDAD

- Busca el título de la página en el navegador. Cuando pasas el ratón por encima de la pestaña del navegador, puedes ver el título de la página como un pequeño widget en Chrome. Este texto debe definir claramente el contexto de la página dentro del sitio web.
- Comprueba la estructura básica de la página web para asegurarte de que tiene los atributos y la jerarquía de elementos adecuados. Desactiva el CSS y comprueba que los elementos aparecen en un orden adecuado para el lector de pantalla. Chrome tiene una vista de árbol de accesibilidad en las DevTools para mostrar el orden de los elementos.
- Comprueba la navegación mediante teclado, incluyendo el resaltado correcto de los elementos y las operaciones de teclado para retroceder, avanzar y salir.
- Comprueba que los mensajes de error, las etiquetas, los enlaces y, en general, cualquier texto de la página comunican la intención correcta.
- Comprueba la legibilidad de la página al cambiar el tamaño del texto, utilizando las preferencias del sistema o las opciones de zoom del navegador.
- Comprueba la legibilidad en escala de grises. Por ejemplo, los usuarios de Mac pueden activar este ajuste a través de Preferencias del Sistema → Accesibilidad → Pantalla → Usar escala de grises.
- Comprueba los subtítulos de los contenidos de vídeo y audio, asegurándote de que tienen sentido y están sincronizados.

- Comprueba si las descripciones de texto alternativo de las imágenes son significativas. Puedes comprobarlo desactivando la opción de descarga de imágenes en la configuración de tu navegador (por ejemplo, en Chrome, selecciona Configuración → Configuración del sitio → Imágenes → Bloquear); el navegador mostrará entonces el texto alternativo en lugar de las imágenes.
- Comprueba que el flujo del lector de pantalla es significativo y que el usuario final puede completar el flujo de usuario.

Herramientas automatizadas de auditoría de la accesibilidad

La lista de comprobación de accesibilidad incluye algunos aspectos que sólo es posible verificar con intervención humana, como ver el sitio en escala de grises, ampliar y reducir la imagen, comprobar el significado del texto alternativo, etc. Puedes complementarlo con herramientas de auditoría automatizadas que escanearán la estructura HTML básica y alertarán si faltan las etiquetas de accesibilidad de algún elemento. Estas herramientas pueden ahorrarte mucho tiempo y esfuerzo al darte información instantánea sobre las etiquetas que faltan durante la propia fase de desarrollo.

Vienen en forma de analizadores de código estático y comprobadores de accesibilidad en tiempo de ejecución. [eslint-plugin-jsx-a11y](#) es una herramienta de linting para React; es un complemento de ESLint que aplica varias normas de accesibilidad directamente en tu JSX. Del mismo modo, [Codelyzer](#) tiene reglas de linting para las normas de accesibilidad en el código fuente de TypeScript, HTML, CSS y Angular. Estas herramientas te darán información mientras desarrollas, mientras que los verificadores en tiempo de ejecución (como axe-core, Pa11y CI y Lighthouse CI, que se tratan más adelante en el capítulo) te darán información sobre la

página web real después del desarrollo. Pueden ejecutarse en la máquina de desarrollo local para obtener información más rápidamente sobre las páginas desarrolladas como parte de cada historia de usuario y también como parte de CI para pruebas continuas.

Además de utilizar herramientas como éstas, puedes añadir flujos funcionales que atiendan a las necesidades de accesibilidad, como tener una sección de transcripción separada debajo de cualquier vídeo/audio o un conjunto de mensajes de error e instrucciones significativos, como pruebas funcionales automatizadas a nivel micro y macro.

Pruebas manuales

Las pruebas manuales son fundamentales para validar la accesibilidad del sitio web. Como se ha mencionado anteriormente, las herramientas automatizadas sólo comprueban la estructura HTML, y la lista de comprobación sólo incluye los elementos obligatorios que son comunes a todas las páginas. Qedarán muchos elementos por tratar aparte de éstos como parte de las pruebas manuales; por ejemplo, verificar el flujo funcional con un lector de pantalla y un teclado. Así que, como parte de las pruebas manuales, puedes centrarte en distintos ámbitos en distintas fases, como las siguientes:

Pruebas de historias de usuarios

Como parte de la prueba de la historia de usuario, asegúrate de que la lista de comprobación funciona correctamente en todas las páginas a las que se extiende la historia de usuario. Puedes combinar esto con la herramienta de evaluación de la accesibilidad web *WAVE*, un servicio gratuito en línea de WebAIM. Ayuda a encontrar problemas de accesibilidad en la página web, según las directrices WCAG 2.0. Aunque las comprobaciones no verifican todos los requisitos de la norma

WCAG 2.0, resalta los problemas visualmente en la página web en un navegador, lo que puede ayudarte a darte cuenta de algunos problemas de accesibilidad de los que quizás no eras consciente. Alternativamente, puedes utilizar Lighthouse, que (como vimos en el último capítulo) forma parte de Chrome DevTools, para obtener auditorías de accesibilidad en la página web. Ambas herramientas se tratan más adelante en este capítulo.

Prueba de características

Con las pruebas a nivel de historia de usuario, tú cubrirás la mayor parte del esfuerzo de las pruebas de accesibilidad. Pero cuando una funcionalidad esté completa, tendrás que hacer otra ronda de pruebas manuales para asegurarte de que los usuarios finales pueden completar el flujo de usuario sólo con acceso al teclado y de que los lectores de pantalla pueden navegar por la funcionalidad como se espera. Este nivel de pruebas de las funciones ayudará a identificar cualquier falta de coherencia en la navegación de extremo a extremo de la aplicación.

Para probar la navegación con el teclado, utiliza las teclas Tabulador y Tab+Mayúsculas para avanzar y retroceder por el sitio web, la tecla Intro para seleccionar y las teclas de flecha arriba y abajo para las selecciones desplegables. Mientras haces esto, asegúrate de que el foco está en el elemento correcto y de que ese elemento está claramente resaltado. Para probar el flujo del lector de pantalla, puedes utilizar la extensión de Chrome mencionada anteriormente.

Con estas comprobaciones, terminarás las pruebas de accesibilidad de extremo a extremo de las funciones.

Prueba de liberación

Por último, cuando se hayan completado todas las funciones de la versión , se recomienda realizar pruebas con usuarios finales

reales, incluidas personas con discapacidad. Dado que los distintos usuarios pueden tener diversos dispositivos de asistencia, esto te proporcionará información en tiempo real antes de que el producto se someta a una evaluación final para la certificación de conformidad. [UserTesting.com](#) es un servicio de pruebas a distancia en el que puedes solicitar personas con discapacidad como probadores para tu sitio web.

Certificación de conformidad

Una vez que el sitio web está listo, los expertos en normas WCAG realizan la evaluación de conformidad antes de que se ponga en marcha. No se trata de una unidad centralizada; las organizaciones pueden tener expertos internos o contratar consultores para que hagan la evaluación final una vez que el producto esté listo para la certificación.

Puesto que todos y cada uno de los elementos de la página requieren cambios para hacerla accesible, desplazarse a la izquierda es la única forma de rescatar a tu equipo de la ardua tarea de arreglar todos los problemas de accesibilidad al final del ciclo de desarrollo de .

Ejercicios

En la discusión anterior mencioné algunas herramientas automatizadas de auditoría de accesibilidad. Puedes probar algunas de estas herramientas como parte de los siguientes ejercicios.

NOTA

Las herramientas de auditoría de accesibilidad ayudan a confirmar que la estructura HTML está intacta y alertan si no lo está. Por ejemplo, comprueban que todas las etiquetas HTML estén cerradas, que todas las imágenes tengan un atributo de texto alternativo, que todos los elementos de formulario tengan etiquetas, que todos los ID de elemento sean únicos, etc. Son útiles para realizar un análisis preliminar y proporcionar información rápida; sin embargo, no eliminan la necesidad de realizar pruebas manuales.

ONDA

WAVE es una herramienta de evaluación en línea de la accesibilidad que puedes utilizar para comprobar si una página web cumple las normas de accesibilidad. Tiene disposiciones para determinar la estructura de la página sin CSS, y señala problemas relacionados con cosas como el contraste de color de los elementos, los atributos lang, etc. WAVE es sencillo y gratuito.

Flujo de trabajo

Para realizar una auditoría con la herramienta WAVE:

1. Abre el [sitio web de WAVE](#).
2. Introduce la URL de tu aplicación en la casilla "Dirección de página web". Como alternativa, puedes utilizar [el sitio web de demostración inaccesible de la WAI](#), que se ha hecho inaccesible intencionadamente con fines didácticos.
3. Haz clic en la flecha para ejecutar la auditoría.

[La Figura 9-4](#) muestra el resumen de los resultados de la auditoría del sitio de demostración de la WAI. La herramienta ha identificado 3 elementos estructurales y 6 características, y ha marcado 37 errores y alertas y 2 errores de contraste. Puedes ver los iconos de error,

acuerdo y alerta junto a los respectivos elementos de la página web a la derecha.

Summary



Summary



Details



Reference



Structure



Contrast

37

Errors

2

Contrast Errors

37

Alerts

6

Features

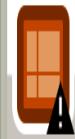
3

Structural Elements

0

ARIA

View details >



*Red dot with a white let
crescent as well as the sun. This
that says 'CITYLIGHTS' which is t
Finally, the slogan of the portal,
turquoise green handwriting sty



Traffic: Construction w



Figura 9-4. Informe de auditoría de WAVE sobre el sitio web de demostración inaccesible de WAI

Si haces clic en la pestaña Detalles, junto a Resumen, se mostrarán los detalles del error, como se ve en la **Figura 9-5**.

Details

Summary Details Reference Structure Contrast

✓ X 37 Errors

✓ 19 X Missing alternative text

✓ 7 X Linked image missing alternative text

✓ 10 X Spacer image missing alternative text

✓ 1 X Language missing or invalid



Figura 9-5. Pantalla de detalles del error WAVE

La página de demostración tiene 19 imágenes sin texto alternativo, 7 imágenes que son enlaces sin texto alternativo, 10 imágenes espaciadoras sin texto alternativo y 1 atributo de idioma que falta o no es válido. Los diferentes estilos de iconos de la pestaña Detalles pueden corresponderse con los iconos de la página web para facilitar la identificación y la depuración.

A continuación, para ver la estructura de la página, puedes desactivar los estilos CSS mediante el control presente encima de la sección de resumen. La pestaña Estructura mostrará entonces el análisis de la estructura de la página. Como puedes ver en [la Figura 9-6](#), cuando los estilos están desactivados, el texto se superpone en la página y resulta torpe. A la página web también le faltan las jerarquías adecuadas y las secciones Cabecera, Navegación y Principal, lo que la califica de inaccesible.



Figura 9-6. Análisis de la estructura de la página WAVE con los estilos desactivados

Ahora prueba el sitio de demostración accesible de la WAI, que es una versión accesible del mismo sitio web. Puedes ver que la estructura de la página está correctamente diseñada con una jerarquía, como se muestra en la Figura 9-7.

Styles: OFF ON

Structure

Summary Details Reference **Structure** Contrast

h1 Accessible Home Page - Before and After Demonstration

h2 Navigation menu:

h1 Welcome to CityLights

h2 Heat wave linked to temperatures

h2 Man Gets Nine Months in Violin Case

h2 Lack of brains hinders research

h2 Elsewhere on the Web

- [Skip to content \(within demo page\)](#)
- [Skip to navigation \(within demo pag](#)



Explore Site by Topic: Quick Menu

Traffic: Construction work on Main Road

Today: Thursday 14 January 2021, Sunny

h2 Navigation menu:



- Home
- [News](#)
- [Tickets](#)
- [Survey](#)

Figura 9-7. Análisis de la estructura de la página WAVE del sitio web de demostración accesible de la WAI

En un sitio habilitado para la accesibilidad como éste, puedes verificar fácilmente la secuencia de elementos y contenidos y comprobar en que la navegación es la esperada dada la jerarquía mostrada.

Faro

Si tu aplicación no es accesible públicamente , es posible que no puedas utilizar WAVE. Una alternativa es utilizar la herramienta Lighthouse de Google, que te permite auditar la accesibilidad de un sitio web utilizando tu navegador local Chrome.

Flujo de trabajo

Prueba estos sencillos pasos para ver cómo funciona Lighthouse:

1. Abre **el sitio web de demostración inaccessible de la WAI** en Chrome.
2. Abre Chrome DevTools utilizando el atajo de teclado Cmd-Opción-I en macOS o Mayúsculas-Ctrl-J en Windows/Linux.
3. En la pestaña Lighthouse, selecciona la categoría Accesibilidad, como se muestra en **la Figura 9-8**, y haz clic en "Generar informe".

The screenshot shows the W3C Web Accessibility Initiative website. At the top, there's a navigation bar with a plus sign, '(new report)', and a search icon. Below the header, there's a large blue button with a red and yellow accessibility icon. To the right of the button is a sidebar titled 'Categories' with several checkboxes: 'Performance', 'Progressive Web App', 'Best practices', 'Accessibility' (which is checked), and 'SEO'. Below the sidebar, there's a message about identifying and fixing common problems related to site performance, accessibility, and user experience, with a link to 'Learn more'. On the left side of the main content area, there's a navigation menu with tabs: 'News', 'Tickets', 'Survey', 'Template', 'Home Page', 'Report', 'Show', 'Home Page', 'Report', and 'Annotations'. Below this menu, there's a banner with the text 'S your access to the city' and a sun-like graphic. At the bottom, it says 'on Main Road' and 'Today: Thursday 1'.

Welcome to Citylights

Citylights is the new portal for visitors and residents. Find out what

Figura 9-8. Uso de Lighthouse en Chrome DevTools para generar un informe de accesibilidad

El informe de auditoría de accesibilidad de Lighthouse estará disponible en breve en el mismo panel, como se ve en [la Figura 9-9](#).



51

Accessibility

These checks highlight opportunities to [improve the accessibility of your web app](#). Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

Contrast – These are opportunities to improve the legibility of your content.

▲ Background and foreground colors do not have a sufficient contrast ratio.

Low-contrast text is difficult or impossible for many users to read. [Learn more](#).

Failing Elements

Free Penguins

`Free Penguins`

More City Parks

`More City Parks`

Internationalization and localization – These are opportunities to improve the interpretation of your content by users in different locales.

▲ `<html>` element does not have a `[lang]` attribute

Names and labels – These are opportunities to improve the semantics of the controls in your application. This may enhance the experience for users of assistive technology, like a screen reader.

▲ Image elements do not have `[alt]` attributes

Figura 9-9. Informe de auditoría de accesibilidad de Lighthouse para el sitio web de demostración inaccesible de la WAI

Como puedes ver, informa de problemas similares a los identificados por WAVE (dos problemas de contraste, un atributo `lang` que falta, etc.). Además, para ayudar a la depuración, se presentan las líneas de código reales que contienen los errores, y hay enlaces educativos para ayudar a los desarrolladores a solucionarlos. Lighthouse proporciona una puntuación global en la parte superior del informe para dar una idea de lo buena o mala que es la página, y hay una lista de comprobación en la parte inferior del informe ("Elementos adicionales a comprobar manualmente") para indicar lo que la auditoría no ha cubierto. Esta lista también tiene enlaces educativos para guiar a en la realización de la verificación manual.

Módulo Nodo Faro

El **módulo Lighthouse Node** realiza una auditoría similar a la versión que forma parte de Chrome DevTools, pero puede ejecutarse desde la línea de comandos. Por tanto, puede utilizarse para integrarse con CI y ofrece más flexibilidad en la forma de configurar y notificar las ejecuciones.

Flujo de trabajo

Para ejecutar las auditorías de accesibilidad de Lighthouse desde la línea de comandos, sigue estos pasos:

1. Suponiendo que ya tienes instalado **Node.js**, utiliza el siguiente comando para instalar Lighthouse:

```
$ npm i -g lighthouse
```

2. Ejecuta la auditoría utilizando este comando:

```
$ lighthouse --chrome-flags="--headless" URL
```

Por defecto, el informe se genera como un archivo HTML, como se ve en la **Figura 9-10**, en el mismo directorio de trabajo.



Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

Contrast – These are opportunities to improve the legibility of your content.

▲ Background and foreground colors do not have a sufficient contrast ratio.



Internationalization and localization – These are opportunities to improve the interpretation of your content by users in different locales.

▲ `<html>` element does not have a `[lang]` attribute



Figura 9-10. Informe CLI de Lighthouse para el sitio web de demostración inaccesible de la WAI

Puedes añadir este archivo HTML como artefacto de salida en la canalización CI para facilitar la verificación. También puedes hacer que la compilación falle en si las puntuaciones están por debajo de un umbral con una tarea de compilación envolvente.

Herramientas de comprobación adicionales

Otro par de herramientas de auditoría que se utilizan con frecuencia para las pruebas de accesibilidad son Pa11y CI y axe-core. Veamos lo que ofrecen.

Módulo Nodo Pa11y CI

Pa11y CI es una herramienta de línea de comandos que viene como módulo de Node. Ejecuta auditorías de accesibilidad en una o varias URL e informa de los problemas, de forma similar a lo que vimos con las herramientas WAVE y Lighthouse. Para comprobar varias páginas web con esta herramienta, incluye las URL en la sección `urls` del archivo de configuración, como se ve en [el Ejemplo 9-1](#).

Alternativamente, puedes pasar un mapa del sitio XML en la línea de comandos con la opción `--sitemap`. Puedes añadir Pa11y CI a la etapa de construcción del proyecto en CI, o incluso como una etapa separada en la canalización.

NOTA

La accesibilidad se abrevia a veces como *a11y*, que significa "a-[11 letras intercaladas]-y".

Ejemplo 9-1. Archivo de configuración de Pa11y CI con los sitios de

demostración WAI antes y después

```
{  
    "defaults": {  
        "timeout": 1000,  
        "viewport": {  
            "width": 320,  
            "height": 480  
        }  
    },  
    "urls": [  
        "https://www.w3.org/WAI/demos/bad/after/home.html",  
        "https://www.w3.org/WAI/demos/bad/before/home.html"  
    ]  
}
```

La herramienta incluye opciones para establecer un umbral de errores y advertencias hasta el que puede pasar la compilación CI, definir tamaños de ventana gráfica para realizar auditorías y definir el tiempo de espera para que se cargue la página.

Núcleo del eje

La [documentación de GitHub](#) para axe-core afirma que puede encontrar, de media, el 57% de los problemas WCAG automáticamente. Funciona con muchos navegadores, como Microsoft Edge, Google Chrome, Firefox, Safari e IE. La herramienta tiene muchas extensiones incorporadas. Por ejemplo, para integrar Java Selenium WebDriver, puedes añadir axe-core como dependencia de Maven. Del mismo modo, para Cypress, tienes el módulo cypress-axe Node. También existen las bibliotecas vue-axe y react-axe para añadir pruebas en el frontend.

Básicamente, axe-core proporciona API para realizar auditorías de accesibilidad en páginas web, que pueden añadirse como parte de pruebas funcionales. Por ejemplo, la API `run()` ejecuta la auditoría de accesibilidad en la página actual y lanza errores de aserción en caso de fallo. Así que, al igual que otras aserciones de elementos en

la página, puedes añadir esta línea adicional para evaluar la accesibilidad de la página dentro de las pruebas funcionales.

En resumen, hemos visto herramientas independientes que pueden ejecutarse como parte de los scripts de construcción del proyecto o como una etapa separada en CI, y herramientas que pueden integrarse como parte de un conjunto de pruebas funcionales. Sea cual sea la opción que elijas, lo mejor es elaborar una estrategia para el proceso de pruebas continuas de modo que tu equipo obtenga información al principio del ciclo de desarrollo.

Todo el esfuerzo que realices dará sus frutos, ya que podrás asegurarte de que proporcionas un acceso cómodo a la aplicación a Matt, Fred, Helen, Laxmi, Connie, Xiao, Abbie, Maya, Philip y itambién a ti mismo!

Perspectivas: La accesibilidad como cultura

Aunque a lo largo de este capítulo hemos hablado de la accesibilidad en el contexto de las aplicaciones web, estos conceptos no sólo se aplican a los sitios web. La accesibilidad es una cultura y requiere un cambio de mentalidad. Cuando adoptamos esta mentalidad, empezamos a hacernos preguntas como Si envío un correo electrónico sólo con imágenes y sin resumen de texto alternativo, ¿será accesible para todos? Si utilizo un tamaño de letra pequeño en las diapositivas de mi presentación, ¿podrá leerlas todo el mundo? ¿Puedo elegir mensajes sencillos y claros en lugar de verbos eruditos para que todo el mundo pueda entenderlos? Estoy seguro de que todos conocemos las respuestas a estas preguntas.

Puntos clave

Éstos son los puntos clave de este capítulo:

- Las funciones de accesibilidad web son esenciales para algunos, pero útiles para todos.
- La comunidad de discapacitados representa la tercera economía mundial en términos de poder adquisitivo, lo que convierte a la accesibilidad en un sólido argumento comercial.
- Muchos gobiernos tienen políticas que obligan a la accesibilidad, por lo que también es un requisito legal.
- La WAI del W3C ha elaborado un conjunto de Pautas de Accesibilidad al Contenido en la Web que los equipos de desarrollo de software deben seguir. Consulta la última versión de las pautas en el momento de desarrollar tu proyecto.
- El ecosistema de la accesibilidad es amplio y abarca herramientas y tecnologías que van más allá de las de la web. Es una buena idea probar al menos una tecnología de asistencia, como un lector de pantalla, para ayudarte a pensar en buenas características de accesibilidad para tu aplicación.
- Debemos integrar la accesibilidad en el ciclo de vida del desarrollo de software desde las fases iniciales, ya que adaptarla al final será una pesadilla.
- Muchos marcos de desarrollo web incorporan funciones de accesibilidad.
- Las pruebas de accesibilidad pueden desplazarse hacia la izquierda con listas de comprobación de accesibilidad y herramientas automatizadas de auditoría estática y en tiempo de ejecución, como Codelyzer, Pa11y CI, Lighthouse, WAVE y axe-core.
- Puedes ponerte en contacto con organizaciones que prestan servicios de pruebas de usuarios, incluidos usuarios con discapacidad, para obtener opiniones en tiempo real sobre tu aplicación.

Capítulo 10. Pruebas de requisitos interfuncionales

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

¡Cuando entendemos los CFR es cuando realmente entendemos la calidad!

Las empresas suelen pensar en cientos de requisitos funcionales, buscando añadir valor para los clientes y obtener ingresos . Estos requisitos funcionales constituyen el núcleo de los servicios empresariales ofrecidos a los clientes: por ejemplo, la función de reservar un viaje con una aplicación de transporte o realizar un pago con un servicio de banca por Internet. Sin embargo, la mera implementación de estos requisitos funcionales no basta para garantizar el éxito. Imagina que quieres reservar un viaje y tienes que esperar cinco minutos para ver la lista de opciones disponibles. Probablemente podrías llamar a un taxi en ese tiempo, así que ¿para qué molestarse en utilizar la aplicación? O tal vez la aplicación hace su trabajo funcionalmente bien, pero se necesitan varios pasos para reservar un viaje. La complejidad sería frustrante, y probablemente buscarías una alternativa más fácil de usar tarde o temprano. Del mismo modo, si descubrieras que la aplicación expone tus datos personales, seguramente te desharías de ella. Estos son sólo algunos ejemplos de por qué las empresas y los equipos de software deben centrarse en los requisitos interfuncionales (CFR). Hacen que la aplicación sea completa y, lo que es más importante, imprimen alta calidad.

Las CFR son características de la aplicación que deben incorporarse a cada característica funcional. Por ejemplo, un par de CFR para una aplicación de transporte podría ser que la aplicación respondiera a los usuarios en x segundos, que los usuarios pudieran realizar cualquier acción en n pasos, y que la aplicación transmitiera y almacenara los datos de los usuarios de forma segura. Sólo cuando los CFR se construyan y prueben a fondo en todas las funciones, cualquier aplicación tendrá posibilidades de convertirse en un competidor fuerte en el mercado.

REQUISITOS INTERFUNCIONALES FRENTE A NO FUNCIONALES

A menudo oirás referirte a los CFR como *requisitos no funcionales* (NFR). Yo, junto con muchos otros en la industria del software, preferimos el término *interfuncional*, ya que hace hincapié en que los requisitos se extienden por toda la aplicación y deben construirse y probarse como parte de cada historia de usuario y característica. Además, llamarlos "no funcionales" puede causar la falsa impresión de que no son esenciales, lo que va totalmente en contra del objetivo que queremos conseguir: iconstruir una aplicación de alta calidad!

Si estás acostumbrado a pensar en los requisitos del software como funcionales o no funcionales, en este capítulo verás que algunas características "funcionales", como la autenticación y la autorización, se denominan características interfuncionales. Esto se debe a que están repartidas por toda la aplicación. Por ejemplo, tenemos que verificar la autenticidad de cada solicitud de servicio y responder sólo con la información relevante según los niveles de acceso/permisos del solicitante cada vez.

Aunque hemos estado hablando de los CFR en los capítulos anteriores (mientras considerábamos temas como el rendimiento, la seguridad, la accesibilidad y las pruebas visuales y de datos), este

capítulo se centra específicamente en llamar la atención sobre toda la gama de estos requisitos. En el proceso, adoptaremos una visión más amplia de los CFR, y discutiremos una estrategia general de pruebas de CFR que pueda servir para proporcionar una retroalimentación continua al equipo. También hablaremos de algunas metodologías y herramientas de comprobación esenciales para ayudar a aplicar la estrategia de comprobación.

Bloques de construcción

En [el Capítulo 1](#), hablamos de cómo la empresa y los clientes conciben la calidad del software de forma diferente, y de cómo ambas partes exigen una larga lista de atributos de calidad. Estos atributos se traducen esencialmente en los CFR estándar para cualquier aplicación. [La Tabla 10-1](#) enumera 30 CFR comunes, con definiciones y ejemplos de cada uno.

Tabla 10-1. Definiciones sencillas de una larga lista de CFRs

CFR	Definición simple
Accesibilidad	Capacidad del sistema para que los usuarios con discapacidad puedan acceder a la aplicación sin problemas, como se explica en el Capítulo 9 , por ejemplo, mediante la integración de lectores de pantalla.
Archivabilidad	Capacidad del sistema para almacenar y recuperar el historial de eventos y transacciones de la aplicación según sea necesario, como almacenar el historial de pedidos de compra online de un usuario.
Auditabilidad	Capacidad del sistema para rastrear los eventos y estados de una aplicación a través de registros, entradas en la base de datos, etc. Como se explica en el Capítulo 7 , esta característica ayuda a defenderse contra la amenaza del repudio.
Autenticación	Capacidad del sistema para permitir que sólo los usuarios autenticados accedan a los servicios de la aplicación en todas las capas. Por ejemplo, una simple función de inicio de sesión.
Autorización	Capacidad del sistema para restringir el acceso a los servicios de la aplicación en función de los permisos, como restringir el acceso para ver los detalles de las cuentas sólo a determinados empleados del banco.
Disponibilidad	Capacidad del sistema para prestar los servicios de la aplicación durante un periodo o umbral definido, como se expone en el Capítulo 8 .
Compatibilidad	Capacidad de dos o más sistemas para trabajar en tandem sin interrumpirse mutuamente. Por ejemplo, la capacidad de la aplicación para funcionar con una versión anterior del mismo servicio (lo que se conoce como compatibilidad con versiones anteriores).

CFR	Definición simple
Conformidad	Adherencia del sistema a los requisitos legales y a las normas del sector, como WCAG 2.0.
Configurabilidad	Capacidad del sistema para configurar el comportamiento de la aplicación con variables, como la posibilidad de configurar los tipos de autenticación multifactor.
Coherencia	Capacidad del sistema para producir resultados coherentes en entornos distribuidos sin pérdida de información, como poder mostrar los comentarios de una publicación en las redes sociales en el orden correcto, independientemente de la geolocalización del usuario final.
Extensibilidad	Capacidad del sistema para incorporar nuevas funciones, como poder añadir un nuevo tipo de método de pago a la aplicación.
Instalabilidad	Capacidad del sistema para instalarse en plataformas compatibles, como sistemas operativos y navegadores.
Interoperabilidad	Capacidad del sistema para interactuar con aplicaciones que funcionan en múltiples tecnologías y plataformas. Por ejemplo, un sistema de gestión de empleados que se integre con sistemas de seguros, productos de gestión de nóminas, sistemas de evaluación del rendimiento, etc.
Localización/internacionalización	Capacidad para escalar la aplicación a distintas regiones con una experiencia de usuario diferente, si es necesario, y traducciones de idiomas. Por ejemplo, <i>amazon.de</i> está localizado para usuarios de habla alemana. Este CFR también suele denominarse I10n/ i18n por las mismas razones que a11y (véase la nota del Capítulo 9).
Mantenibilidad	Capacidad de la aplicación para ser mantenida fácilmente a largo plazo, con código legible,

CFR	Definición simple
	pruebas, etc. Un ejemplo es la creación de nombres de métodos significativos.
Monitoreo	Capacidad del sistema para recopilar datos sobre sus actividades y alertar cuando se encuentren errores predefinidos o cuando las métricas aceptables se salgan de los límites. Por ejemplo, alertar cuando el servidor no funciona.
Observabilidad	Capacidad del sistema para analizar la información recopilada por los sistemas de monitoreo con el fin de depurar y obtener información sobre el comportamiento de la aplicación, por ejemplo, para comprender cómo se utiliza cada función durante los días punta, las semanas, etc.
Rendimiento	Capacidad del sistema para responder a tiempo a las solicitudes de los usuarios, incluso en momentos de máxima carga. Por ejemplo, la disponibilidad de viajes debe presentarse a los usuarios en x segundos, incluso en picos de carga.
Portabilidad	Capacidad de la aplicación para ser enviada a nuevos entornos, como la integración con nuevos tipos de bases de datos y proveedores de nube.
Privacidad	Capacidad del sistema para proteger los datos privados y sensibles de los usuarios, como encriptar los datos de las tarjetas de crédito mientras se almacenan en la base de datos.
Recuperabilidad	Capacidad del sistema para recuperarse de las interrupciones del sistema, por ejemplo, disponiendo de mecanismos automáticos de copia de seguridad de los datos.
Fiabilidad	Capacidad del sistema para tolerar errores y mantener continuamente los servicios y datos con precisión. Por ejemplo, las aplicaciones suelen incorporar mecanismos de reintento

CFR	Definición simple
	para hacer frente a fallos de red y otros fallos transitorios.
Informar	Capacidad del sistema para presentar informes significativos a la empresa y a los usuarios finales basados en los eventos recopilados. Por ejemplo, Amazon permite a los usuarios crear informes del historial de pedidos.
Resiliencia	Capacidad del sistema para gestionar errores y tiempos de inactividad. Por ejemplo, se pueden poner en marcha soluciones de equilibrio de carga para que las peticiones se envíen sólo a los servidores que estén en línea.
Reutilización	Capacidad del sistema para reutilizar el código y los servicios de la aplicación según sea necesario para implementar nuevas funciones; por ejemplo, reutilizar componentes de diseño en múltiples conjuntos de aplicaciones empresariales.
Escalabilidad	Capacidad del sistema para gestionar la expansión a nuevas regiones, más usuarios, etc. Por ejemplo, la mayoría de los proveedores de nubes tienen opciones para activar una función de autoescalado, que garantiza que se añadan recursos informáticos adicionales cuando hay una carga pesada.
Seguridad	Capacidad del sistema para frenar las vulnerabilidades y defenderse de posibles ataques, utilizando las herramientas y métodos analizados en el Capítulo 7 .
Apoyabilidad	Capacidad del sistema para ayudar a los nuevos desarrolladores a integrarse en los equipos y a los nuevos usuarios a integrarse en el código de la aplicación. Un ejemplo es la automatización de los pasos de configuración de la base de código y del conjunto de pruebas.
Comprobabilidad	Capacidad del sistema para simular diferentes casos de prueba y experimentar con la

CFR	Definición simple
	aplicación. Por ejemplo, crear mocks para servicios de terceros con el fin de simular diferentes casos de prueba y probar las integraciones.
Usabilidad	Capacidad del sistema para proporcionar una experiencia de usuario intuitiva, significativa y fácil. Por ejemplo, tener un diseño de navegación coherente con un panel de cabecera.

No pretende ser una lista exhaustiva; puede haber otras . En conjunto, las CFR, o *-habilidades*, como a veces se las denomina, definen las cualidades *de ejecución y evolución* de la aplicación. Las cualidades de ejecución se refieren al comportamiento de la aplicación durante el tiempo de ejecución, como la disponibilidad, la autenticación, el monitoreo y otras. Las cualidades evolutivas, como la mantenibilidad, escalabilidad, extensibilidad, etc., se refieren a la calidad del código estático de la aplicación. Cuando las cualidades de ejecución no se integran en la aplicación, los usuarios finales y la empresa serán testigos directos del impacto. Cuando no se abordan las cualidades evolutivas, los equipos de software reciben el golpe primero, y esto pronto se convierte en un problema para la empresa. Por ejemplo, los usuarios finales se frustran cuando el sistema no está disponible, y cuando el código es imposible de mantener, los miembros del equipo se frustran, con la consiguiente pérdida de productividad. Para evitar estas frustraciones, los equipos deben establecer un conjunto de CFR para la aplicación justo al principio del desarrollo y comprobarlos continuamente a lo largo del ciclo de entrega, al igual que los requisitos funcionales.

Para echar una mano en esto, ahora hablaremos de una estrategia general de pruebas CFR.

Estrategia de pruebas CFR

Hablemos del modelo FURPS, un modelo utilizado para clasificar todos los requisitos del software,¹ para empezar. Utilizaremos este modelo para establecer una estrategia de pruebas CFR de alto nivel. FURPS significa funcionalidad, usabilidad, fiabilidad, rendimiento y capacidad de soporte. Los temas pueden elaborarse como sigue:

Funcionalidad

Esta categoría de requisitos de software puede experimentarse como flujos de usuario en la aplicación, como el flujo de inicio de sesión, el flujo de disponibilidad de viajes y el flujo de reservas.

Usabilidad

Esta categoría representa el conjunto de requisitos de que afectan a la experiencia del usuario, como la calidad visual, la compatibilidad del navegador, la accesibilidad, la facilidad de uso, etc.

Fiabilidad

Estos requisitos contribuyen a que sea una aplicación coherente, tolerante a fallos y recuperable.

Rendimiento

Estos requisitos están relacionados con los KPI del backend y las métricas de rendimiento del frontend , como se explica en [el Capítulo 8](#).

Apoyabilidad

Esta categoría incluye todas las cualidades evolutivas del código , como la mantenibilidad, la comprobabilidad, el código seguro, etc.

Los CFR de [la Tabla 10-1](#) pueden visualizarse en la misma línea. Por ejemplo, la accesibilidad, tal y como se trató en [el Capítulo 9](#), se manifiesta a través de características funcionales, como añadir transcripciones a los vídeos, y también a través del diseño de la aplicación. Por tanto, el enfoque de las pruebas de accesibilidad comprenderá métodos y herramientas utilizados para probar tanto la funcionalidad como la usabilidad. Del mismo modo, una forma de incorporar la seguridad es añadir características funcionales relacionadas con la autenticación, como el inicio de sesión del usuario, e imbuir prácticas relacionadas con la seguridad en el código estático.

Esta sección presenta estrategias de prueba para cada uno de estos cinco temas, como se representa en [la Figura 10-1](#). Para formular la estrategia de pruebas de CFR específica de tu proyecto, descompón los distintos aspectos de los CFR y adopta los métodos y herramientas adecuados en función de las prioridades de tu proyecto.

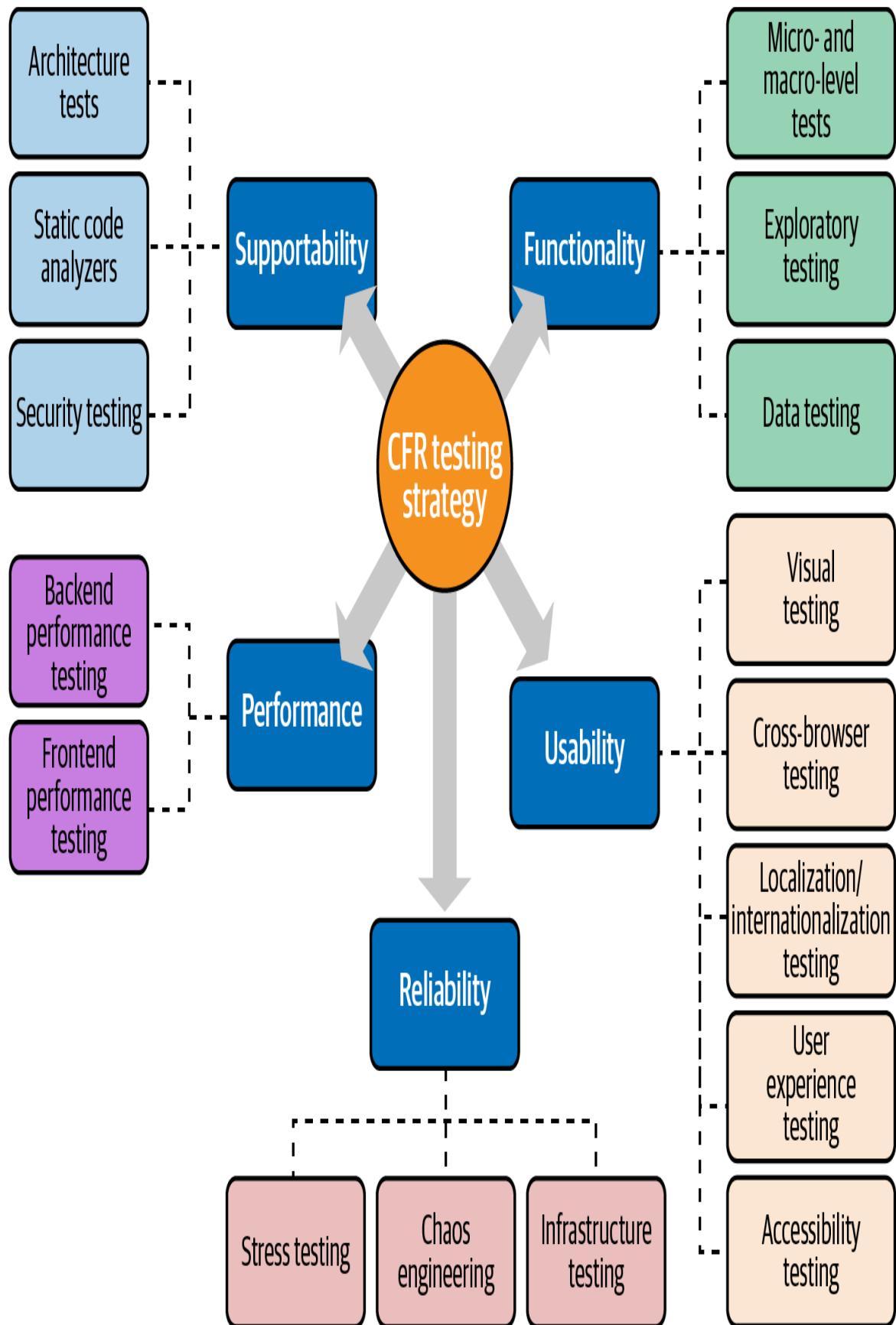


Figura 10-1. Una estrategia de pruebas CFR, descompuesta en los cinco temas

Funcionalidad

Para probar los aspectos funcionales de los CFR, pueden emplearse las herramientas y métodos de pruebas funcionales manuales exploratorias y automatizadas que se comentan en los Capítulos 2 y 3, en las distintas capas de la aplicación. Para reiterar, pueden utilizarse herramientas como Postman, Selenium WebDriver, REST Assured y JUnit para automatizar estos aspectos funcionales y obtener retroalimentación continua. Además, las herramientas y métodos de comprobación de datos comentados en el Capítulo 5 serán esenciales para probarlos.

Una llamada de atención especial para probar las características funcionales relacionadas con el cumplimiento, como la función de autenticación fuerte del cliente que forma parte de la PSD2 o las características funcionales relacionadas con el GDPR, es que es crucial recopilar la información adecuada sobre esas normativas e implicar al equipo legal tanto en la fase de pruebas como en la de recopilación de requisitos. Para una referencia rápida, se incluye una sección sobre requisitos normativos como parte de la siguiente sección del capítulo (ver "Pruebas de conformidad").

Usabilidad

Para abordar metódicamente las pruebas de usabilidad , podemos deconstruir la usabilidad en algunos aspectos, como la calidad visual, la compatibilidad entre navegadores, la localización/internacionalización, el diseño de la experiencia del usuario y la accesibilidad. En el Capítulo 6 hablamos de herramientas y enfoques para comprobar la calidad visual y la compatibilidad entre navegadores, y en el Capítulo 9, de la accesibilidad. Vamos a discutir aquí los enfoques de comprobación de los aspectos restantes:

Pruebas de localización/internacionalización

Las pruebas de localización pueden abordarse de varias maneras. Si la apariencia de la interfaz de usuario varía según la configuración regional, debes realizar pruebas visuales en . Si la apariencia no varía, sino que sólo cambian el idioma y características como los formatos de fecha y dinero, puedes confiar en las pruebas unitarias y en las pruebas manuales. Por ejemplo, puedes añadir pruebas unitarias para comparar los archivos de cadenas de las locales en busca de claves que falten, aparte de las validaciones de los formatos de fecha y dinero. Sin embargo, cuando cambia el idioma, a veces se ve afectada la longitud del texto, lo que puede provocar cambios en el diseño de la interfaz de usuario. En esos casos, puedes adoptar de nuevo prácticas de pruebas visuales.

La comprobación manual del texto específico de una lengua debe seguir una determinada secuencia de pasos para evitar la duplicación de esfuerzos. El primer paso es obtener un texto significativo para todos los elementos, mensajes, etc., de una persona que conozca el idioma, seguido de la aprobación del propietario del producto o de cualquier representante aprobado por la empresa. Lo siguiente es documentar el texto correcto en cada historia de usuario para permitir el desarrollo y la prueba manual de la historia. La mayoría de las veces, cuando se omiten estos pasos, a los desarrolladores no les queda más remedio que llenar el texto utilizando un servicio de traducción en línea, lo que se traduce en un doble esfuerzo de pruebas antes y después de obtener las cadenas aprobadas.

Es importante tener en cuenta que aplazar estas pruebas de localización hasta justo antes del lanzamiento supone el riesgo de encontrarse con un diseño de interfaz de usuario roto al final del ciclo de entrega, ya que existe la posibilidad de que el texto

traducido no se ajuste al diseño de los elementos, como se ha mencionado anteriormente.

NOTA

Las pruebas funcionales basadas en la interfaz de usuario no deben utilizarse para verificar todo el texto de la aplicación, ya que eso las haría muy lentas. Utiliza estas pruebas sólo para verificar los flujos funcionales en las distintas localizaciones, si son diferentes. En tales casos, puedes reutilizar las pruebas funcionales basadas en la interfaz de usuario parametrizando las cadenas utilizadas en las afirmaciones y los identificadores de elementos, siempre que tu estrategia general de pruebas se ajuste a la pirámide de pruebas.

Experiencia del usuario

La experiencia de usuario engloba todos los aspectos de la aplicación relacionados con el diseño , como lo intuitivos que son los flujos de usuario, cuántos clics tarda el usuario en obtener la información que necesita, si los iconos transmiten el significado correcto, si la paleta de colores de la aplicación es del agrado de los usuarios finales, etc. Estos aspectos se investigan al principio del proyecto y se incorporan al diseño. Por ejemplo, en un proyecto móvil de venta al por menor en el que trabajé, descubrimos que la gente de Italia prefería ver colores vivos, como el rojo vibrante, y diseñamos la aplicación con esa paleta de colores.

Como práctica general, debes incluir aspectos de la experiencia del usuario en las pruebas exploratorias manuales para cada historia de usuario. El grupo de Nielson y Norman ha realizado una amplia investigación sobre el diseño de la experiencia del usuario y ha recopilado una lista de **10 heurísticas de usabilidad** a seguir, que pueden incorporarse a las pruebas. La mayoría de las veces, los propietarios del producto y los diseñadores de UX colaboran en estas pruebas. También hay herramientas como

UserZoom y Optimal Workshop que pueden utilizarse para realizar pruebas de UX en los prototipos de diseño con usuarios finales reales. He visto que estas pruebas, cuando se realizan periódicamente durante el ciclo de entrega con diferentes grupos de usuarios finales, dan como resultado una mejora significativa del diseño.

Las pruebas A/B son otra forma de obtener información en tiempo real sobre la experiencia del usuario en producción. Aunque se llama prueba, en realidad es más experimentación: consiste en presentar diferentes diseños de UX de la misma función como prototipos a dos grupos de usuarios finales diferentes en producción y recopilar datos sobre los comportamientos de los usuarios para que el equipo de producto pueda decidir el diseño final. Por ejemplo, un experimento sencillo sería saber si un botón de Venta tiene más probabilidades de ser pulsado si es rojo o azul. En un experimento de este tipo, se presentan los botones rojo y azul a distintos grupos de usuarios en producción, y se recogen y analizan los datos de uso durante un periodo determinado. Este tipo de experimento puede requerir **capacidades de ciencia de datos**, y normalmente un equipo de propietarios de producto, científicos de datos, desarrolladores y diseñadores de experiencia de usuario trabajan juntos en tales experimentos .

Fiabilidad

Según **la Tabla 10-1**, los CFR que contribuyen a la fiabilidad de la aplicación son la recuperabilidad, la resistencia, la auditabilidad , la archivabilidad, la elaboración de informes, el monitoreo, la observabilidad y la coherencia. Muchos aspectos de la fiabilidad, como la gestión de errores, los mecanismos de reintento, los mecanismos de reserva para puntos únicos de fallo, las medidas para garantizar la coherencia de los datos y las integraciones con herramientas de terceros para los servicios de monitoreo,

observabilidad e información, pueden experimentarse como flujos de usuario. Aquí pueden desplegarse enfoques de pruebas funcionales, como los analizados en los Capítulos 2 y 3. Aparte de éstos, los otros métodos de prueba que contribuyen a las pruebas de fiabilidad son los siguientes:

Ingeniería del caos

La Ingeniería del Caos es una forma de desenterrar fallos inherentes a la aplicación que podrían provocar interrupciones del sistema, fallos y otros desastres, haciendo que la aplicación no sea fiable. Normalmente, este método descubre las incógnitas desconocidas y es inmensamente útil en sistemas a gran escala. La Ingeniería del Caos se trata en detalle en la siguiente sección del capítulo.

Pruebas de infraestructura

La infraestructura es una de las muchas partes importantes de una aplicación que contribuyen a su fiabilidad y recuperabilidad. Si se cae, se cae todo. Además, la capa de infraestructura tiene que estar cableada adecuadamente para soportar las capacidades de autoescalado, alerta/monitorización, equilibrio de carga y archivo. Aunque las pruebas centradas en la capa de infraestructura aún no se han generalizado, están ganando terreno debido a la creciente necesidad de las empresas de escalar ampliamente. También trataremos este tema en detalle en la siguiente sección del capítulo.

Rendimiento

En el Capítulo 8 hablamos de la importancia del rendimiento y de una selección de herramientas y métricas para las pruebas de rendimiento tanto del frontend como del backend. Para reiterar, algunas de las métricas clave son la disponibilidad, el tiempo de respuesta y la concurrencia, y las herramientas que pueden ayudar

con las pruebas de rendimiento incluyen JMeter, WebPageTest y Lighthouse. Un punto adicional a tener en cuenta es que las pruebas de rendimiento sirven para cumplir los requisitos de escalabilidad identificando el umbral de avería del sistema, y por tanto contribuyen también a mejorar la fiabilidad de la aplicación.

Apoyabilidad

La soportabilidad se refiere a todas las cualidades del código evolutivo , como la compatibilidad, la configurabilidad, la extensibilidad, la instalabilidad, la interoperabilidad, la portabilidad, la mantenibilidad, la reutilizabilidad, la seguridad y la comprobabilidad. Algunas de sus manifestaciones funcionales, como la configurabilidad de las características funcionales, la compatibilidad con los protocolos necesarios, la instalabilidad en los sistemas operativos adecuados, las características de interoperabilidad, etc., pueden probarse utilizando los enfoques de pruebas funcionales comentados anteriormente en el libro, con la configuración del entorno y los stubs adecuados. Otros enfoques para probar la compatibilidad son

Pruebas de arquitectura

Las pruebas de arquitectura son añadidas para afirmar un conjunto de características arquitectónicas, como verificar que las clases correctas están bajo los paquetes correctos (garantizando así la reutilización). Estas pruebas automatizadas proporcionan información al equipo en caso de desviaciones de las características arquitectónicas esenciales que se diseñaron para atender a las CFR, como la reutilización, la portabilidad, la mantenibilidad, etc. Hablaremos de algunas herramientas que pueden utilizarse para escribir dichas pruebas en "**Pruebas de arquitectura**".

Analizadores estáticos de código

Muchas herramientas realizan análisis estáticos de código y proporcionan información útil que sirve para mejorar la mantenibilidad. Por ejemplo, [Checkstyle](#) garantiza que el equipo se ciña a un estilo de codificación común. [PMD](#) es una herramienta que informa de problemas como variables no utilizadas, bloques de captura vacíos, código duplicado, etc. También permite al equipo añadir reglas personalizadas específicas para las normas del proyecto. [ESLint](#) es una herramienta similar para comprobar el código JavaScript en busca de posibles errores de estilo y código, y [SonarQube](#) es una herramienta ampliamente adoptada que ayuda a evaluar la cobertura del código y a escanear en busca de vulnerabilidades. En capítulos anteriores, también hablamos de otros analizadores de código estático que examinan el código para garantizar que es seguro y accesible.

Utilizando estos métodos y herramientas, puedes desplazar tus pruebas de CFR hacia la izquierda. Como se explica en [el Capítulo 4](#), estas CFR pueden probarse continuamente junto con las pruebas funcionales como parte de la IC, lo que permite al equipo obtener información continua sobre todas las dimensiones de la calidad y, por tanto, entregar continuamente software de alta calidad a sus clientes!

Otros métodos de prueba CFR

Para ayudarte a cumplir el objetivo de desplazar las pruebas CFR hacia la izquierda y poder realizar entregas continuas, aquí se tratan con más detalle varios de los métodos de pruebas CFR introducidos en la sección anterior, como la Ingeniería del Caos, las pruebas de arquitectura y las pruebas de infraestructura. También puedes leer sobre un conjunto de requisitos normativos comúnmente aplicados hacia el final de la sección, que apoyarán tus esfuerzos de pruebas de cumplimiento.

NOTA

El título de esta sección subraya el hecho de que ya hemos hablado de varios métodos y herramientas de comprobación del CFR, en [los Capítulos 5 a 9](#).

Ingeniería del caos

La fiabilidad de las aplicaciones es uno de los CFR críticos de , ya que cualquier interrupción del servicio se traduce directamente en una pérdida para la empresa. Un [estudio de Gartner](#) de 2014 estimó que el coste del tiempo de inactividad oscila entre 140.000 y 540.000 dólares por hora para algunas empresas, y no me sorprendería que el coste fuera aún mayor en 2022. En reconocimiento de la importancia de la fiabilidad, los productos establecidos en el mercado, como los Servicios Web de Amazon, se esfuerzan por conseguir un tiempo de actividad del 99,999%, es decir, un tiempo de inactividad acumulado de sólo 5 minutos y 15 segundos al año.

Algunos de los factores que pueden provocar tiempos de inactividad son errores en la aplicación, puntos únicos de fallo en la arquitectura, problemas de red, fallos de hardware, altas cargas inesperadas de tráfico y problemas con servicios de terceros de los que depende la aplicación. La mayoría de estos factores se tienen en cuenta al diseñar la arquitectura, y los equipos también toman las medidas preventivas pertinentes durante el desarrollo. Por ejemplo, el método de retroceso exponencial se adopta ampliamente para gestionar el tiempo de inactividad de un servicio: prescribe que la frecuencia de solicitud a un servicio caído disminuya exponencialmente para que tenga tiempo de respiración para recuperarse rápidamente. Del mismo modo, el modelo de implementación azul/verde se aplica con frecuencia para evitar el tiempo de inactividad durante las actualizaciones del sistema; funciona teniendo dos instancias de producción idénticas, en las que

una está activa y la otra se utiliza para actualizar, y luego se cambia para que sea la instancia activa. Aparte de métodos como éstos, los equipos gestionan el tiempo de inactividad de forma preventiva de varias maneras, como teniendo réplicas para compartir la carga elevada, infraestructura de autoescalado, gestión adecuada de errores de las entradas, etc. Sin embargo, a pesar de todos estos esfuerzos, los sistemas distribuidos a gran escala plantean retos discretos a la fiabilidad de la aplicación en forma de flujos de trabajo enrevesados, dependencias de múltiples capas, fallos de terceros, errores de los sistemas descendentes, etc., que no pueden preverse fácilmente y acaban provocando tiempos de inactividad.

Consideremos un ejemplo hipotético. Un equipo de 50 miembros trabaja en una aplicación distribuida a gran escala y configura dos instancias por separado para atender a clientes de EE.UU. y el Reino Unido. Configuraron cada instancia para que redirigiera a la otra cuando una se cayera y crearon capacidades funcionales en la aplicación para gestionar las solicitudes de ambas regiones. El equipo probó la funcionalidad y el flujo de redirecciónamiento. También comprobaron el rendimiento de la aplicación bajo carga. Sin embargo, cuando la instancia del Reino Unido dejó de funcionar por problemas técnicos al mismo tiempo que se producía un pico de ventas en EE.UU. y todas las solicitudes del Reino Unido se redirigieron a la instancia de EE.UU., la aplicación acabó dando errores a todos los usuarios. Más tarde se descubrió que la causa estaba en uno de los sistemas descendentes de terceros con un límite de solicitudes por hora, que empezaba a lanzar errores cuando se superaba el límite de velocidad. En la práctica, se trata de uno de esos casos de perímetro difíciles de localizar. El equipo había actuado con la diligencia debida, pero en realidad es difícil que alguien conozca todos los detalles de los sistemas distribuidos a tan gran escala.

Ese hipotético equipo no fue el único que tuvo esta experiencia. Netflix también tuvo experiencias problemáticas cuando su servicio

se convirtió en nativo de la nube: las instancias de la nube se enfrentaban a interrupciones imprevistas debidas a diversos problemas, lo que provocaba pérdidas y prolongaba las horas de trabajo de los ingenieros. Se lo tomaron como un reto, imitando deliberadamente los fallos y resolviendo los problemas de su aplicación uno a uno, hasta que se hizo totalmente resistente a esas interrupciones imprevistas. Para lograr este objetivo diseñaron una herramienta llamada Chaos Monkey, que hacía caer una instancia aleatoria de un clúster cada día durante las horas de trabajo, con los ingenieros aplicando las medidas de seguridad adecuadas. Este enfoque garantizó que se abordaran todos y cada uno de los fallos imprevistos inherentes a su sistema, convirtiéndolo en resistente y fiable. Basándose en este éxito, evolucionaron y cristalizaron la práctica, llamándola *Ingeniería del Caos*.

Una definición formal del libro *Chaos Engineering* de Nora Jones y Casey Rosenthal (O'Reilly) es la siguiente:

La Ingeniería del Caos es la disciplina que consiste en experimentar en un sistema distribuido para crear confianza en la capacidad del sistema para soportar condiciones turbulentas en la producción.

En otras palabras, consiste en realizar experimentos con la aplicación, simulando errores, interrupciones y otros escenarios inesperados, y observando el comportamiento de la aplicación. Esta práctica se está adoptando ampliamente en la industria del software, y muchas empresas la han hecho evolucionar para adaptarla a sus necesidades. A partir de los aprendizajes colectivos de la industria, las características fundamentales de la Ingeniería del Caos pueden describirse como sigue:

- Se trata más de experimentar que de probar, es decir, no se trata de verificar los comportamientos esperados del sistema ante problemas desconocidos, sino de observar el

comportamiento del sistema en situaciones inesperadas y obtener información.

- El propósito de experimentar es ganar confianza en la fiabilidad y resistencia del sistema. Puedes optar por *no* experimentar cuando tengas suficiente confianza en la capacidad de tu sistema para hacer frente a turbulencias desconocidas.
- Es especialmente beneficioso cuando desarrollas un sistema distribuido a gran escala .
- La Ingeniería del Caos no es responsabilidad de una función concreta, como los ingenieros de DevOps o los probadores. Es una actividad de equipo en la que todas las partes interesadas colaboran para diseñar el experimento, llevarlo a cabo y depurar el comportamiento.

Tal vez, si hubieran realizado esos experimentos de caos, ese equipo de 50 miembros habría detectado antes el problema de limitación de velocidad y ihabría evitado el desastroso apagón!

Experimento del caos

Si piensas orquestar experimentos de caos, el equipo de Netflix recomienda realizarlos directamente en producción, ya que las variables de la vida real son muy difíciles de simular en un entorno de pruebas. También recomiendan diseñar capacidades para pausar los experimentos y revertir el sistema a la normalidad. Hoy en día existen varias herramientas que ayudan a programar experimentos de caos, como Chaos Toolkit y ChaosBlade, para que no tengas que poner y quitar cosas manualmente en producción.

Para realizar un experimento de caos, desarrolla primero una hipótesis que pueda poner en duda la fiabilidad de la aplicación, junto con un equipo interfuncional. A continuación, define una hipótesis de estado estacionario, que es el comportamiento previsto de la aplicación durante el experimento. Programa el experimento

con la herramienta que elijas y ejecútalo en producción. Si la herramienta te avisa de que el experimento ha fallado -es decir, que no se ha cumplido la hipótesis de estado estacionario-, tu equipo interfuncional puede entrar en acción.

Para que veas cómo funciona una de las herramientas de experimentación del caos, [el Ejemplo 10-1](#) muestra la configuración de un experimento sencillo utilizando [el Chaos Toolkit](#), una herramienta de código abierto escrita en Python. El experimento está programado para simular un problema técnico (en este caso, borrando un archivo de configuración en la instancia actual de la aplicación) y para comprobar si una instancia alternativa de la aplicación sigue funcionando.

Ejemplo 10-1. Experimento de caos para simular un problema técnico y observar el comportamiento de la aplicación

```
{  
    "version": "1.0.0",  
    "title": "Application should still be up if there are technical  
issues",  
    "description": "When a particular config file is missing, application  
should  
                still be up from another instance",  
    "contributions": {  
        "reliability": "high",  
        "availability": "high"  
    },  
    "steady-state-hypothesis": {  
        "title": "Application is up and running",  
        "probes": [  
            {  
                "type": "probe",  
                "name": "homepage-must-respond-ok",  
                "tolerance": 200,  
                "provider": {  
                    "type": "http",  
                    "timeout": 2,  
                    "url": "https://www.example.com/"  
                }  
            }  
        ]  
    }  
}
```

```

        ],
    },
    "method": [
        {
            "type": "action",
            "name": "file-be-gone",
            "provider": {
                "type": "python",
                "module": "os",
                "func": "remove",
                "arguments": {
                    "path": "/path/config-file"
                }
            },
            "pauses": {
                "after": 1
            }
        }
    ]
}

```

El guión comienza describiendo la intención del experimento y etiquetándolo como un caso de prueba que contribuye a la alta fiabilidad y disponibilidad. A continuación, elabora la hipótesis de estado estacionario y el método para desencadenar el problema técnico. El método utiliza la capacidad de la herramienta para eliminar un archivo de una ruta determinada y hace una pausa de 1 segundo antes de que la herramienta intente verificar la hipótesis de estado estacionario, utilizando una de las sondas de la herramienta para golpear la URL de la aplicación y comprobar si devuelve un código de estado 200 en 2 segundos. Este experimento puede ejecutarse desde la línea de comandos y observarse.

Supongamos que ejecutamos el experimento y falla. Investigando más a fondo, descubrimos que la aplicación tarda 4 segundos en responder en lugar de los 2 segundos esperados, debido a los obstáculos en el redireccionamiento a la instancia alternativa. Esa será una valiosa información obtenida de este sencillo experimento de caos.

El Conjunto de Herramientas del Caos proporciona muchas otras API para realizar diversos tipos de experimentos, y es fácil configurarlos como archivos JSON como el que se muestra aquí. También puede generar informes HTML al final de los experimentos.

Pruebas de arquitectura

Al principio de cualquier proyecto, se reflexiona y elabora una lista de requisitos funcionales y multifuncionales, y se establece un diseño de arquitectura propicio. Por ejemplo, digamos que para garantizar la mantenibilidad y la reutilización, el diseño de la arquitectura propone que la aplicación tenga capas separadas. Además, como el rendimiento es una prioridad, los mecanismos de almacenamiento en caché se colocan en las capas adecuadas. Pero existe una ley universal que puede trastocar estas decisiones de diseño bien meditadas, conocida como la ley de Conway. En su artículo "[¿Cómo inventan los comités?](#)" Melvin Conway afirma que las estructuras de los equipos, en particular las vías de comunicación entre las personas, influyen inevitablemente en el diseño final del producto. Los equipos individuales que trabajan en partes más pequeñas de un gran sistema optimizarán inadvertidamente sus partes respectivas sin tener en cuenta las necesidades generales. Por ejemplo, un equipo puede optar por dar prioridad al rendimiento frente a la reutilización y pasar por alto las capas, lo que dará lugar a la necesidad de volver a trabajar más adelante. Aquí es donde las pruebas de arquitectura, cuando se colocan correctamente para vigilar las características arquitectónicas esenciales, resultan útiles: proporcionan información a los equipos cuando se desvían del panorama general.

Herramientas como [ArchUnit](#) para Java y [NetArchTest](#) para .NET, entre [otras](#), pueden utilizarse para escribir dichas pruebas. Por ejemplo, puedes introducir pruebas de arquitectura para comprobar las dependencias cíclicas, de modo que se preserve continuamente la mantenibilidad , o para asegurarte de que los paquetes son

independientes, de modo que sean reutilizables. Las pruebas ArchUnit son similares a las pruebas JUnit y pueden ejecutarse como parte de una canalización CI. **El Ejemplo 10-2** muestra una prueba ArchUnit para afirmar que todas las clases del servicio de gestión de pedidos residen en el paquete oms, para garantizar la reutilización. Así, siempre que sea necesario incluir una clase fuera de las responsabilidades del servicio de gestión de pedidos, el equipo se verá empujado a discutir el panorama general y decidir dónde encaja adecuadamente.

Ejemplo 10-2. Prueba ArchUnit para asegurar la reutilización

```
@Test
public void order_classes_must_reside_in_oms_package() {

    classes().that().haveNameMatching("*order*").should().resideInAPackage(".oms..")
        .as("order classes should reside in the package
'..oms..'")
        .check(classes);
}
```

Del mismo modo, **JDepend** es una herramienta que produce métricas sobre la calidad del diseño en términos de extensibilidad, mantenibilidad y reutilización. Realiza un análisis estático del código en todas las clases Java y da diferentes puntuaciones de diseño para un paquete Java determinado. (**NDepend** es una herramienta paralela en el mundo .NET.) JDepend utiliza el número de clases abstractas e interfaces de un paquete como medida de su extensibilidad, comprueba las dependencias de paquetes externos y emite alertas cuando hay dependencias no deseadas, y comprueba las dependencias cíclicas de los paquetes. Las pruebas de JDepend pueden escribirse como pruebas JUnit e integrarse con CI para obtener información continua sobre la calidad de la arquitectura.

El ejemplo 10-3 muestra una prueba JDepend que comprueba si los paquetes A y B dependen entre sí, introduciendo dependencias cíclicas y dificultando así la reutilización.

Ejemplo 10-3. Una prueba JDepend para afirmar las dependencias cíclicas de un paquete

```
import java.io.*;
import java.util.*;
import junit.framework.*;

public class PackageDependencyCycleTest extends TestCase {
    private JDepend jdepend;

    protected void setUp() throws IOException {
        jdepend = new JDepend();
        jdepend.addDirectory("/path/to/project/A/classes");
        jdepend.addDirectory("/path/to/project/B/classes");
    }

    public void testAllPackages() {
        Collection packages = jdepend.analyze();
        assertEquals("Cycles exist",
                    false, jdepend.containsCycles());
    }
}
```

En la misma línea, se pueden escribir pruebas para verificar que un paquete sólo tiene las dependencias esperadas o que no tiene ninguna dependencia. De este modo, los equipos pueden obtener información continuamente cada vez que se produzca un cambio no deseado en las características críticas de la arquitectura.

Pruebas de infraestructura

El término *infraestructura* a alto nivel se refiere a los recursos computacionales (por ejemplo, máquinas, VMs, contenedores), estructuras de red (por ejemplo, VPNs, entradas DNS, proxies, puertas de enlace), y recursos de almacenamiento (AWS S3, SQL Server, sistemas de gestión de secretos, etc.) necesarios para soportar el buen funcionamiento de la aplicación. Las pruebas de infraestructura implican probar la instalación y configuración de estos recursos. Se trata de un área emergente en las pruebas.

La necesidad de probar la infraestructura surge principalmente de la creciente demanda de escalar las aplicaciones. Esta demanda se debe principalmente a que las empresas, una vez que tienen éxito, quieren ampliar rápidamente sus servicios en línea a nuevas regiones y empezar a dar servicio a un mayor número de clientes. Para permitir ese rápido escalado, deben tener la capacidad de replicar la pila de aplicaciones existente de extremo a extremo, incluida la configuración de la infraestructura, en poco tiempo, a ser posible automáticamente, con un solo clic. Aunque la mayoría de los equipos de software tienen un proceso automatizado establecido para probar, agrupar e implementar la aplicación en cualquier entorno con un solo clic, no siempre tienen la misma capacidad en el lado de la infraestructura, lo que introduce una brecha en su capacidad para escalar rápidamente. Aquí es donde la práctica de *la Infraestructura como Código* (IaC) se vuelve muy útil.

El término IaC se refiere a la práctica de diseñar la configuración de la infraestructura como código reutilizable, igual que el código de la aplicación, para potenciar la entrega continua y la escalabilidad. Por ejemplo, se escribe código para poner en marcha una instancia en la nube con 3 GB de memoria utilizando las API del proveedor de la nube, configurar reglas del equilibrador de carga específicas de la aplicación y configurar un cortafuegos. Estas funciones deben probarse para que el mismo código pueda utilizarse para poner en marcha nuevas instancias de infraestructura cuando haya una carga elevada o para permitir la expansión a nuevas regiones.

Terraform de HashiCorp es una herramienta de código abierto ampliamente adoptada para programar el código de la infraestructura utilizando un estilo de codificación declarativo. Permite que el código funcione en múltiples proveedores de nubes. He aquí algunas cosas que hay que tener en cuenta al probar código de infraestructura escrito con Terraform en distintas fases del camino hacia la producción:

- Terraform proporciona el comando `terraform validate` para comprobar si hay errores de sintaxis en el código de Terraform, que puede aplicarse ya en la fase de desarrollo.
- **TFLint**, un complemento de linting para Terraform, puede ayudar a analizar el código estático de la infraestructura en busca de sintaxis obsoleta, desviaciones de las buenas prácticas como convenciones de nomenclatura, etc. TFLint también puede comprobar si los tipos de imagen especificados son ofrecidos por proveedores de nube populares como AWS, Azure, etc.
- Durante el proceso de desarrollo incremental, Terraform puede comparar los últimos cambios de código con el estado del entorno existente y presentar una vista previa de los cambios como medida de seguridad antes de la ejecución. Por ejemplo, si los cambios de código provocan un borrado involuntario de la base de datos, la función de vista previa salvará el día! El comando para ello es `terraform plan`. También puedes escribir pruebas automatizadas contra la salida de este comando para verificar ciertos aspectos, como el cumplimiento de la política de seguridad.
- El siguiente paso es desplegar el código de infraestructura para crear instancias reales en la nube y verificar si las instancias tienen los recursos de infraestructura previstos; por ejemplo, si la instancia se ejecuta dentro de la subred privada y tiene el espacio de disco necesario. Estos casos de prueba pueden automatizarse utilizando herramientas como Terratest, AWSSpec, Inspec y Kitchen-Terraform, y añadirse a CI.
- Luego viene la prueba de extremo a extremo de los componentes de la infraestructura. Tenemos que comprobar si los componentes pueden interactuar entre sí de la forma esperada; por ejemplo, si el servidor web puede realizar una llamada a los servicios de la aplicación. Estas pruebas, en cierto modo, quedan cubiertas cuando la implementación del código

de la aplicación tiene éxito y las pruebas funcionales se ejecutan sin problemas. Pero también es posible detectar estos problemas antes de desplegar la aplicación escribiendo pruebas de infraestructura utilizando una combinación de las herramientas mencionadas anteriormente.

Kief Morris, autor del libro *Infrastructure as Code* (O'Reilly), sugiere que la distribución de las pruebas de infraestructura en las distintas capas puede formar un patrón de diamante en lugar de una pirámide. Esto se debe a que las pruebas unitarias para el código declarativo de bajo nivel, como el de Terraform, pueden no ser de mucha utilidad y se recomienda reducirlas al mínimo. Así que, dependiendo de la naturaleza del código de la infraestructura, deberíamos optar por añadir las pruebas pertinentes en las capas adecuadas.

Aparte de las pruebas funcionales de extremo a extremo, los otros aspectos que hay que probar cuando se trata de la infraestructura son:

Escalabilidad

Debemos comprobar que las instancias se autoescalan en función de la carga y verificar que las funciones de la aplicación funcionan sin problemas tras el escalado.

Seguridad

La seguridad de la infraestructura es un aspecto crítico que hay que comprobar. Herramientas como **Snyk IaC** comprueban posibles vulnerabilidades en el código de la infraestructura durante el desarrollo. Algunos de los casos de prueba de seguridad, como la comprobación de puertos abiertos inesperados, instancias públicas y privadas adecuadas, etc., pueden probarse manualmente y escribiendo pruebas de infraestructura automatizadas.

Conformidad

A veces, el código de la infraestructura debe adherirse a las políticas y características de cumplimiento de . Por ejemplo, para cumplir la norma PCI DSS (la norma PCI DSS se trata en el siguiente apartado), deben configurarse los cortafuegos adecuados. Para comprobar el cumplimiento de las normas, HashiCorp ofrece **Sentinel** para empresas.

Una herramienta de código abierto para comprobar las características de cumplimiento en Terraform es **terraform-compliance**. La herramienta se basa en Python y proporciona una capa de desarrollo basada en el comportamiento, igual que Cucumber, para escribir pruebas. La herramienta ejecuta las pruebas contra la salida del comando `terraform plan` en lugar de la instancia real.

Operatividad

Todas las demás funciones operativas, como , como el archivo de registros para la auditabilidad, la integración de herramientas de monitoreo, las funciones de mantenimiento automatizado, etc., también deben probarse.

Dependiendo de la complejidad y naturaleza del código de la infraestructura, puedes optar por escribir pruebas automatizadas para estos casos e integrarlas con CI. Muchas de las herramientas para pruebas automatizadas de infraestructura siguen evolucionando y requieren conocimientos de codificación que van más allá de un solo lenguaje. Por ejemplo, Terratest utiliza GoLang, **terraform-compliance** utiliza Python, y AWSSpec utiliza Ruby. Además, muchas herramientas de pruebas automatizadas pueden requerir una infraestructura real para estar en funcionamiento, lo que conlleva costes. Teniendo en cuenta estas limitaciones, puedes elaborar una

estrategia de pruebas de infraestructura específica para las necesidades de tu aplicación.

Pruebas de conformidad

Dos normativas de aplicación habitual en la web son el **GDPR** y las WCAG 2.0. Hablamos extensamente de las WCAG 2.0 en [el Capítulo 9](#). Aquí exploraremos brevemente el GDPR, y luego echaremos un vistazo rápido a algunas normativas relacionadas con los pagos que deberías conocer.

NOTA

Esta sección sólo pretende servir de breve introducción a estos requisitos normativos. Se recomienda a los equipos de software que se pongan en contacto con asesores jurídicos para obtener los detalles específicos de su aplicación y dominio.

Reglamento General de Protección de Datos (RGPD)

El objetivo principal del GDPR es proteger los datos privados de los ciudadanos de la UE en . Si tu objetivo es vender productos a ciudadanos de la UE, entonces tu sitio web estará sujeto al cumplimiento del GDPR. Del mismo modo, si una escuela de EE.UU. permite la admisión de ciudadanos de la UE a través de su sitio web, tendrá que cumplir los requisitos del GDPR. El incumplimiento puede acarrear fuertes sanciones, de hasta el 4% de los ingresos anuales de la empresa.

NOTA

Los distintos países también tienen sus propias leyes de protección de datos y privacidad. En abril de 2022, el **71% de los países** de todo el **mundo** contaban con una legislación adecuada en materia de protección de datos y privacidad. Por ejemplo, Canadá tiene la Ley de Protección de la Privacidad del Consumidor (CPPIA) y el Reino Unido tiene su propia versión del GDPR (post-Brexit).

Según el GDPR, los *datos privados* son cualquier información que, por sí sola o combinada con otra información, pueda utilizarse para identificar a un individuo vivo. El origen racial o étnico del individuo, sus creencias religiosas o filosóficas, sus opiniones políticas, su orientación sexual, sus datos genéticos, sus datos biométricos, sus condenas penales pasadas o presentes, etc., se clasifican como *datos personales sensibles*, que deben protegerse cuidadosamente. Incluso muchos identificadores en línea, como direcciones IP, direcciones MAC, identificadores de dispositivos móviles, cookies, identificadores de cuentas de usuario y otros datos generados por el sistema que pueden utilizarse para identificar a una persona viva, están bajo la protección del GDPR. Para proteger los datos, el GDPR recomienda que los equipos de desarrollo apliquen los principios de la **Privacidad por Diseño** (el marco de la **Privacidad por Diseño**, desarrollado por la Dra. Ann Cavoukian, establece siete principios fundamentales que se centran en prevenir cualquier suceso que invada la privacidad).

Algunas de las medidas técnicas que puedes aplicar son la protección de los datos en reposo mediante sales dinámicas y técnicas hash, la encriptación de los datos en tránsito, la adhesión al principio del menor privilegio, la seudonimización, la anonimización de los datos y otras medidas generales de seguridad de los datos que se tratan en el **Capítulo 7**.

El GDPR también protege los derechos de los usuarios a controlar sus datos **de varias formas**, como las siguientes:

Derecho a ser informado

Tienes que informar a los usuarios de la aplicación de cómo se utilizan sus datos personales. Esto suele hacerse a través de la política de privacidad del sitio.

Derecho de acceso

Los usuarios tienen derecho a solicitar sus registros personales almacenados.

Derecho al olvido

Los usuarios pueden solicitar a los propietarios del sitio que eliminen sus datos personales cuando no exista ninguna razón de peso para seguir procesándolos.

Derecho a restringir el tratamiento

Los usuarios pueden prohibir el tratamiento de sus datos personales. El sitio web puede seguir almacenando los datos, pero ya no procesarlos.

Derecho de rectificación

Los usuarios pueden corregir la información incompleta o inexacta en el sitio web.

Derecho a la portabilidad

Los usuarios pueden obtener y reutilizar sus datos personales.

Derecho de oposición

Los usuarios pueden oponerse a que su información personal se utilice para marketing, investigación y estadísticas.

Derechos relacionados con la toma automática de decisiones

Se debe pedir a los usuarios su consentimiento para utilizar sus perfiles para la toma de decisiones automatizada, como la elaboración de perfiles.

La mayoría de estos requisitos pueden probarse utilizando enfoques de pruebas funcionales. Por ejemplo, puedes añadir pruebas automatizadas a nivel micro y macro para afirmar que no hay opt-in implícito, verificar que los datos personales sólo se almacenan tras obtener el consentimiento del usuario y comprobar que la información personal no se almacena en los registros de la aplicación. Los conceptos y la mentalidad de las pruebas de seguridad tratados en [el Capítulo 7](#) encajarán perfectamente aquí.

PCI DSS y PSD2

Si tu aplicación se ocupa de pagos con tarjeta de crédito (como la mayoría de los sitios web de venta al por menor) o presta servicios de pago a la región de la UE, hay dos normativas que entrarán en juego:

Norma de Seguridad de Datos del Sector de las Tarjetas de Pago (PCI DSS)

La DSS de la PCI es una norma mundial definida por el [Consejo de Normas de Seguridad de la PCI](#) para proteger las transacciones con tarjeta en línea. Se aplica a cualquier entidad de que almacene, procese o transmita datos de titulares de tarjetas. Esto significa, en general, que se aplica a todos los sitios que toman datos de tarjetas de crédito, incluso a los sitios de donaciones. PCI DSS no es un requisito legal, sino una norma obligatoria que esperan los bancos y comerciantes para las transacciones con tarjeta. Existen multas por incumplimiento, según los respectivos contratos entre la empresa y el procesador

de pagos. Normalmente, las empresas pueden validar su cumplimiento mediante un cuestionario de autoevaluación.

PCI DSS proporciona **12 directrices** para hacer seguras las transacciones con tarjeta de crédito en una aplicación, como cifrar la transmisión, disponer de un cortafuegos, actualizar el software antivirus, etc. Por tanto, cuando realices las pruebas, debes pensar en escenarios para proteger los datos de la tarjeta, como enmascarar los datos de la tarjeta en la interfaz de usuario y en todas las ubicaciones de almacenamiento, aplicar restricciones al acceso a los datos de la tarjeta, evitar almacenar los datos de la tarjeta en registros, etc. Un ejercicio de modelado de amenazas, como el que se expone en **el Capítulo 7**, resultará útil en este caso.

Directiva de Servicios de Pago (PSD2)

La DSP fue la primera **directiva sobre servicios de pago** aplicada en la UE, cuyo objetivo era impedir los delitos de pago en línea . También pretendía aumentar la competencia en el sector de los pagos, para evitar que los bancos monopolizaran los servicios de pago. La PSD2 es una revisión de la norma PSD original. Su cumplimiento es obligatorio por ley en la región de la UE para todos los proveedores de servicios de pago. Si estás creando una aplicación que proporciona servicios de pago a clientes de la región de la UE, debes prestar atención a la normativa PSD2.

La PSD2 se centra principalmente en las funciones **de autenticación fuerte del cliente (SCA)** y en ampliar el alcance de la PSD2 dentro y fuera de la región de la UE; por ejemplo, exige el cumplimiento si incluso una parte de la transacción implica a un estado miembro de la UE. Para cumplir la PSD2, las empresas tienen la opción de elegir un proveedor de servicios de pago que ya la cumpla, como **Stripe** o **PayPal**, o incorporar las funciones SCA de los servicios de pago a sus aplicaciones. En términos sencillos, la SCA puede equiparse a la autenticación multifactor.

La Comisión Europea define la SCA como un mecanismo de autenticación que utiliza al menos dos de los tres elementos de verificación siguientes:

- El conocimiento único del usuario sobre algo, como una contraseña
- La posesión única de algo por parte del usuario, como una tarjeta de débito o crédito o un dispositivo móvil
- Los identificadores biométricos únicos del usuario, como su cara, voz o huellas dactilares

Estas funciones tienen que probarse a fondo para garantizar que cumplen la PSD2.

En resumen, el primer paso en las pruebas de cumplimiento es desarrollar una comprensión profunda de la legislación. A continuación, se pueden emplear adecuadamente los respectivos enfoques de prueba del CFR, tal como se han expuesto en la sección de estrategia (que abarca los cinco temas), para probarlos de forma holística. Una vez que la aplicación esté probada y lista, el equipo legal o una entidad autorizada se implicará para la certificación de cumplimiento. El ciclo de pruebas de conformidad sólo se considerará completado si la certificación es satisfactoria.

Y con eso, estás equipado para probar la larga lista de CFR que tu aplicación puede necesitar para prosperar y para permitir a tu equipo hacer entrega continua desplazando las pruebas de CFR a la izquierda.

Perspectivas: La Evolucionabilidad y la Prueba del Tiempo

Hemos hablado de cómo aprovechar la calidad comprobando los requisitos funcionales y multifuncionales de la aplicación. Sin

embargo, en este punto es importante comprender que los requisitos del software no están grabados en piedra al principio del proyecto. Como se ha establecido anteriormente, los requisitos del software cambian continuamente junto con las necesidades del mercado; ese cambio es inevitable. También es inevitable que los nuevos requisitos casi siempre amenacen la implementación existente si no se gestionan con prudencia. Por ejemplo, un miembro del equipo puede anular precipitadamente la encriptación para mejorar el rendimiento y comprometer así por completo la seguridad de la aplicación.

Ésta es la premisa central del libro *Building Evolutionary Architectures* de Neal Ford, Rebecca Parsons y Patrick Kua (O'Reilly), en el que los autores prescriben un nuevo CFR: *la evolvabilidad*, que es la capacidad del sistema para conservar las características de arquitectura existentes (por ejemplo, arquitectura en capas, métodos de persistencia de datos, cifrado en reposo y en tránsito) que facilitan un conjunto determinado de requisitos funcionales y multifuncionales, al tiempo que incorpora nuevos cambios. Para lograr la evolucionabilidad, recomiendan la implantación de guardarraíles adecuados para las características arquitectónicas esenciales que puedan ser no negociables, que proporcionarán a los equipos una retroalimentación instantánea cada vez que se desvíen. Estos guardarraíles pueden adoptar la forma de pruebas automatizadas en torno a cada uno de los requisitos funcionales y multifuncionales (como pruebas de rendimiento, análisis de seguridad, resultados de auditorías de accesibilidad, pruebas de arquitectura y pruebas funcionales de micro y macronivel), así como métricas de cobertura de código, métricas de analizadores estáticos de código, etc. Este conjunto de pruebas y métricas, denominadas colectivamente *funciones de adecuación*, guiarán a los equipos en la realización de cambios incrementales sin comprometer la implementación existente, y en el proceso crearán una arquitectura evolutiva.

Para consolidar los puntos de vista presentados aquí, todos los métodos y herramientas de pruebas funcionales y multifuncionales que hemos ido aprendiendo a lo largo del libro, incluido lo que se recoge en este capítulo, ayudan colectivamente a construir una arquitectura evolutiva que resista la prueba del tiempo, iademás de imbuir de alta calidad en la actualidad!

Puntos clave

Éstos son los puntos clave de este capítulo:

- Los requisitos interfuncionales, comúnmente llamados requisitos no funcionales, son tan esenciales como los funcionales para el éxito de la aplicación. Los requisitos funcionales y los interfuncionales juntos hacen de la aplicación un producto de alta calidad.
- Los CFR definen principalmente las cualidades de ejecución y evolución de la aplicación.
- Las CFR se aplican a una amplia gama de funciones de la aplicación y, por tanto, deben desarrollarse y probarse como parte de cada historia de usuario. Tener una lista de comprobación de las CFR como parte de cada historia de usuario puede ser una forma de garantizar la realización de las pruebas de las CFR.
- El modelo FURPS abstrae temas de todos los requisitos del software. Se puede ver que los CFR se manifiestan a lo largo de esos temas.
- El capítulo proporciona una estrategia de pruebas para cada uno de los cinco temas del modelo FURPS, que puede utilizarse para elaborar una estrategia de pruebas de CFR específica del proyecto. Esta estrategia de pruebas debe prestar atención a

cada CFR individualmente, en función de las necesidades del proyecto.

- Desplaza tus pruebas de CFR hacia la izquierda automatizando estas pruebas e integrándolas con CI.
- La Ingeniería del Caos es un método de experimentación para desvelar los fallos inherentes a la aplicación que podrían hacerla poco fiable. Los experimentos deben realizarse iterativamente en equipo.
- Herramientas como ArchUnit y JDepend ayudan a afirmar las características arquitectónicas de la aplicación para mantener algunas de las cualidades del código evolutivo.
- Las pruebas de infraestructura son un área emergente en las pruebas. Es necesaria en los casos en que necesitas escalar la aplicación rápidamente. Las herramientas automatizadas de pruebas de infraestructura aún están evolucionando y pueden suponer costes adicionales en términos de ampliación de conocimientos e implementaciones reales de pruebas de infraestructura.
- El GDPR y las WCAG 2.0 son normativas de aplicación habitual en las aplicaciones web. Para comprobar su cumplimiento, se nos puede exigir que conozcamos a fondo estas normativas, esencialmente aprendiendo de un equipo jurídico.
- Las pruebas funcionales y multifuncionales se convierten en funciones de adecuación y no sólo ayudan a los equipos a entregar hoy software de alta calidad, sino que también ayudan a crear arquitecturas evolutivas que superarán la prueba del tiempo.

¹ Desarrollado en Hewlett-Packard y descrito originalmente por Robert Grady en su libro *Practical Software Metrics for Project Management and Process*

Improvement (Prentice-Hall).

Capítulo 11. Pruebas móviles

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

iImagina un día sin tu móvil!

Desde la aparición de los smartphones, los dispositivos móviles se han convertido en un miembro más para muchos de nosotros. Han aportado sofisticación a nuestras vidas al ofrecernos servicios cotidianos con sólo tocarlos y deslizarlos. No se me ocurre ningún otro objeto que sirva para tantas cosas como un smartphone. Los utilizamos para comprar alimentos, ropa, electrodomésticos y otras necesidades básicas. Los utilizamos para leer libros, ver películas y jugar a juegos para entretenernos. Los teléfonos inteligentes nos facilitan las operaciones bancarias, el pago de facturas y la organización de nuestros calendarios. Y lo que es más, nos aportan una sensación de seguridad psicológica, pues sabemos que la ayuda está a una llamada de distancia.

Con todas estas ventajas y usos, no es de extrañar que haya **6.600 millones de usuarios de teléfonos inteligentes** en todo el mundo. Lo que puede sorprender, sin embargo, es que haya más de **8.000 millones de suscripciones móviles**, ¡mucho más que personas en el planeta! La escala de adopción es asombrosa, pero los estudios también correlacionan estas cifras con un uso extenso. Por ejemplo, un estudio reciente descubrió que los estadounidenses, de media, consultan sus teléfonos **344 veces al día**, o cada 4 minutos. Del mismo modo, un usuario medio de smartphone en todo el mundo accede a **10 aplicaciones al día y a 30 aplicaciones al mes**. Y este amplio uso de los teléfonos inteligentes no se limita a ningún grupo

de edad específico: Se calcula que los jóvenes de 18 a 24 años dedican 93,5 horas al mes a los teléfonos inteligentes, frente a las 62,7 horas de los de 45 a 54 años y las 42,1 horas de los mayores de 65 años (o aproximadamente 3 horas, 2 horas y 1,5 horas al día, respectivamente). Tal es el impacto de los teléfonos inteligentes en todos nosotros.

Con todo este uso, no debería sorprender que en 2021 hubiera **5,7 millones de aplicaciones** disponibles en las principales tiendas de aplicaciones, Google Play y App Store de Apple, y que el número de aplicaciones móviles siga creciendo en los próximos años, ya que están demostrando ser lucrativas para las empresas. Sólo en 2020, las aplicaciones móviles generaron unos ingresos de más de **318.000 millones de dólares** en todo el mundo, y se prevé que aumenten a más de 613.000 millones de dólares en 2025.

¿Por qué son importantes todas estas cifras? Porque nosotros, como desarrolladores y probadores de software, vamos a desarrollar y probar estas aplicaciones móviles, y es una clara llamada a la acción para perfeccionar nuestras habilidades móviles. El objetivo de este capítulo es ofrecer una visión de la mentalidad y las herramientas de las pruebas móviles. Si te preguntas en qué se diferencian las pruebas móviles de las pruebas web, este capítulo responderá a esa pregunta. Presenta el panorama móvil general y las particularidades de las pruebas móviles frente a las pruebas web. Aprenderás una estrategia para probar completamente la capa móvil, incluidas las pruebas funcionales automatizadas, así como las pruebas de rendimiento, seguridad, accesibilidad, visuales y CFR. Además, este capítulo contiene ejercicios guiados para que te pongas al día y estés preparado para los proyectos móviles.

Bloques de construcción

Para empezar, echemos un vistazo al panorama móvil general, a los retos que nos plantea y a los aspectos específicos que requieren

nuestra atención al probar aplicaciones móviles.

Introducción al paisaje móvil

Como se visualiza en [la Figura 11-1](#), hay tres áreas principales que debe tener en cuenta al abordar el panorama móvil: los dispositivos, las propias aplicaciones y la red. Echemos un vistazo a cada una de ellas.

The Mobile Landscape

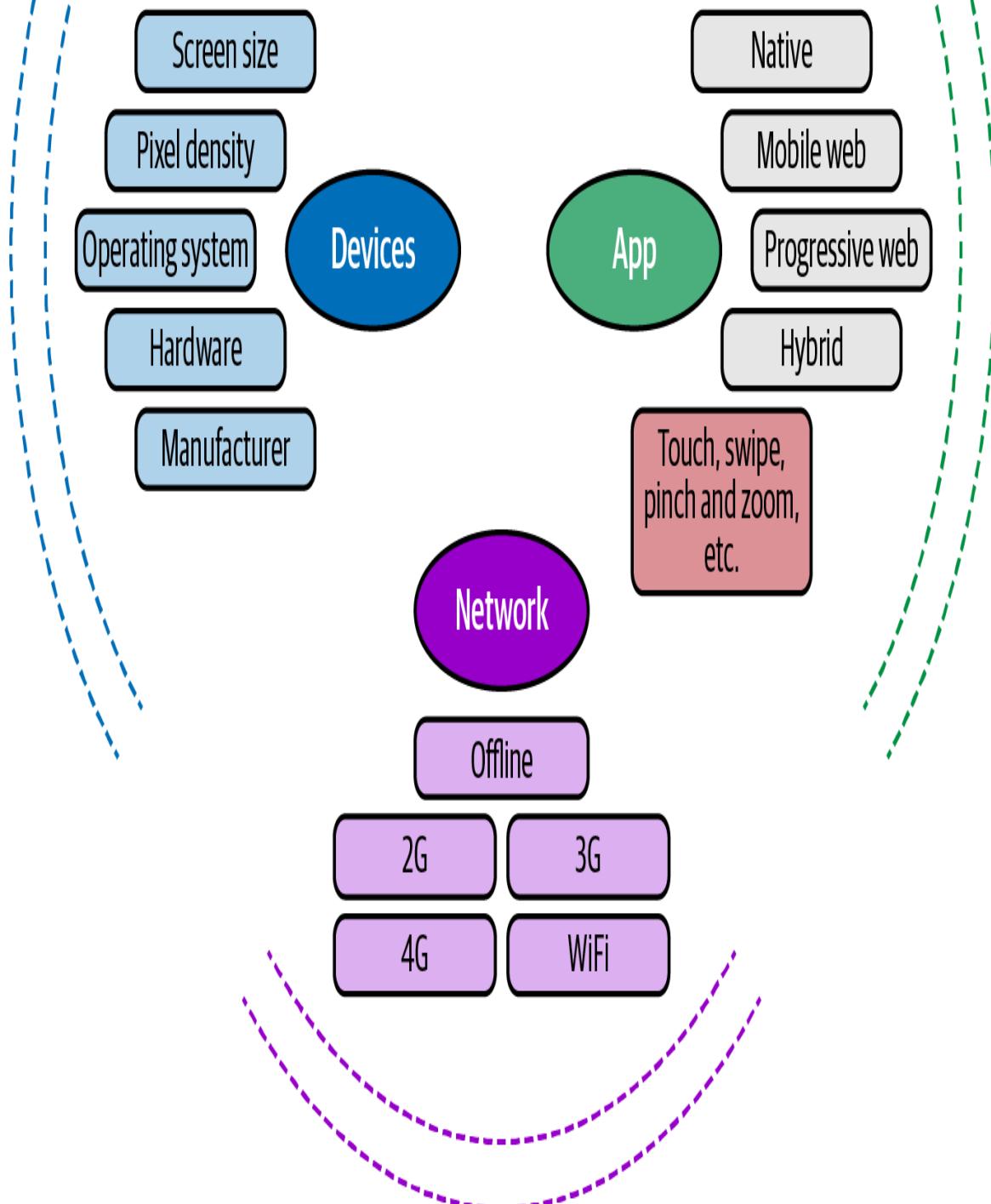


Figura 11-1. El paisaje móvil

Dispositivos

Como parte de su evolución, los dispositivos móviles han llegado a variar en varias dimensiones entre sí. Es fundamental comprender estas dimensiones para decidir qué dispositivos utilizar para las pruebas. Como norma general, debes tratar de proporcionar cobertura de pruebas para al menos el 85% de los dispositivos objetivo. Aquí tienes una lista de las distintas dimensiones de los dispositivos que debes tener en cuenta al decidir tu estrategia de pruebas:

Tamaño de la pantalla

Los dispositivos móviles incluyen tanto las tabletas como los smartphones. Hay más de **mil millones de usuarios de tabletas en todo el mundo**, iy no es una cifra despreciable! Con todos los diferentes factores de forma y modelos de dispositivos, no debería sorprender que los tamaños de pantalla de **las tabletas** y **los teléfonos inteligentes** varíen significativamente. Es más, el tamaño de la pantalla difiere dentro del mismo dispositivo dependiendo de su orientación (es decir, si se mantiene en modo de visualización horizontal o vertical), y los teléfonos modernos permiten ver varias aplicaciones en pantalla dividida, subdividiendo aún más el espacio de pantalla disponible.

El tamaño de la pantalla tiene un gran impacto en la experiencia general del usuario final, lo que hace que diseñar, desarrollar y probar para distintos tamaños de pantalla sea fundamental en el panorama móvil. Por ejemplo, en las pantallas más pequeñas el usuario puede tener que desplazarse para ver toda la página, mientras que en las pantallas más grandes puede haber mucho espacio vacío: ninguna de las dos cosas puede ser una experiencia de usuario acogedora.

Densidad de píxeles

Un píxel es una pequeña área cuadrada en la pantalla que contiene una parte de la información que se muestra, y la densidad de píxeles es el número de píxeles distintos que caben en una pulgada cuadrada de la pantalla. Cuanto mayor sea la densidad, mejor será la experiencia de visualización. Los dispositivos móviles no sólo difieren en el tamaño de sus pantallas, sino que dispositivos con el mismo tamaño de pantalla pueden tener **diferentes densidades de píxeles**. En función de su densidad de píxeles, los dispositivos se clasifican en densidad baja, media, alta, extra alta, extra-extra alta o extra-extra-extra alta. Tu¹ Esta dimensión de los dispositivos móviles afecta especialmente a la representación de las imágenes, ya que éstas se redimensionan automáticamente en determinados dispositivos para adaptarse al tamaño de la pantalla, lo que provoca borrosidad o distorsión. Por eso, las imágenes deben diseñarse y programarse específicamente para las respectivas densidades de píxeles, y esto debe incluirse en las pruebas.

NOTA

La *resolución* de una pantalla es una indicación en del número de píxeles que puede mostrar horizontal y verticalmente. Por ejemplo, una pantalla con una resolución de 1.024 x 768 puede mostrar 1.024 píxeles en horizontal y 768 en vertical cuando el dispositivo está en orientación horizontal.

Sistema operativo

Al igual que el mundo de los ordenadores de sobremesa tiene Windows, macOS, Linux y otros sistemas operativos, el mundo de los móviles tiene Android, iOS, Windows Mobile, Symbian, KaiOS, etc. Como puedes adivinar, Android e iOS ocupan los dos primeros puestos, representando **el ~99% del uso de sistemas operativos móviles** en todo el mundo. Pero la cosa no acaba ahí.

Hay muchas versiones de cada SO que siguen teniendo soporte oficial y que muchos utilizan con teléfonos antiguos. Esto se conoce como *fragmentación*. Por ejemplo, en 2020, **Android 6.0** seguía siendo la segunda versión de Android más utilizada, a pesar de haber sido lanzada en 2015, mientras que Android 9.0 ocupaba el primer lugar. Por tanto, el alcance de tus pruebas debe incluir también diferentes versiones del SO, ya que algunas versiones pueden no admitir determinadas funciones o manejarlas de forma diferente.

Hardware

Las configuraciones de hardware de los dispositivos móviles - RAM, CPU, capacidad de la batería, capacidad de almacenamiento local, etc.- son otra dimensión que varía de un modelo a otro. El hardware influye en el rendimiento de la aplicación en términos de procesamiento paralelo, rapidez en la renderización de la aplicación y la experiencia general del usuario de la aplicación. Especialmente cuando tu aplicación depende de capacidades integradas en el dispositivo, como el GPS, la cámara, el micrófono, la pantalla táctil y otros sensores de hardware, la experiencia del usuario final variará en función de las capacidades inherentes del hardware.

Esta dimensión es importante, ya que puede afectar incluso a la funcionalidad principal de la aplicación. Por ejemplo, una aplicación móvil diseñada para recopilar información sobre supervivientes durante catástrofes como tsunamis o ciclones no puede depender de que los voluntarios posean una cámara de gama alta y debe tener cuidado de no consumir demasiada batería. Dependiendo del uso previsto, puede que tu aplicación tenga que diseñarse, desarrollarse y probarse teniendo en cuenta las estrictas condiciones del hardware.

Fabricante del aparato

Hay varios fabricantes de dispositivos en el mercado actualmente, como Oppo, Samsung, Xiaomi, LG, Motorola, Google, Apple, etc. Algunos de estos fabricantes tienen sus propias versiones personalizadas de Android, como Cyanogen OS, Oxygen OS e Hydrogen OS. Además, cada fabricante de dispositivos sigue su propio diseño de hardware, como proporcionar un botón central de inicio o un botón de retroceso. Hay que tener en cuenta estos matices al desarrollar y probar aplicaciones móviles.

Evidentemente, como puedes ver, las consideraciones relacionadas con los propios dispositivos plantean un montón de retos que los equipos de software deben gestionar . A continuación, veamos el problema desde el punto de vista de las aplicaciones.

Aplicación

Una especialidad clave de las aplicaciones móviles es el variado conjunto de interacciones que permiten. Además del clic y tecleo estándar de que admiten las aplicaciones web, en una aplicación móvil puedes deslizar, tocar, pulsar prolongadamente, acercar y alejar, pellizcar y acercar, pulsar y arrastrar, girar, iy mucho más! Estos gestos e interacciones son una gran parte de lo que hace que el uso del móvil sea más atractivo y personalizado. Un subconjunto de estas interacciones puede hacerse comúnmente disponible en toda la aplicación, como deslizar de izquierda a derecha en cualquier página para mostrar el menú, o deslizar hacia arriba desde la parte inferior de la pantalla para que aparezcan características adicionales de la funcionalidad actual. Estas interacciones comunes se convierten en requisitos interfuncionales de toda la aplicación, que deben diseñarse, construirse y probarse como parte de cada historia de usuario. Sin embargo, la posibilidad de mejorar las interacciones se ve limitada por el tipo de aplicación. Aunque como usuarios finales puede que nunca hayamos considerado estas distinciones entre aplicaciones móviles, como equipos de software necesitamos

conocer los diferentes tipos de aplicaciones para poder abordar las pruebas en consecuencia. Actualmente, los siguientes cuatro tipos de aplicaciones móviles se adoptan con frecuencia:

Nativo

Las aplicaciones nativas se desarrollan principalmente para trabajar en un único sistema operativo móvil, como Android o iOS. Las ventajas de elegir desarrollar aplicaciones nativas son que pueden proporcionar un rendimiento excelente, ofrecen acceso al hardware del dispositivo y a todas las funciones y API del sistema operativo (incluidos los gestos), pueden funcionar sin conexión y tienen un aspecto coherente y armonioso. Las aplicaciones nativas de Android suelen estar escritas en Java o Kotlin, y las de iOS se desarrollan con Objective C o Swift. Se distribuyen a través de Google Play y Apple App Store, respectivamente. Cada una de estas plataformas de distribución tiene directrices de cumplimiento y procesos de aprobación tras la presentación de la aplicación, por lo que puede haber un retraso antes de que la aplicación se ponga a disposición del público. Aunque esto puede no ser problemático, incluso las correcciones urgentes de errores pasan por el proceso de aprobación, lo que retrasa su lanzamiento! Otro inconveniente importante es el coste de desarrollo, ya que hay que desarrollar una aplicación nativa distinta para cada uno de los sistemas operativos de destino.

Web móvil

Las aplicaciones web para móviles son sitios web a los que se accede mediante navegadores web para móviles. Tienen las ventajas de ser independientes del sistema operativo, de no requerir procedimientos de instalación y de no necesitar espacio de almacenamiento local para su instalación. Tampoco dependen de las tiendas de aplicaciones para su aprobación o distribución. Además, estas aplicaciones pueden desarrollarse con las

tecnologías habituales de desarrollo web (como HTML5 y CSS), por lo que no es necesario aprender lenguajes específicos de los SO móviles. Las desventajas son que no tienen acceso a funciones del SO como la agenda telefónica, la cámara, etc., y no pueden funcionar sin conexión. Como resultado, la experiencia del usuario es muy limitada.

Híbrido

Las aplicaciones híbridas aportan lo mejor de ambos mundos nativo y web. Una aplicación híbrida se desarrolla utilizando tecnologías de desarrollo web estándar, como HTML, JavaScript o CSS, y luego se envuelve en un contenedor nativo que proporciona acceso a las API relacionadas con el sistema operativo. React Native, Ionic, Apache Cordova y Flutter son algunos de los marcos de desarrollo de aplicaciones híbridas más populares que permiten esto, permitiendo incluso que la misma aplicación funcione en varios sistemas operativos. Las aplicaciones híbridas deben enviarse a una tienda de aplicaciones para su distribución, pero los elementos web de la aplicación pueden alojarse en un servidor y obtenerse a través de la red. Como resultado, la actualización de estas partes de la aplicación puede hacerse fácilmente sin pasar por el proceso de aprobación de la tienda. Sin embargo, esto limitará la visualización offline de la aplicación, por lo que los equipos suelen almacenar un conjunto mínimo de contenidos seleccionados localmente en el dispositivo para permitir la visualización offline. En general, el enfoque híbrido facilita el desarrollo y reduce los costes de desarrollo. La contrapartida, sin embargo, es el rendimiento, donde las aplicaciones nativas son mejores. Además, como estas aplicaciones suelen crearse para funcionar en distintos sistemas operativos, puede producirse el efecto secundario no deseado de alienar a algunos usuarios finales, acostumbrados a utilizar sus aplicaciones de una determinada manera en su sistema operativo preferido.

Web progresiva

Las aplicaciones web progresivas (PWA) son versiones avanzadas de las aplicaciones web móviles. Los usuarios pueden instalarlas en sus dispositivos a través de una URL, y ocupan muy poco espacio de almacenamiento. Aunque son aplicaciones web, pueden proporcionar notificaciones push, pueden funcionar sin conexión y tienen acceso a funciones del sistema operativo, lo que hace que la experiencia sea similar a la de una aplicación nativa. El rendimiento de las PWA también está a la par con el de las aplicaciones nativas, y como son aplicaciones web, pueden funcionar en todos los sistemas operativos y navegadores. Y lo que es más, todas estas capacidades se pueden conseguir con costes de desarrollo más baratos en comparación con las aplicaciones nativas e híbridas. Dadas estas ventajas, las PWA se han convertido en la opción preferida de las empresas hoy en día. Twitter **sustituyó su aplicación web móvil por una aplicación web progresiva en 2017**, y ha visto una disminución del 20% en la tasa de rebote, un aumento del 75% en los tweets enviados y un aumento del 65% en las páginas vistas por sesión!

Como habrás deducido, el tipo de aplicación que elijas crear determinará el alcance de las pruebas en términos de comportamiento offline frente a online, compatibilidad con funciones específicas del sistema operativo, comportamiento de actualización de la aplicación, interacciones, etc. Con esto, pasemos a la última consideración importante: la red.

Red

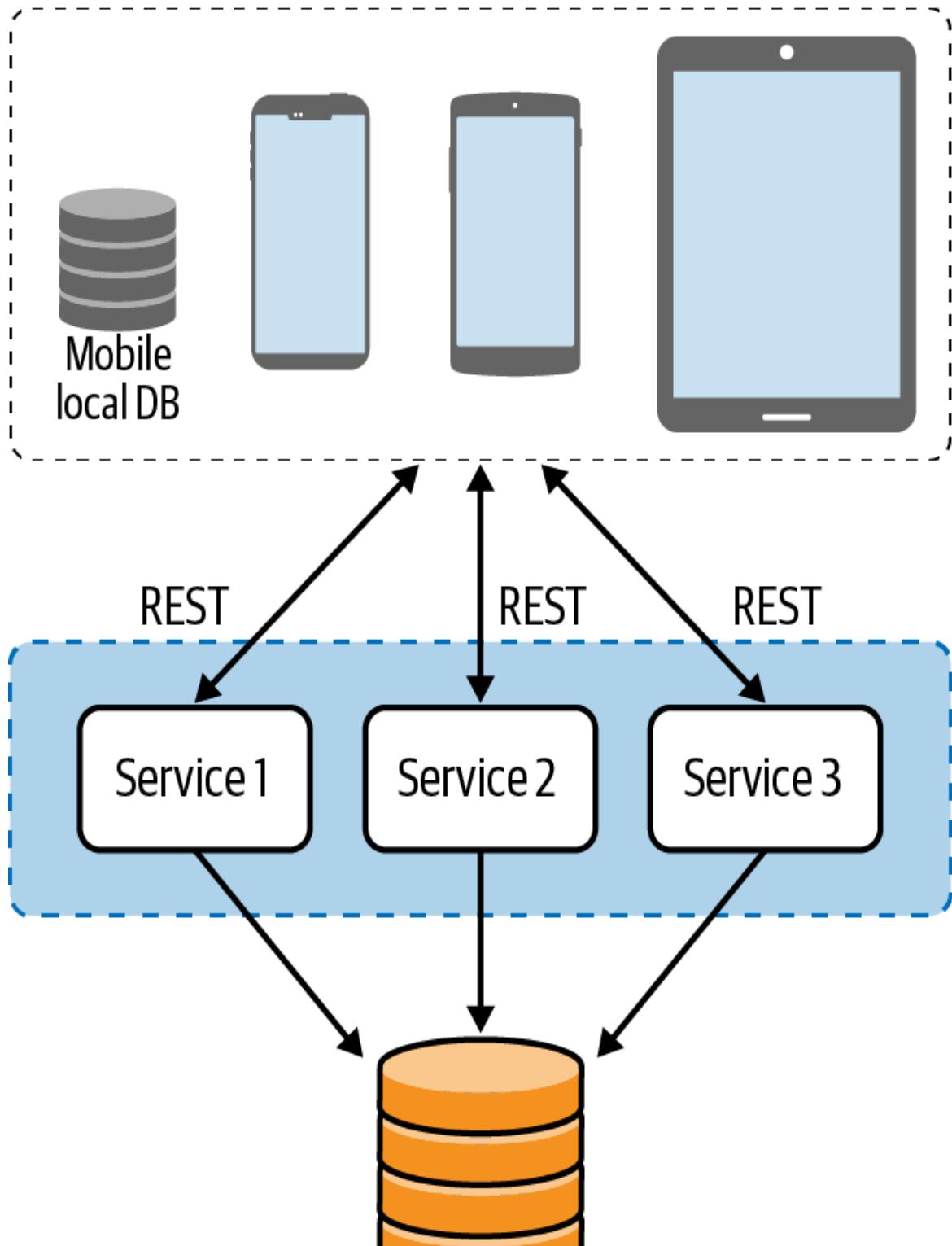
Las personas de todo el planeta no tienen el mismo acceso a las redes de alta velocidad. Aunque el escaso ancho de banda suele ser un problema en lugares remotos, incluso dentro de las zonas urbanas la conectividad de red no es constante. Por eso, cuando tu aplicación depende de la conectividad de red, tienes que admitir

distintos tipos de red móvil (como 2G, 3G y 4G), además de WiFi y estar completamente desconectado. Tendrás que probar cómo maneja tu app escenarios como los tiempos de espera de la red, la visualización de errores en las oscilaciones de la red (por ejemplo, al cambiar entre 4G y 3G), la actividad sin conexión, el rendimiento del lanzamiento con distintos tipos de red, etc. Puede que incluso tengas que diseñar la aplicación teniendo en cuenta estas limitaciones desde el principio. Por ejemplo, Facebook lanzó la [aplicación Facebook Lite](#) principalmente para resolver los problemas de ancho de banda de la red. Puede funcionar con 2G y, en general, ofrece un funcionamiento sin problemas en conexiones de red inestables.

Mirar el panorama móvil a través de estas tres lentes debería haberte dado una idea de las complejidades adicionales que hay que abordar en las pruebas móviles. Para que comprendas mejor el ámbito de las pruebas móviles, a continuación profundizaremos un poco más en la arquitectura de una aplicación móvil.

Arquitectura de aplicaciones móviles

Ya hablamos de la arquitectura de una aplicación web típica en [el Capítulo 2](#). Como recordarás, tiene una interfaz de usuario web que recibe las peticiones del usuario y servicios que procesan las peticiones colaborando con la capa de base de datos. La arquitectura de una aplicación móvil no es muy diferente. Como se muestra en [la Figura 11-2](#), la UI móvil sustituye a la capa de UI web, y el resto suele ser más o menos igual.



Centralized or independent
database per service

Figura 11-2. Arquitectura de una aplicación móvil

Observa el componente adicional, la BD local, en la capa móvil. Aquí es donde las aplicaciones nativas e híbridas almacenan los datos seleccionados, como el nombre de usuario, la foto del perfil, el último contenido obtenido, etc., para soportar el comportamiento offline y acelerar el tiempo de renderización de la aplicación. El resto del flujo es similar al de una aplicación web, donde la aplicación móvil llama a los servicios a través de la red y completa las peticiones de los usuarios. Por tanto, el enfoque de las pruebas de los servicios y la capa de la base de datos sigue siendo el mismo: haces pruebas a nivel micro y macro y escribes pruebas unitarias, de integración y de API. También probarás el rendimiento de los servicios, la seguridad, el cumplimiento legal y otros CFR. Además, debes realizar pruebas específicas de la interfaz de usuario móvil, prestando atención a sus complejidades inherentes. Estas áreas específicas de enfoque en las pruebas de la IU móvil son las que trataremos a continuación.

NOTA

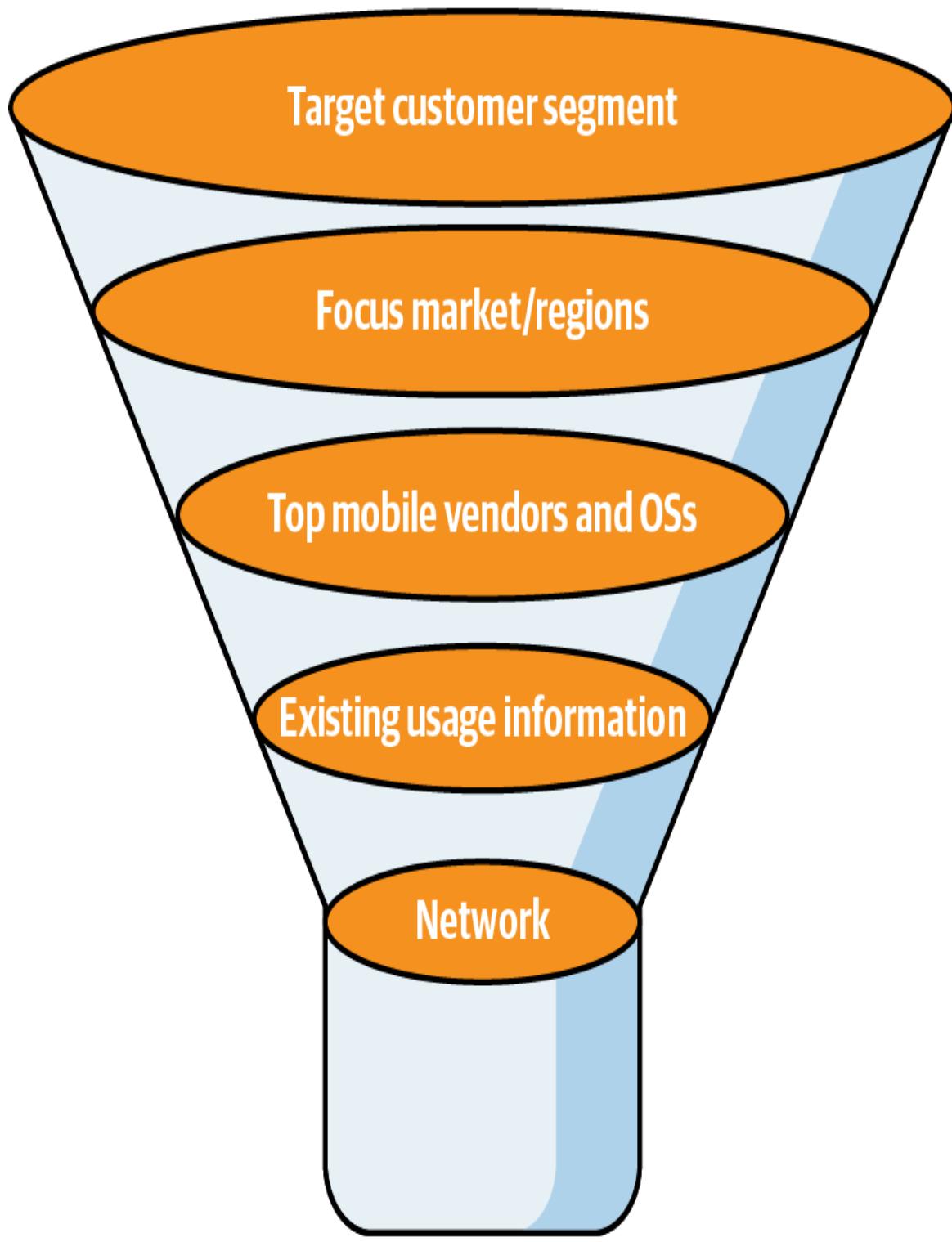
Para conocer mejor los componentes interiores de una capa de interfaz de usuario móvil, explora la [guía de arquitectura Android](#) de Google.

Estrategia de pruebas móviles

Lo primero que debes tener en cuenta en tu estrategia de pruebas móviles es la lista de dispositivos con los que probarás . El objetivo es proporcionar una cobertura de pruebas para el 85% del segmento de clientes objetivo para cada historia de usuario, ya que probar en todas las permutaciones de los dispositivos (tamaños de pantalla, sistemas operativos, hardware, etc.), como se ha mencionado antes, es poco probable que sea una opción viable. Por ejemplo, probar

todas las versiones de Android y todos los dispositivos de todos los fabricantes llevaría mucho tiempo y sería caro. Especialmente en un proceso de desarrollo iterativo como Agile o Scrum, tener que probar en decenas de dispositivos para validar una historia de usuario afectaría negativamente al ritmo de entrega de tu equipo. Por tanto, reducir la lista de dispositivos de prueba es crucial. Aquí tienes una serie de preguntas a las que debes encontrar respuesta para filtrar los dispositivos y cumplir el objetivo de una cobertura del 85%, como se ilustra en la **Figura 11-3**:

- ¿Cuáles son los segmentos de clientes objetivo de la empresa? Por ejemplo, un negocio de ropa de gama alta puede dirigirse a la clase acomodada, y por tanto pueden dejarse de lado los teléfonos de gama baja.
- ¿A qué mercados/países concretos quiere expandirse la empresa, y cuáles son los principales sistemas operativos y proveedores en esos mercados? Por ejemplo, la empresa de ropa puede querer centrarse en las ciudades europeas, y Samsung y Apple son los **dos principales proveedores en Europa**. Esto reduce un poco más la lista de dispositivos, ya que puedes suponer que hay muchas posibilidades de que los usuarios acomodados utilicen dispositivos insignia de cada una de estas marcas.
- Si la empresa ya tiene presencia en Internet, ¿cómo es el uso específico de cada dispositivo? Por ejemplo, su aplicación web existente podría ser más accesible desde iPhones y tabletas Samsung.
- ¿Cuál es el rango de ancho de banda de red disponible en los mercados objetivo? Por ejemplo, se calcula que la velocidad media de la red móvil en Europa es de unos **54 Mbps**, lo que exige la inclusión de dispositivos compatibles con 4G. Los criterios de red cobran especial importancia cuando se trata de teléfonos de gama baja.



Goal: 85% testing coverage

Figura 11-3. Estrategia de filtrado de dispositivos móviles

Una vez que obtengas las respuestas a estas preguntas de la empresa o de un representante de producto, puedes seleccionar tres o cuatro teléfonos que tengan las características adecuadas, y quizás algunos dispositivos más que te gusten para comprobarlos durante las habituales batidas de bichos del equipo.

NOTA

Seleccionar los dispositivos para las pruebas es una actividad que debe realizarse al principio del propio proyecto. Una vez elegidos los dispositivos, debes hacer un análisis de costes sobre si comprarlos o suscribirte a un proveedor de dispositivos alojados en la nube, como AWS Device Farm, Firebase Test Lab, Xamarin Test Cloud, Perfecto o Sauce Labs. Estos proveedores te permiten ejecutar pruebas automatizadas en sus dispositivos reales alojados, pero en estos casos las interacciones pueden ser más lentas.

La Figura 11-4 muestra una estrategia de pruebas para la capa de interfaz de usuario móvil. Como puedes ver, los métodos de prueba son similares a los que hemos tratado a lo largo del libro hasta este punto. A continuación, hablaremos de cómo los distintos métodos ayudan a navegar por las complejidades que plantea el panorama móvil.

The mobile testing strategy

Exploratory testing

Functional automated testing

Visual testing

Continuous testing

Data and performance testing

Other CFR testing

Exploratory testing

Functional automated testing

Data testing

Continuous testing

Accessibility, security, performance testing

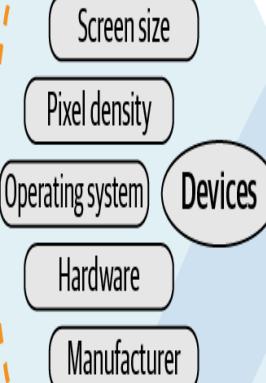
Other CFR testing

Data testing

Performance testing

Exploratory testing

Other CFR testing



Network

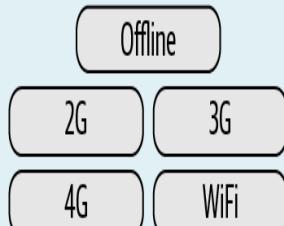


Figura 11-4. Una estrategia de pruebas móviles

Pruebas exploratorias manuales

Las pruebas exploratorias adquieren aún más importancia en el panorama móvil debido a la miríada de combinaciones de dispositivos, aplicaciones y redes para casos de prueba. Las técnicas y la estrategia de pruebas exploratorias elaboradas en el Capítulo 2 también serán de ayuda aquí, y aplicarlas a través de estas tres lentes permitirá una inspección exhaustiva del comportamiento de la aplicación. La opción de Chrome DevTools para ver sitios web en cualquier resolución, como vimos en el Capítulo 8, puede utilizarse para facilitar la exploración de aplicaciones web móviles. Para explorar otros tipos de aplicaciones, puedes comprar los dispositivos que hayas decidido utilizar para las pruebas o utilizar emuladores/simuladores.

Los emuladores y simuladores son programas que crean un entorno de dispositivo virtual en un ordenador. Por ejemplo, Android Studio proporciona emuladores de Android que imitan las configuraciones exactas de hardware y software de dispositivos reales como el Google Nexus 4, el Samsung Galaxy 5, el Moto G y muchos otros. Del mismo modo, iOS proporciona simuladores para iPhones y iPads. Veremos la configuración de un emulador de Android en la sección de ejercicios más adelante en este capítulo. Aunque pueden ser suficientes para hacer pruebas de cordura de la aplicación, los emuladores y simuladores tienen limitaciones a la hora de imitar algunas características del hardware, como ciertos gestos táctiles, integraciones de sensores, etc. Personalmente, considero que las pruebas con estas herramientas son insuficientes para considerar que una versión de una aplicación está lista para su lanzamiento, y el sector está de acuerdo: tanto Apple como Google recomiendan realizar pruebas en dispositivos reales antes del lanzamiento. Pueden resultarte útiles durante el desarrollo de pruebas para comprobar rápidamente el código, pero en general debes utilizar emuladores y

simuladores sólo en ausencia de dispositivos reales o para una rápida comprobación de cordura .

CONSEJO

En un entorno de pruebas ágil, puedes desplazar fácilmente las pruebas de dispositivos hacia la izquierda. Por ejemplo, el desarrollador puede utilizar la resolución más desafiante (LDPI o MDPI) durante el desarrollo. Durante las pruebas dev-box, el analista de negocio, el ingeniero de control de calidad y el desarrollador pueden probar cada uno una versión del dispositivo. Y con fines de regresión, las pruebas automatizadas pueden ejecutarse en un conjunto de dispositivos elegidos en CI.

Pruebas funcionales automatizadas

Las pruebas del comportamiento funcional de la aplicación y las interacciones de pueden automatizarse utilizando pruebas funcionales de extremo a extremo, unitarias y basadas en la interfaz de usuario. Al igual que las pruebas funcionales de las aplicaciones web, deben integrarse en el proceso CI para obtener información continua. Las pruebas de extremo a extremo también pueden utilizarse para garantizar que las funciones de la aplicación funcionan de forma fiable en un conjunto de dispositivos de destino, ejecutándolas como pruebas de humo o regresiones nocturnas. Appium y Espresso son un par de herramientas muy utilizadas para escribir pruebas de extremo a extremo basadas en la interfaz de usuario para Android, mientras que Appium y XCUI Test se utilizan mucho cuando se desarrolla para iOS. Espresso y XCUI Test sólo pueden utilizarse para aplicaciones nativas, pero Appium puede utilizarse para automatizar los tres tipos de aplicaciones (nativas, híbridas y web móviles). Más adelante en este capítulo aprenderás a utilizar esta herramienta para crear pruebas de extremo a extremo basadas en la interfaz de usuario.

Prueba de datos

Cuando se trata de aplicaciones móviles, los datos pueden almacenarse en varias capas con distintos fines. Como se muestra anteriormente en la Figura 11-2, existe una BD móvil local, una base de datos común y el almacenamiento local del dispositivo para almacenar y recuperar datos. Es esencial comprender el flujo de datos hacia todos estos almacenamientos e incluirlos en tus pruebas, como se explica en el Capítulo 5. Por poner algunos ejemplos de flujo de datos en aplicaciones móviles, una aplicación de red social como Facebook podría almacenar las publicaciones más recientes en la BD local del móvil, de modo que cuando las condiciones de la red sean inestables, pueda utilizarse para renderizar rápidamente la aplicación. En estos casos de uso, puede que tengas que pensar en la experiencia del usuario con respecto a mostrar información obsoleta, el volumen de datos que puede almacenarse en la BD local, mantener la BD local sincronizada, etc. Además, los usuarios pueden acceder a la aplicación desde varios dispositivos, como un teléfono, una tableta y un navegador web de escritorio. Así que hay que probar la sincronización de datos con las BD de todos los dispositivos.

Cuando se trata de la base de datos común, es necesario que haya un seguimiento de las distintas transacciones realizadas desde distintos dispositivos y un mecanismo para actualizar los datos apropiados sin conflictos. Por ejemplo, un usuario podría guardar eventos del calendario desde distintos dispositivos móviles y sincronizarlos cuando la red esté disponible. Esto debe rastrearse y actualizarse en la base de datos común. Tales escenarios pueden automatizarse también como pruebas funcionales a nivel micro y macro. En resumen, la sincronización bidireccional de datos entre la base de datos común y la base de datos móvil local a través de los dispositivos, limitada por las condiciones de la red, puede ser un aspecto esencial a probar cuando se trata de pruebas de datos en aplicaciones móviles.

Además, cuando haya funcionalidades que requieran almacenar archivos en el almacenamiento local del dispositivo y recuperarlos de él, también habrá que probarlas. Tendrás que tener en cuenta las condiciones límite para el almacenamiento interno y externo del dispositivo (cuando el almacenamiento está lleno o no está disponible) y las limitaciones del SO para procesar distintos formatos de archivo. En general, cuando pienses en las pruebas de datos, ¡es una buena idea ver tu aplicación móvil a través de las tres lentes!

Pruebas visuales

La prueba visual quedará cubierta cuando realices manualmente la prueba de dispositivos . Como ya se ha mencionado, también puedes desplazar tus pruebas de dispositivos hacia la izquierda. También puedes automatizar las pruebas visuales para distintos tamaños de pantalla (que habrás elegido durante la selección de dispositivos) utilizando Appium y [Applitools Eyes](#) con fines de regresión. Applitools Eyes, como se explica en [el Capítulo 6](#), es un servicio de pago que emplea IA para automatizar las pruebas visuales de las aplicaciones móviles, mientras que Appium es de código abierto. Más adelante en este capítulo se incluye un ejercicio para automatizar las pruebas visuales utilizando Appium. Puedes elegir entre las dos herramientas en función de los factores mencionados en [el Capítulo 6](#).

Pruebas de seguridad

En [el capítulo 7](#), hablamos de la mentalidad de las pruebas de seguridad de y de lo esencial que hay que buscar desde el punto de vista de la seguridad al probar una funcionalidad. Esto también puede aplicarse a las pruebas móviles. Por ejemplo, las consideraciones durante las pruebas deben incluir la encriptación y el almacenamiento seguro de los datos sensibles de los usuarios,

mecanismos de autenticación fuertes, permisos adecuados para acceder a otras aplicaciones del teléfono, etc.

También hablamos de las herramientas de pruebas de seguridad en el Capítulo 7, que escanean automáticamente el código estático de la aplicación en busca de vulnerabilidades de seguridad e inyectan ataques conocidos para detectar vulnerabilidades en la aplicación. Como se ha mencionado, esas herramientas también funcionan en la capa de servicios y pueden utilizarse en esa capa de las aplicaciones móviles. Una herramienta estática automatizada y de escaneo de seguridad dinámica específica para la capa de interfaz de usuario móvil (Android/iOS/Windows) la proporciona una herramienta de código abierto llamada **Mobile Security Framework (MobSF)**. GitLab, una popular plataforma DevOps, también proporciona **AST para aplicaciones móviles con MobSF**. Más adelante, en este mismo capítulo, veremos cómo utilizar MobSF y otra herramienta automatizada de análisis de seguridad.

Más allá de confiar en herramientas automatizadas de escaneado de seguridad, es esencial que los equipos de software conozcan los **10 principales riesgos OWASP para aplicaciones móviles** y los aborden durante el desarrollo. Dependiendo del nivel de conocimientos del equipo, también puede ser necesario contratar a expertos en pruebas de penetración después del desarrollo.

Por último, dado que las pruebas de seguridad de las aplicaciones móviles son un área bastante especializada en el momento de escribir estas líneas, recomiendo estar al tanto de la sabiduría colectiva de la comunidad OWASP en a través de su **Guía de Pruebas de Seguridad Móvil**.

Pruebas de rendimiento

Del Capítulo 8, ya sabes cómo configurar pruebas automatizadas de carga/estrés/remojo para tus servicios. Sigue midiendo y monitorizándolos.

Cuando se trata de la capa de interfaz de usuario móvil, el rendimiento puede significar algunas cosas diferentes que en una interfaz de usuario web, ya que las aplicaciones móviles suelen funcionar en un entorno con recursos limitados y compartidos de CPU, memoria, batería y condiciones de red. Para consolidar el enfoque de las pruebas de rendimiento móvil, asegúrate de dos aspectos:

1. La aplicación no debe monopolizar ni agotar ninguno de los recursos críticos del dispositivo, como la CPU, la memoria y la batería.
2. La aplicación debe responder rápidamente a las acciones del usuario final.

Para probar el punto 1, puedes utilizar herramientas de perfilado en los respectivos sistemas operativos; por ejemplo, el **perfilador de Android** en Android Studio y **XCode Instruments** para iOS. Las validaciones sobre el consumo de recursos pueden añadirse como pruebas unitarias automatizadas utilizando las herramientas respectivas e integrarse con los pipelines CI para pruebas continuas de rendimiento. Appium también proporciona una API para obtener datos de rendimiento similares para aplicaciones Android, como verás más adelante en este capítulo.

Cuando pruebas el punto 2, presta especial atención a elementos como el tiempo de lanzamiento de la aplicación, o el tiempo que transcurre desde que haces clic en el ícono de la aplicación hasta que se abre. Éste debe ser **inferior a 5 segundos**. Del mismo modo, la respuesta a cualquier tipo de acción dentro de la aplicación debe ser inferior a 3 s, o aumentará la tasa de rebote. Los retrasos, sin embargo, se ven afectados significativamente por el ancho de banda de la red cuando hay llamadas a servicios. Puedes simular diferentes condiciones de red en tu emulador o simulador para medir el tiempo de respuesta de la app. Otra parte de las pruebas de rendimiento dentro de la aplicación son las pruebas de estrés. Para estresar la

aplicación, puedes desencadenar múltiples acciones rápidamente, como tocar varios botones, acercar y alejar, enviar solicitudes, navegar por páginas, etc., para ver si la aplicación se bloquea. Android viene con una herramienta automatizada llamada Monkey que realiza pruebas de estrés automatizadas; exploraremos esa herramienta más adelante en este capítulo.

En resumen, como ocurre con muchos de los otros tipos de pruebas, cuando pienses en las pruebas de rendimiento de las aplicaciones móviles, debes considerarlas a través de las tres lentes.

Pruebas de accesibilidad

La WAI del W3C proporciona directrices detalladas sobre cómo **aplicar las WCAG 2.0 a las aplicaciones móviles**. Las pautas de accesibilidad móvil siguen los mismos cuatro principios clave: la aplicación debe ser perceptible, operable, comprensible y robusta. Algunas de las características que hay que comprobar son la posibilidad de acercar y alejar la imagen, la legibilidad en tamaños de pantalla pequeños, el contraste de color entre los elementos, un espacio razonable para hacer clic en los botones, un diseño coherente en toda la aplicación, la colocación de los elementos adecuados dentro del área de visualización sin que el usuario tenga que desplazarse, etc. Tanto iOS como Android proporcionan bastantes herramientas para comprobar la accesibilidad, como se indica en los siguientes subapartados, aunque las herramientas automatizadas de comprobación de la accesibilidad son actualmente limitadas.

iOS

iOS proporciona las siguientes herramientas para las pruebas de accesibilidad de :

- El lector de pantalla VoiceOver está disponible en tanto en dispositivos físicos como en simuladores de iOS. Puede utilizarse

para probar de extremo a extremo los flujos de usuario.

- El **inspector de accesibilidad de XCode** está disponible en los simuladores de iOS para inspeccionar elementos y comprobar si tienen establecidos adecuadamente atributos relacionados con la accesibilidad. Esto puede utilizarse con fines de depuración.

Android

Android tiene un soporte más elaborado para las pruebas de accesibilidad de shift-left . Empezando por la izquierda, las herramientas son

- **Android Studio**, un entorno de desarrollo, puede configurarse para que muestre avisos de pelusa por diversos problemas de accesibilidad durante el desarrollo.
- **Espresso** (y Robolectric, antes de la versión 4.5) es una herramienta nativa de automatización de pruebas de aplicaciones UI de Android con la posibilidad de escanear cada vista de la aplicación en busca de disposiciones de accesibilidad. Estas pruebas pueden integrarse con un conjunto de pruebas Espresso existente y ejecutarse en CI.
- TalkBack es el lector de pantalla integrado en Android. Puede utilizarse para probar de extremo a extremo los flujos de usuario.
- **El Escáner de accesibilidad** es una herramienta que audita las aplicaciones móviles en busca de problemas de accesibilidad. Puede utilizarse durante la fase de prueba manual de la historia de usuario.

Además, Android tiene una función de Acceso por Interruptor que permite el uso de dispositivos de asistencia externos para interactuar con las apps (conocidos como interruptores), y admite BrailleBack para conectar una pantalla braille al dispositivo y Acceso por Voz para controlar un dispositivo Android con comandos hablados.

Durante la presentación de la aplicación, la tienda Google Play incluso proporciona a los equipos informes de auditoría de accesibilidad previos al lanzamiento.

Pruebas CFR

Los CFR tratados en el Capítulo 10 siguen siendo relevantes en las pruebas móviles: por ejemplo, auditabilidad, portabilidad, fiabilidad, compatibilidad, etc. Además de la seguridad y la accesibilidad , entre los CFR en los que debes centrarte explícitamente al probar la interfaz de usuario móvil se incluyen:

Usabilidad

Si lo piensas, un dispositivo móvil es un objeto muy personal. Como ya se ha dicho, la mayoría de los adultos pasan de dos a tres horas al día en sus teléfonos, y estos dispositivos se han convertido en como extremidades adicionales para muchos de nosotros. En consecuencia, la usabilidad es una preocupación importante. Los usuarios finales pueden estar convencidos de descargar una aplicación basándose en la fanfarria, pero sólo es probable que la utilicen continuamente si pueden personalizarla a su gusto. Por ejemplo, el usuario final puede ser zurdo o diestro, tener el hábito de la multitarea con varias aplicaciones abiertas, utilizar aplicaciones mientras conduce, ser multilingüe, preferir un tipo de interacción a otro, etc. Todos estos factores deben tenerse en cuenta al realizar las pruebas desde el punto de vista de la usabilidad. Por supuesto, puede que no sea posible satisfacer las necesidades de los más de 7.700 millones de individuos únicos; más bien, se trata de afinar nuestra mentalidad para considerar específicamente estos aspectos de usabilidad al probar aplicaciones móviles. El enfoque de las pruebas de usabilidad del Capítulo 10 también puede aplicarse aquí. Además, una recomendación importante es hacer un estudio preliminar sobre los comportamientos del usuario final en

tu mercado/regiones objetivo. Google proporciona ayuda en este sentido: el [sitio Think with Google](#) ofrece infografías detalladas sobre el comportamiento de los usuarios de móviles en varios países, además de otros informes clave relacionados con los móviles.

Interrupciones

Este es un sabor específico para móviles del CFR de fiabilidad. Dado que los dispositivos móviles se utilizan para diversos fines, incluidos la mensajería y las llamadas telefónicas, es un hecho que el flujo de cualquier app puede verse interrumpido por distracciones externas. El comportamiento típico del usuario es mantener la aplicación actual en segundo plano mientras atiende a tales distracciones, como llamadas telefónicas entrantes o notificaciones importantes del chat. Una vez atendidas, se reanuda el flujo de la app.

Así que, al realizar pruebas móviles, es importante tener en cuenta los comportamientos de interrupción. ¿Qué ocurre con la solicitud actual cuando la aplicación se aparca de repente en segundo plano? ¿Qué ocurre con la autenticación cuando la aplicación se detiene y luego se reanuda desde el fondo? ¿Qué ocurre con una solicitud en curso cuando la aplicación se cierra o se cierra bruscamente? ¿Qué ocurre si el dispositivo se queda sin batería durante el flujo de trabajo de la aplicación? Recuerda que esto es un CFR y debe probarse en toda la aplicación.

Instalabilidad y actualizabilidad

La instalación de aplicaciones en distintos dispositivos y sistemas operativos desde las respectivas tiendas de aplicaciones es un aspecto esencial de las pruebas móviles. La instalación requiere una cierta cantidad de espacio de almacenamiento local en el dispositivo. Además, durante la instalación, la aplicación puede pedir a los usuarios los permisos pertinentes para acceder al

hardware del dispositivo o a otras aplicaciones (cámara, micrófono, contactos, galería de fotos, servicios de localización, etc.). Hay que probar estos escenarios de instalación, incluyendo casos de fallo como la falta de espacio de almacenamiento suficiente, la denegación de permisos de la app y la incompatibilidad con la versión del SO del dispositivo. Además, las pruebas de actualización de la app deben garantizar que no se rompan los flujos existentes. Por ejemplo, los cambios en las estructuras de las bases de datos locales, si los hubiera, no deberían afectar a las funciones existentes. Además, el usuario no debe salir de la aplicación tras la actualización. Recuerda probar el caso de actualizar desde versiones anteriores de la app y no sólo desde la última versión de la app. Si las actualizaciones requieren nuevos permisos de la aplicación, también hay que probarlos.

Como la instalación y las actualizaciones dependen de las condiciones de la red, incluye también varios escenarios relacionados con la red. Del mismo modo, asegúrate de que la desinstalación de la aplicación funciona perfectamente.

Monitoreo

A diferencia de lo que ocurre con las aplicaciones web, las caídas de aplicaciones son bastante comunes en el mundo móvil, y por tanto el monitoreo es un CFR crítico cuando se trata de aplicaciones móviles. A veces, las condiciones que provocan el bloqueo de una aplicación son difíciles de reproducir, y las herramientas de monitoreo (Firebase Crashlytics, Dynatrace, New Relic, etc.) pueden ser clave para comprender el problema. Por tanto, incluso durante la fase de desarrollo, deberías integrar estas herramientas en el entorno de pruebas para que sea más fácil depurar los fallos de la aplicación.

Ten en cuenta que las pruebas de algunos de estos CFR, como la instalación y actualización correctas en tus dispositivos y sistemas operativos de destino o el **comportamiento ante una interrupción**, pueden automatizarse utilizando pruebas funcionales a nivel micro y macro para obtener una retroalimentación continua -recuerda aquí la estrategia de pruebas continuas del [Capítulo 4](#)-.

Esto nos lleva al final de la estrategia de pruebas móviles. A continuación, nos pondremos manos a la obra con la creación de conjuntos de pruebas utilizando algunas de las herramientas mencionadas en esta sección.

Ejercicios

Estos ejercicios te guiarán en la configuración de un marco Java-Appium para crear pruebas funcionales y visuales de la interfaz de usuario. He elegido [Appium](#) porque, como ya he dicho, es compatible con los tres tipos de aplicaciones móviles (nativas, web e híbridas) y puede funcionar en varios sistemas operativos.

Appium

Appium es una herramienta de código abierto apoyada por una vibrante comunidad. Como herramienta de automatización multiplataforma, Appium agrupa los marcos de automatización específicos del sistema operativo, como XCUITest (proporcionado por Apple para iOS) y UiAutomator (proporcionado por Google para Android) bajo un conjunto común de API envolventes: las API WebDriver que conocimos en [el Capítulo 3](#). Por ejemplo, Appium utiliza el objeto DesiredCapabilities para instanciar el objeto driver e interactuar con la aplicación. Además, las API `findElements(By.id)`, `click()`, `isElementPresent()`, y otras, siguen siendo las mismas. Como resultado, la curva de aprendizaje es muy ligera si te sientes cómodo con las pruebas automatizadas utilizando

Selenium WebDriver. Además, al igual que WebDriver, Appium es independiente del idioma. Esto significa que puedes escribir pruebas en el lenguaje de programación que prefieras, como Ruby, Python, Java, JavaScript, etc., con las respectivas bibliotecas cliente.

Appium ha anunciado su próxima versión principal, la 2.0, que redefine cómo deben instalarse el servidor Appium, los controladores de automatización y los plug-ins. Por ejemplo, los controladores de automatización específicos del sistema operativo estaban incluidos en Appium 1.x, mientras que en 2.x deben instalarse por separado. En el momento de escribir estas líneas, se encuentra en fase beta y se espera que se publique en 2022. Dado que es el camino a seguir para Appium, utilizaremos la versión beta 2.x para este ejercicio. Utilizaremos Android, pero dado que las API de Appium son las mismas en todos los sistemas operativos, deberías poder aplicar pasos similares para escribir pruebas también para aplicaciones iOS.

APPIUM, ¡UNA HERRAMIENTA RPA!

La automatización robótica de procesos (RPA) es un tema candente en la industria en estos días. Se considera que es una forma de reducir la carga de las tareas manuales mundanas relacionadas con los procesos y acelerar la eficacia operativa automatizando los procesos empresariales de principio a fin. En otras palabras, se refiere a la automatización de procesos empresariales típicos, como capturar datos en una hoja de cálculo, introducirlos en una herramienta interna, activar un trabajo para procesar los datos, verificar que aparecen en línea, etc.

Appium 1.x, aunque se utiliza principalmente para la automatización de aplicaciones móviles, admite la automatización de aplicaciones de escritorio [de Windows](#) y [Mac](#) con los controladores pertinentes. Además, no requiere que la aplicación bajo prueba sea desarrollada por el equipo que crea las pruebas para poder interactuar con ella (es decir, no necesita el código fuente). Así que, junto con sus capacidades Selenium WebDriver, Appium 1.x puede utilizarse [como herramienta RPA](#). Esperemos que esta compatibilidad se mantenga en Appium 2.x.

¡Empecemos!

Requisitos previos

Los requisitos previos son similares a los de las otras herramientas de automatización de las que hablamos en [el Capítulo 3](#), así que comprueba si los tienes antes de instalarlas. Necesitarás

- [Node.js](#), para configurar el servidor Appium
- La última versión de [Java](#) (utilizaremos la biblioteca cliente Java de Appium para este ejercicio)
- Un IDE como [IntelliJ](#)

- **Maven**

Emulador de Android

Una vez que hayas configurado correctamente todos los requisitos previos, crea un emulador de Android en el que ejecutar tu prueba Appium siguiendo estos pasos:

1. Descarga y configura **Android Studio**. Esto trae consigo el SDK de Android y las herramientas necesarias.
2. En Android Studio, selecciona Más acciones → Gestor de AVD. (AVD significa Dispositivo Virtual Android).
3. Haz clic en Crear dispositivo virtual para ver una lista de perfiles de hardware existentes para tabletas, teléfonos, dispositivos Wear OS, etc. Elige la categoría Teléfono y selecciona un perfil - por ejemplo, Pixel 2, 5,0 pulgadas- y haz clic en Siguiente.
4. Selecciona una versión del sistema operativo Android, como Android 8.0. Se te dará la opción de descargarlo si no tienes la versión solicitada.
5. Dale un nombre al emulador en la siguiente pantalla -por ejemplo, "Oreo"- y haz clic en Finalizar. Tu emulador de Android 8.0 Pixel 2 debería aparecer ahora en la lista de dispositivos virtuales disponibles.
6. Pulsa el botón reproducir/ejecutar para iniciar el emulador.

Para este ejercicio, puedes descargar la aplicación demo para Android *ApiDemos-debug.apk* [del repositorio GitHub de Appium](#). Instala la app arrastrándola y soltándola dentro del emulador, luego ábrela y echa un vistazo para familiarizarte con ella.

Configuración de Appium 2.0

Para configurar Appium, sigue estos pasos:

1. Ejecuta el siguiente comando para instalar Appium v2.0:

```
$ npm install -g appium@next
```

NOTA

Ten en cuenta que este paso está sujeto a cambios tras el lanzamiento oficial.

2. Configura el controlador UiAutomator2 ejecutando el siguiente comando:

```
$ appium driver install uiautomator2
```

NOTA

Para iOS necesitarás el controlador XCUITest, que se puede instalar utilizando el comando `appium driver install xcuitest`.

3. Inicia el servidor Appium con el siguiente comando:

```
$ appium server -ka 800 -pa /wd/hub
```

4. Descarga **Appium Inspector**, una herramienta GUI que nos permite encontrar los localizadores de elementos en una aplicación móvil.

Flujo de trabajo

Como ya se ha mencionado, Appium utiliza el objeto `DesiredCapabilities` para instanciar una conexión con la app móvil. En Appium Inspector, puedes configurar este objeto a través de la

GUI y conectar con la app para inspeccionar los elementos. Prueba a inspeccionar la aplicación Android de demostración siguiendo estos pasos:

1. Abre el inspector y proporciona los valores de Capacidades deseadas que se muestran en [la Figura 11-5](#). Guárdalos para poder reutilizarlos más adelante.

 Appium Server Select Cloud Providers

Remote Host 127.0.0.1 Remote Port 4723

Remote Path /wd/hub SSL

> Advanced Settings

Desired Capabilities Saved Capability Sets 0 Attach to Session...

deviceName	text	V	Android Emulator	
platformName	text	V	android	
automationName	text	V	UiAutomator2	
app	filepath	V	/Users/gayath	 

Automatically add necessary Appium vendor prefixes on start 

JSON Representation

```
{  
  "deviceName": "Android Emulator",  
  "platformName": "android",  
  "automationName": "UiAutomator2",  
  "app": "/Users/gayathri/Desktop/My  
Book/Code/AppiumExample/src/main/resources/apps/ApiDemos-  
debug.apk"  
}
```

Figura 11-5. Capacidades deseadas para conectarse a la aplicación de demostración

2. Haz clic en Iniciar sesión. Esto debería abrir la aplicación de demostración dentro de la ventana del inspector.
3. Haz clic en el ícono Seleccionar elementos (el tercer ícono de la barra superior), y pasa el ratón por encima de la aplicación para inspeccionar los elementos. Verás que los elementos se resaltan a medida que pasas el ratón sobre ellos. Selecciona uno, y las propiedades del elemento se mostrarán en el panel derecho, como se ve en la **Figura 11-6**.

The screenshot shows the API Demos application running on an Android device. The title bar displays "API Demos" and the time "3:17". The navigation bar at the bottom includes icons for back, home, and recent apps.

The main content area shows a list of categories:

- Accessibility
- Animation
- App
- Content
- Graphics
- Media
- NFC
- OS
- Preference
- Text
- Views

The "Accessibility" category is highlighted with a blue background.

On the right side, there are two panels:

- Source** tab is selected. The "App Source" panel displays the XML code for the current screen:

```
<android.widget.FrameLayout>
    <android.view.ViewGroup resource-
        id="android:id/decor_content_parent">
```
- Actions** tab is also present.
- Selected Element** panel shows the details of the selected element:
 - Icon bar: φ, ↗, ⌂, ⌚, ⌚
 - Find By: Selector
 - accessibility id: Accessibility
 - xpath: //android.widget.Text
View[@content-desc
="Accessibility"]
 - Attribute Value table:

Attribute	Value
elementId	0000000-0000-00 d8-ffff-ffff000000 5
index	1
package	io.appium.android.ap is
class	android.widget.TextVi ew
text	Accessibility
content-desc	Accessibility
resource-id	android:id/text1
checkable	false

Figura 11-6. Inspeccionar elementos en Appium Inspector

El panel de la derecha tiene los atributos del elemento, como `resource-id`, `class`, `text`, etc., que pueden utilizarse como localizadores de elementos. Observa el valor `package`, `io.appium.android.apis`. Lo necesitarás para escribir pruebas. También puedes enviar comandos como Pulsar, Enviar teclas y Borrar a los elementos seleccionados pulsando los botones del panel derecho. Esto es útil para la depuración, por ejemplo para comprobar si un Toque en un elemento te lleva a la página siguiente como estaba previsto.

Suena fácil, ¿verdad? La configuración del marco de pruebas de la interfaz de usuario también es sencilla.

La configuración del marco Java-Appium es similar a la configuración del marco Java-Selenium comentada en [el Capítulo 3](#). Puedes utilizar Maven y TestNG junto con la biblioteca cliente de Appium. También puedes adoptar el Modelo de Objetos de Página para las pruebas de interfaz de usuario móvil.

Tomemos un caso de prueba sencillo para automatizar: abre la app de demostración y comprueba el texto del segundo elemento, que es "Accesibilidad", en la página de inicio. Sigue estos pasos:

1. Abre IntelliJ y crea un nuevo proyecto Maven llamado `AppiumExample`.
2. Añade una dependencia Java de Appium y una dependencia TestNG a tu archivo `pom.xml`. (Consulta [el Capítulo 3](#) si necesitas un repaso de estos dos pasos).
3. Crea una nueva carpeta llamada `apps` en `/src/main/resources` y copia y pega aquí el archivo de la aplicación de demostración, `ApiDemos-debug.apk`.
4. Crea un paquete `pages` en `/src/main/java`. Crea los paquetes `tests` y `base` en `/src/test/java`. Tus clases de página van en el

paquete pages, y las clases de prueba van en el paquete tests. Las clases de configuración van en el paquete base.

5. La clase Base contiene los métodos de configuración y desmontaje de Appium con DesiredCapabilities configurado con el nombre del paquete de la app, la ruta de la app, el nombre del emulador, el nombre del dispositivo, el nombre de la plataforma y el nombre del marco de automatización, como se ve en [el Ejemplo 11-1](#).

Ejemplo 11-1. La clase Base con la configuración de Appium

```
// src/test/java/base/Base.java

package base;

import io.appium.java_client.MobileElement;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.remote.MobileCapabilityType;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.testng.annotations.*;
import java.io.File;

public class Base {
    protected AndroidDriver<MobileElement> driver;

    @BeforeMethod
    public void setUp(){
        File appDir = new File("src/main/resources/apps");
        File app = new File(appDir, "ApiDemos-debug.apk");

        DesiredCapabilities capabilities = new
DesiredCapabilities();
        capabilities.setCapability(MobileCapabilityType.DEVICE_NAME,
                            "Android Emulator");

        capabilities.setCapability(MobileCapabilityType.PLATFORM_NAME,
                            "android");

        capabilities.setCapability(MobileCapabilityType.AUTOMATION_NAME,
                            "UiAutomator2");
        capabilities.setCapability(MobileCapabilityType.APP,
```

```

        app.getAbsolutePath());
capabilities.setCapability("avd", "Oreo");
capabilities.setCapability("appPackage",
"io.appium.android.apis");
driver = new AndroidDriver<MobileElement>(capabilities);
}

@AfterMethod
public void tearDown(){
    driver.quit();
}
}

```

6. El **ejemplo 11-2** muestra la clase `HomePage`, con un método que obtiene el texto del segundo elemento de la página de inicio identificado por su id. El id proporcionado aquí es el valor del atributo `resource-id`.

Ejemplo 11-2. La clase `HomePage` con localizador de elementos y acciones

```

// src/main/java/pages/HomePage.java

package pages;

import io.appium.java_client.MobileElement;
import io.appium.java_client.android.AndroidDriver;
import org.openqa.selenium.By;

public class HomePage{

    private AndroidDriver<MobileElement> driver;
    private By textItem = By.id("android:id/text1");

    public HomePage(AndroidDriver<MobileElement> driver) {
        this.driver = driver;
    }

    public String getFirstTextItem(){
        return driver.findElements(textItem).get(1).getText();
    }
}

```

7. El Ejemplo 11-3 muestra la clase HomePageTest, con una prueba que abre la aplicación y afirma el texto del segundo elemento de la página de inicio utilizando TestNG.

Ejemplo 11-3. La clase HomePageTest con la primera prueba

```
// src/test/java/tests/HomePageTest.java

package tests;

import base.Base;
import org.testng.Assert;
import org.testng.annotations.Test;
import pages.HomePage;

public class HomePageTest extends Base {

    @Test
    public void verifyFirstTextItemOnHomePage() throws Exception {
        HomePage homePage = new HomePage(driver);
        Assert.assertEquals(homePage.getFirstTextItem(),
"Accessibility");
    }
}
```

8. Puedes ejecutar la prueba desde el IDE o ejecutando el comando `mvn clean test` en tu terminal. Verás cómo se ejecuta la prueba en el emulador (si el emulador no está abierto, Appium lo abrirá y luego ejecutará la prueba). Cuando ejecutes pruebas desde la línea de comandos, también podrás obtener informes HTML en `/target/surefire-reports/`.

Puedes añadir las pruebas a tus canalizaciones CI para realizar pruebas continuas de tu aplicación móvil. Si necesitas API adicionales para automatizar los casos de prueba de tu aplicación en , como el toque, el desplazamiento o el deslizamiento, consulta la [documentación oficial](#) de Appium.

Plugin de pruebas visuales de Appium

El plug-in de pruebas visuales de Appium 2.0 utiliza OpenCV, una herramienta de procesamiento de imágenes de código abierto , para realizar la comparación de imágenes. Comparado con Applitools Eyes, que vimos en el [Capítulo 6](#), este plug-in tiene capacidades limitadas. Por ejemplo, Applitools Eyes puede desplazarse por toda la página y hacer una comparación visual sin programación adicional, mientras que con Appium tenemos que escribir código para hacer varias capturas de pantalla y unirlas antes de pasar el resultado a la comparación de imágenes. Sin embargo, el complemento de Appium es de código abierto y, por tanto, nos permite añadir al menos un conjunto mínimo de pruebas visuales junto con el conjunto de pruebas funcionales de Appium sin coste adicional.

Vamos a añadir algunas pruebas visuales a la misma prueba de interfaz de usuario que hemos creado antes.

Configurar

Para configurar el complemento, sigue estos pasos:

1. Instala OpenCV ejecutando el siguiente comando:

```
$ npm install -g opencv4nodejs
```

2. Instala el plug-in de pruebas visuales de Appium utilizando el siguiente comando:

```
$ appium plugin install images
```

3. Inicia el servidor Appium con este comando:

```
$ appium server -ka 800 --use-plugins=images -pa  
/wd/hub
```

Flujo de trabajo

El plug-in de Appium, `images`, proporciona dos API para realizar pruebas visuales. Una es hacer una comparación de imágenes entre la imagen de referencia y la imagen real, como se ve aquí:

```
SimilarityMatchingResult result =  
    driver.getImagesSimilarity(baselineImg, actualScreen, options);
```

Y la otra es obtener una puntuación de comparación del objeto `result`, que puede utilizarse para suspender la prueba si es inferior a un valor umbral, como se ve aquí:

```
result.getScore() < 0.99
```

La puntuación de la comparación oscila entre 0 y 1. El valor esperado ideal es 1, pero debido a variaciones sutiles puede que no siempre obtengas la puntuación ideal; puedes utilizar el valor umbral como forma de controlar la sensibilidad de la prueba según convenga a las necesidades de tu proyecto.

Con estas dos API, el flujo de trabajo de las pruebas visuales es sencillo: creas un conjunto de capturas de pantalla base de las pantallas de la aplicación, las comparas con las versiones actuales de las capturas de pantalla de la aplicación y suspendes la prueba si la puntuación es inferior al umbral. **El Ejemplo 11-4** muestra la prueba de interfaz de usuario de Appium creada anteriormente con aserciones visuales adicionales. El código del ejemplo también crea capturas de pantalla base en la primera ejecución de la prueba por defecto. Ten en cuenta que tendrás que crear una clase `BasePage` y añadir allí el código de configuración correspondiente.

Ejemplo 11-4. Pruebas visuales automatizadas con el plug-in Appium 2.0

```
// src/main/java/pages/BasePage.java
```

```
package pages;
```

```
import io.appium.java_client.MobileElement;
import io.appium.java_client.imagecomparison.SimilarityMatchingOptions;
import io.appium.java_client.imagecomparison.SimilarityMatchingResult;
import org.openqa.selenium.OutputType;
import io.appium.java_client.android.AndroidDriver;
import java.io.File;
import org.apache.commons.io.FileUtils;

public class BasePage {
    private File baselineDir = new
File("src/main/resources/baseline_screenshots");

    public void checkVisualQuality(String screen_name,
        AndroidDriver<MobileElement> driver) throws Exception {
        File baselineImg = new File(baselineDir, screen_name + ".png");
        File actualScreen = driver.getScreenshotAs(OutputType.FILE);

        if (baselineImg.exists()) {
            SimilarityMatchingOptions options = new
SimilarityMatchingOptions();
            options.withEnabledVisualization();
            SimilarityMatchingResult result =
                driver.getImagesSimilarity(baselineImg,
actualScreen, options);

            if (result.getScore() < 0.99) {
                File imageDiff = new
File("src/main/resources/baseline_screenshots"
                    + "FAIL_" + screen_name +
".png");
                result.storeVisualization(imageDiff);
                throw new Exception("Visual quality hampered");
            }
        } else {
            FileUtils.copyFile(actualScreen, baselineImg);
        }
    }
}

// src/test/java/tests/HomePageTest.java

public class HomePageTest extends Base {
```

```
@Test
public void verifyFirstTextItemOnHomePage() throws Exception {
    HomePage homePage = new HomePage(driver);
    Assert.assertEquals(homePage.getFirstTextItem(),
"Accessibility");
    BasePage basePage = new BasePage();
    basePage.checkVisualQuality("home_page", driver);
}
}
```

El complemento proporciona una API, `result.storeVisualization()`, para ver las diferencias entre las dos imágenes cuando falla una prueba. Para ver cómo funcionan las pruebas, primero, ejecuta `mvn clean test` desde la línea de comandos. Esto creará una captura de pantalla base de la página de inicio en `/src/main/resources/baseline_screenshots`. Ahora, cuando vuelvas a ejecutar la prueba, debería pasar, ya que no se han realizado cambios en la aplicación. Para que la prueba falle, puedes dar un archivo `.png` diferente como línea de base y volver a ejecutarla. Verás una nueva imagen, como se muestra en la [Figura 11-7](#), generada en la misma carpeta `baseline_screenshots`, en la que se destacan las diferencias de las imágenes.



12:05



API Demos

API Demos

Accessibility

Bouncing Balls

Accessibility

Cloning

Animation

Custom Evaluation

App Layout Animations

Default Layout Animations

Content

Events

Graphics Animation

Hide/Show Animations

Media Animation

Layout Animations

NFC

Loading

OS Graphics

Multiple Properties

Preference

Reversing

Text

Seeking

ViewFlip

ViewFlip

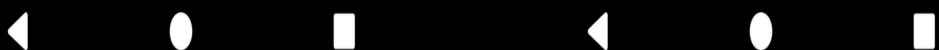


Figura 11-7. Resultados de los fallos de la prueba visual

Hasta ahora, hemos hablado de un enfoque paso a paso para añadir pruebas de interfaz de usuario automatizadas para validar el comportamiento funcional y la calidad visual de una aplicación móvil utilizando Appium. Estos son los dos tipos de pruebas móviles que se practican con más frecuencia. A continuación, exploraremos algunas herramientas adicionales que ayudan en otros tipos de pruebas móviles.

Herramientas de comprobación adicionales

En esta sección, conocerás algunas herramientas que pueden utilizarse para realizar pruebas de rendimiento, seguridad, accesibilidad y datos. Aunque algunos de estos tipos de pruebas no se practican actualmente de forma generalizada en el espacio móvil, es aconsejable empezar a aplicarlas siempre que sea apropiado, por las mismas razones por las que deben adoptarse en un contexto web.

NOTA

Las ilustraciones de esta sección muestran principalmente el flujo de trabajo de las herramientas con Android. Sin embargo, el mismo flujo de trabajo puede aplicarse también para iOS. En los casos en que existen herramientas paralelas para iOS, se destacan en la sección correspondiente del debate anterior sobre la estrategia de pruebas móviles.

Inspector de bases de datos de Android Studio

Si quieras explorar la base de datos local de la aplicación móvil, Android Studio proporciona una **herramienta Inspector de base de datos** que facilita el acceso a la GUI. Como cualquier otro cliente de

base de datos, puedes utilizar la interfaz de la herramienta para añadir/editar/borrar datos y verificar el comportamiento de la app.

Para utilizar el Inspector de Bases de Datos:

1. Selecciona Más acciones → Perfil o Depurar APK en Android Studio, y elige el archivo .apk de la aplicación. Ten en cuenta que este archivo debe tener activada la opción de depuración para que puedas utilizar esta herramienta.
2. Selecciona Ver → Ventana de herramientas → Inspección de aplicaciones. Se abrirá el panel Inspección de aplicaciones en la parte inferior de la pantalla.
3. Ejecuta tu aplicación en un emulador desde Android Studio utilizando el botón verde de ejecutar.
4. Ahora se abrirá el Inspector de Bases de Datos dentro del panel Inspección de la aplicación. Ten en cuenta que la app de demostración no tiene una BD local; **la Figura 11-8** muestra la base de datos local de otra app, sólo como ejemplo.

App Inspection

LowNetworkOreo > [REDACTED]

Database Inspector Background Task Inspector

Databases

Live updates

.db

Comment CommunityNewsFeed CommunityProfile CommunityTimelineFeed LeaderProfile NewsFeed

NewsFeed

Table is empty

actorId : TEXT, NOT NULL
id : TEXT, NOT NULL
objectId : TEXT, NOT NULL
textContent : TEXT
imageURLs : TEXT
 imageURLInfo : TEXT
leaderProfile : TEXT, NOT NULL
verb : TEXT, NOT NULL
time : TEXT, NOT NULL

50

TODO Problems Terminal Logcat Profiler App Inspection Run Event Log Layout Inspector

Figura 11-8. Vista de la herramienta Inspector de bases de datos de Android Studio

En general, puedes utilizarlo para asegurarte de que se almacenan los datos esperados para permitir el acceso sin conexión a la app y para verificar que la información sensible no se almacena sin encriptar.

Herramientas de pruebas de rendimiento

En "[Pruebas de rendimiento](#)", hablamos de los distintos aspectos del rendimiento que hay que probar en las aplicaciones móviles. Aquí exploraremos tres herramientas que pueden ayudarte con ello.

Mono

[Monkey](#) puede considerarse un equivalente de la Ingeniería del Caos para aplicaciones Android. Realiza secuencias aleatorias de acciones como toques, pulsaciones de teclas, clics y otros gestos en la interfaz de usuario de la aplicación e informa de los fallos de la aplicación, si los hay. Monkey se presenta como una sencilla herramienta de línea de comandos. Si ya tienes instalado Android Studio, puedes realizar una prueba de estrés de la aplicación Android de demostración en un emulador o en un dispositivo físico con el siguiente comando:

```
$ adb shell monkey -p "io.appium.android.apis" -v 2000
```

Este comando envía 2.000 eventos diferentes a la aplicación. También puedes observar la ejecución en el emulador/dispositivo. Cuando se produzcan excepciones no gestionadas o la aplicación no responda, Monkey detendrá la ejecución e informará de esos problemas. Puedes personalizar la prueba de estrés para que ejecute eventos específicos pasando los parámetros opcionales apropiados en el comando, como se indica en la [documentación](#).

Controles ampliados: Regulador de red

Otro aspecto de las pruebas de rendimiento que hemos comentado antes era probar el rendimiento de la aplicación en distintas condiciones de red . Los emuladores de Android permiten simular distintos tipos de red, como GSM, GPRS, perímetro, LTE, etc.

También puedes estrangular aún más el ancho de banda ajustando la intensidad de la señal a Buena, Moderada, Mala, Genial, etc. Para probarlo, haz clic en el botón "Más opciones" del panel lateral del emulador y selecciona Móvil en el panel de ajustes. Verás las opciones de estrangulamiento que se muestran en la [Figura 11-9](#). Observa los controles adicionales aparte del tipo de red y la intensidad de la señal, como el estado de los datos, el estado de la voz, etc. Puedes utilizarlos para personalizar aún más la red, según requiera tu caso de prueba.

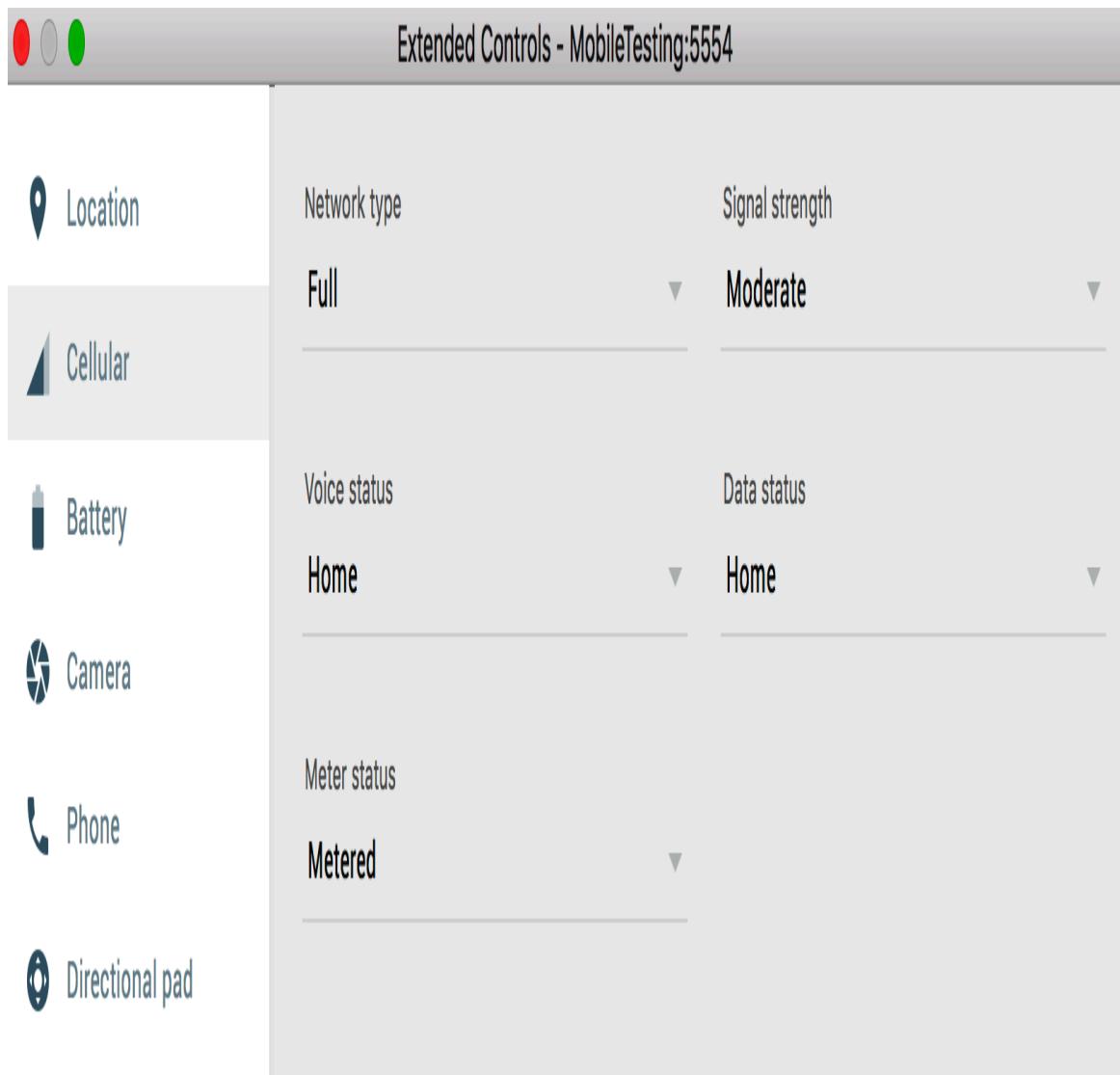


Figura 11-9. Opciones de estrangulamiento de red en el Emulador de Android

API de rendimiento de Appium

Appium proporciona una API para medir el rendimiento de una aplicación Android en términos de consumo de memoria, CPU, batería y red. En puedes añadir pruebas de rendimiento automatizadas junto con las pruebas de interfaz de usuario utilizando esta API, como se muestra aquí:

```
driver.getPerformanceData("package_name", "perf_type", timeout);
```

donde `package_name` es el nombre del paquete de la aplicación, que has utilizado antes para la automatización; `perf_type` se refiere al estado del sistema que quieras monitorizar, como CPU, red, etc.; y `timeout` es el número de segundos que la API sondeará los datos de rendimiento antes de lanzar un error. El valor exacto del parámetro `perf_type` puede obtenerse de otra API de Appium, `getSupportedPerformanceDataTypes()`. Actualmente, hay cuatro valores admitidos: `cpuinfo`, `memoryinfo`, `batteryinfo`, y `networkinfo`.

NOTA

Internamente, esta API está construida sobre la herramienta de línea de comandos `dumpsys` de Android, que emite diagnósticos de los servicios del sistema. Por lo tanto, sólo se puede utilizar con aplicaciones Android.

Utilizando esta API, podemos añadir una prueba de rendimiento, obtener las cifras de rendimiento en distintos puntos del flujo de usuario que está ejecutando la prueba de interfaz de usuario, y afirmar que las cifras están dentro de un determinado umbral. Por ejemplo, podemos afirmar el consumo de memoria respecto a un umbral después de una operación compleja en la aplicación. **El Ejemplo 11-5** muestra los datos de consumo de memoria justo después de abrir la aplicación Android de demostración.

Ejemplo 11-5. Consumo de memoria de la aplicación de demostración mediante la API de rendimiento de Appium

```
driver.getPerformanceData("io.appium.android.apis", "memoryinfo", 10);

// Output
[[totalPrivateDirty, nativePrivateDirty, dalvikPrivateDirty,
eglPrivateDirty, glPrivateDirty, totalPss, nativePss, dalvikPss, eglPss,
glPss, nativeHeapAllocatedSize, nativeHeapSize], [11432, 4708, 1692,
null, null, 20807, 4926, 1717, null, null, 12648, 14336]]
```

Para más detalles sobre la interpretación de estos valores de salida y la adición de aserciones apropiadas específicas de la aplicación, consulta [la documentación de dumpsys](#).

Herramientas de pruebas de seguridad

En esta sección veremos dos herramientas para realizar pruebas de seguridad automatizadas: MobSF y Qark.

MobSF

Como ya se ha mencionado en este capítulo, el Mobile Security Framework es una herramienta de código abierto que realiza análisis estáticos automatizados de y dinámicos de aplicaciones Android, iOS y Windows. También ayuda con el análisis de malware. Para probar MobSF, sigue estos pasos:

1. Descarga e instala [Docker Desktop](#), si aún no lo has hecho, y abre la aplicación. (No necesitas saber mucho sobre el uso de Docker para probar esto. Una advertencia, sin embargo: comprueba las políticas de tu empresa sobre la instalación de Docker en tu portátil de trabajo, ya que es gratuito sólo para uso personal).
2. Obtén el contenedor Docker MobSF ejecutando el siguiente comando:

```
$ docker run -it -p 8000:8000  
opensecurity/mobile-security-framework-mobsf:latest
```

3. Esto configurará MobSF en tu máquina local. Abre <http://0.0.0.0:8000> para ver la página web de MobSF.
4. Sube el APK de la aplicación Android de demostración a esta página web. Alternativamente, puedes utilizar el [APK de](#)

InsecureBankv2, creado intencionadamente con vulnerabilidades con fines didácticos.

5. MobSF escaneará la aplicación y mostrará los resultados en la misma página web local, como se ve en **la Figura 11-10**, resaltando las vulnerabilidades con su gravedad.



Static Analyzer



1	Debug Enabled For App [android:debuggable=true]	high	Debugging was enabled on the app which makes it easier for reverse engineers to hook a debugger to it. This allows dumping a stack trace and accessing debugging helper classes.
2	Application Data can be Backed up [android:allowBackup=true]	medium	This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
3	Activity (com.android.insecurebankv2.PostLogin) is not Protected. [android:exported=true]	high	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.

Figura 11-10. Resultados de la exploración MobSF

También puedes integrar la herramienta con pipelines CI como según la [documentación](#) para escanear automáticamente el código en el check-in.

Qark

[Qark](#) es otra herramienta de escaneo de seguridad de código abierto para aplicaciones Android. Puede escanear el código fuente o los APK. Qark es una herramienta basada en Python. Puedes instalarla con el gestor de paquetes pip utilizando este comando:

```
$ pip install qark
```

y ejecuta el escaneo de seguridad contra tu APK utilizando este comando:

```
$ qark --apk ~/path/to/apk --report-type html
```

Esto generará un archivo de informe HTML destacando las vulnerabilidades.

Como se menciona en [el Capítulo 7](#), las herramientas automatizadas de análisis de seguridad como éstas ayudan a los equipos de desarrollo de software a desplazar las pruebas de seguridad hacia la izquierda. Sin embargo, dependiendo de la habilidad del equipo para realizar pruebas de seguridad y del contexto de la aplicación, puede que aún necesites contratar a probadores de seguridad profesionales hacia el final del desarrollo.

Escáner de accesibilidad

Accessibility Scanner es una herramienta de escaneo de accesibilidad para aplicaciones Android disponible en [Google Play](#). Una vez que lo hayas instalado en tu dispositivo y le hayas concedido los permisos

necesarios, puedes iniciar la aplicación que quieras probar y pulsar el botón azul de la marca de verificación para iniciar el escaneo de accesibilidad. A continuación, aparecerá la opción de grabar el flujo de la aplicación, como se ve en la [Figura 11-11](#).

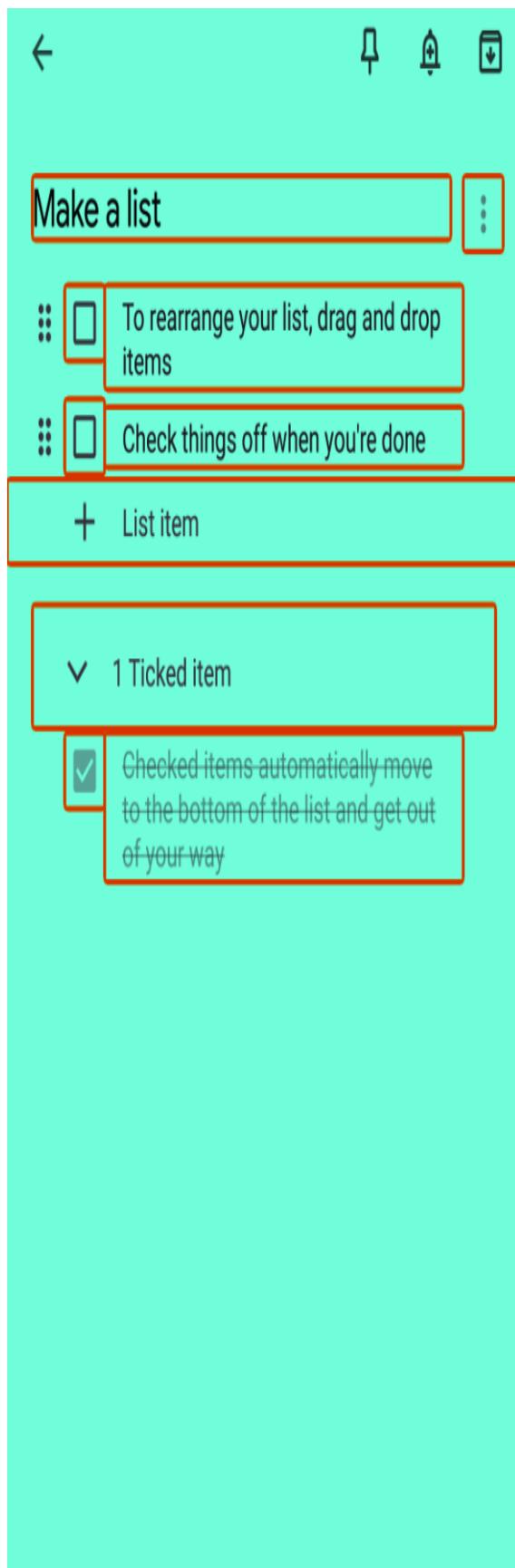
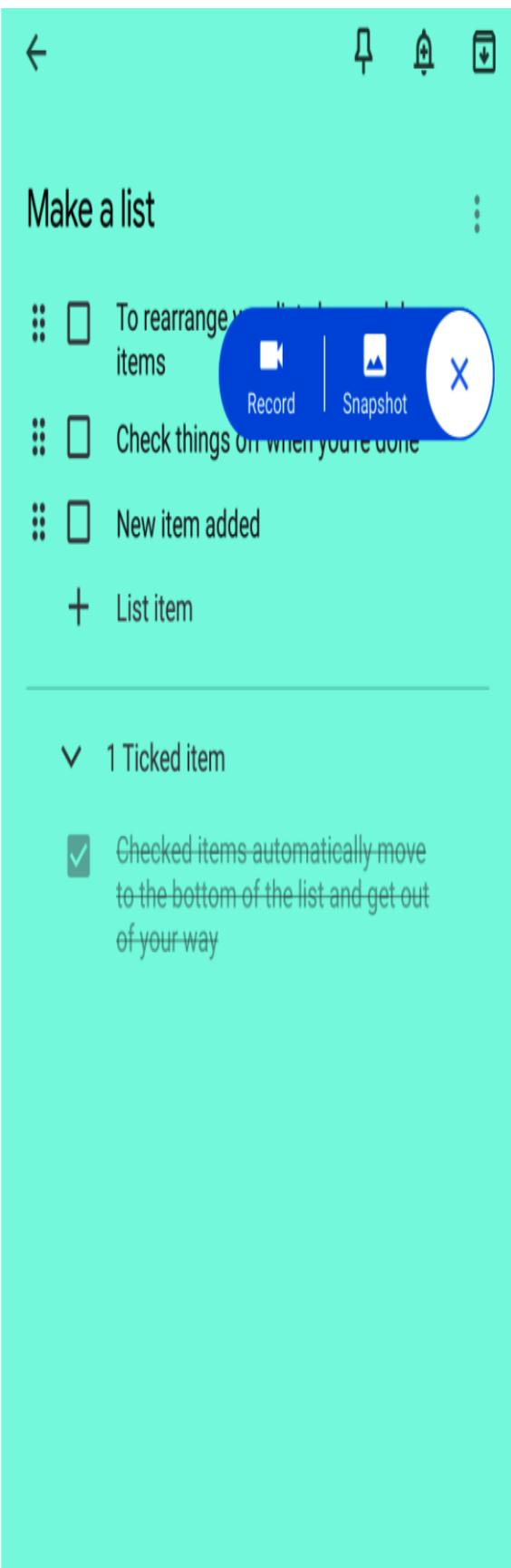


Figura 11-11. Resultados de la app Escáner de accesibilidad

Pruébalo ahora: abre cualquier aplicación en el dispositivo Android y empieza a grabar. Navega por la aplicación y, cuando hayas terminado, detén la grabación pulsando de nuevo la marca de verificación azul. La aplicación Escáner de Accesibilidad mostrará entonces los resultados de accesibilidad de cada pantalla por la que hayas navegado, resaltando los elementos cuyas disposiciones de accesibilidad pueden mejorarse, como se ve en la **Figura 11-11**. Al hacer clic en el texto resaltado se muestra una descripción de los problemas. Esta aplicación puede utilizarse ya en la fase de desarrollo para comprobar si faltan funciones de accesibilidad.

Y con esto, te mereces una gran felicitación, ya que has navegado con éxito por una amplia gama de herramientas para probar diversos requisitos funcionales y multifuncionales de aplicaciones móviles.

Perspectivas: La pirámide de pruebas móviles

Después de haber explorado varias herramientas de pruebas móviles automatizadas a lo largo de este capítulo, tenemos que ver cómo se unen y si pueden ajustarse a la pirámide de pruebas habitual. Discutámoslo rápidamente.

Como ya hemos dicho, la arquitectura básica de una aplicación móvil es similar a la de una aplicación web: la aplicación llama a servicios, que a su vez hablan con la BD. Por tanto, las pruebas de las capas inferiores seguirán ciñéndose a la pirámide de pruebas. Como recordarás, la pirámide de pruebas recomienda tener una capa gruesa de pruebas de micronivel y una capa fina de pruebas de macronivel. Sin embargo, ha habido debates sobre si es posible alcanzar la forma piramidal cuando se trata de la capa móvil. Algunos dicen que, en su experiencia, la pirámide de pruebas móviles adopta una forma invertida, con una capa gorda de pruebas de interfaz de usuario con amplias pruebas manuales y una capa

fina de pruebas unitarias. Sus experiencias varían principalmente por dos factores: el alcance restringido de las pruebas unitarias y de interfaz de usuario en la capa móvil y el contexto de la aplicación.

Para profundizar un poco más, cuando se trata de pruebas unitarias en la capa móvil, lo habitual es que validen pequeñas partes de la funcionalidad proporcionada por una clase o método. Sin embargo, para las funcionalidades que dependen de las API del SO, puede que no se escriban pruebas unitarias para validar el comportamiento de la API, ya que se entiende que el proveedor del SO habrá probado la API. Esto impone la necesidad de realizar pruebas de extremo a extremo, ya sean manuales o automatizadas, para garantizar que las API del SO funcionan como se espera dentro del contexto de la aplicación en diferentes tipos de dispositivos. Además, las características específicas del hardware, como las integraciones con la cámara, los sensores, etc., y las relacionadas con la usabilidad, como la facilidad de desplazamiento, los gestos avanzados, etc., no pueden probarse a fondo ni como parte de las pruebas unitarias ni de las de interfaz de usuario. Estos aspectos obligan a realizar pruebas manuales.

Dado este alcance restringido de las pruebas unitarias y de interfaz de usuario, la forma de la pirámide de pruebas de la aplicación móvil viene dictada por las características de la aplicación. Para las aplicaciones que contienen una lógica funcional extensa y tienen pocas dependencias de factores externos, como el hardware del dispositivo y las API del sistema operativo, la pirámide de pruebas móviles seguirá pareciéndose a la pirámide de pruebas tradicional, ya que habrá más pruebas de nivel inferior que cubran la lógica funcional. Por otro lado, para las aplicaciones que tienen una lógica funcional limitada pero grandes dependencias de factores externos, puede que tengas una pirámide de pruebas invertida. En estos casos, tendrás que planificar una capacidad de pruebas adicional y esforzarte por lograr un equilibrio entre escribir más pruebas de

interfaz de usuario automatizadas y realizar pruebas de regresión manuales exhaustivas.

Puntos clave

Éstos son los puntos clave de este capítulo:

- Con la creciente dependencia de las personas de la esfera móvil y los beneficios que aporta a las empresas, podemos esperar ser testigos de una tendencia al alza continuada en el desarrollo de aplicaciones móviles en los próximos años. Así que, como desarrolladores y probadores de software, debemos prepararnos adquiriendo las habilidades pertinentes para las pruebas móviles.
- Las pruebas móviles difieren de las pruebas web en muchos aspectos. Este capítulo ha introducido los matices del panorama móvil viéndolo a través de tres lentes diferentes, centrándose en consideraciones relacionadas con los dispositivos, las aplicaciones y la red. La diversidad en cada una de estas áreas plantea un reto importante a los equipos de desarrollo de software en todas las fases del ciclo de vida del desarrollo, incluidos el diseño, el desarrollo y las pruebas.
- Tu estrategia de pruebas de aplicaciones móviles debe incorporar los fundamentos de las pruebas de pila completa, como una gran atención a las pruebas a nivel micro y macro, prácticas de pruebas de desplazamiento a la izquierda y pruebas de varias dimensiones de calidad, como la seguridad, el rendimiento y la accesibilidad.
- Los ejercicios y el debate sobre herramientas de prueba adicionales proporcionaron una visión detallada de una variedad de herramientas funcionales y multifuncionales, manuales y automatizadas, que pueden utilizarse para las pruebas móviles.

- La forma de la pirámide de pruebas móviles a veces se invierte, dependiendo de las características y funciones de la aplicación. Este es un aspecto importante de las pruebas móviles que debe tenerse en cuenta desde el principio para poder planificar la capacidad de pruebas adecuada .

¹ También puedes ver las denominaciones LDPI, MDPI, HDPI, XHDPI, XXHDPI y XXXHDPI, donde DPI se refiere a puntos por pulgada, una forma de medir la densidad de píxeles.

Capítulo 12. Ir más allá en las pruebas

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

Los profesionales siguen las instrucciones; los expertos entienden los principios!

Hasta ahora, hemos hablado de todas las habilidades de comprobación que debe poseer un profesional del software para ofrecer con éxito aplicaciones web y móviles de alta calidad. Hemos establecido que las pruebas son un espacio amplio y creciente que ha evolucionado durante décadas para incluir nuevos procesos, herramientas y metodologías. Aunque hoy en día hay 10 lentes diferentes a través de las cuales se pueden ver las habilidades de comprobación (las 10 habilidades de comprobación de pila completa descritas en los capítulos anteriores), mañana podría haber más. Sin embargo, incluso en un entorno tan dinámico, los principios fundamentales de las pruebas permanecerán inalterados, independientemente de la tecnología o el dominio. Comprender estos primeros principios de las pruebas te proporcionará el marco y los conocimientos que necesitas para tener éxito, independientemente de cómo siga creciendo el espacio de las pruebas en el futuro.

En este capítulo, ofreceré una breve visión general de estos primeros principios en las pruebas y sus ventajas fundamentales, y echaré un vistazo a cómo han evolucionado las herramientas y las prácticas de equipo existentes basándose en estos principios. También exploraremos cómo las habilidades sociales de una persona se

suman a sus habilidades técnicas para contribuir al éxito general de su equipo en la entrega de software de alta calidad.

Primeros principios en las pruebas

La Figura 12-1 muestra los siete primeros principios de las pruebas. En los siguientes subapartados, profundizaremos en cada uno de ellos.

Defect prevention over defect detection

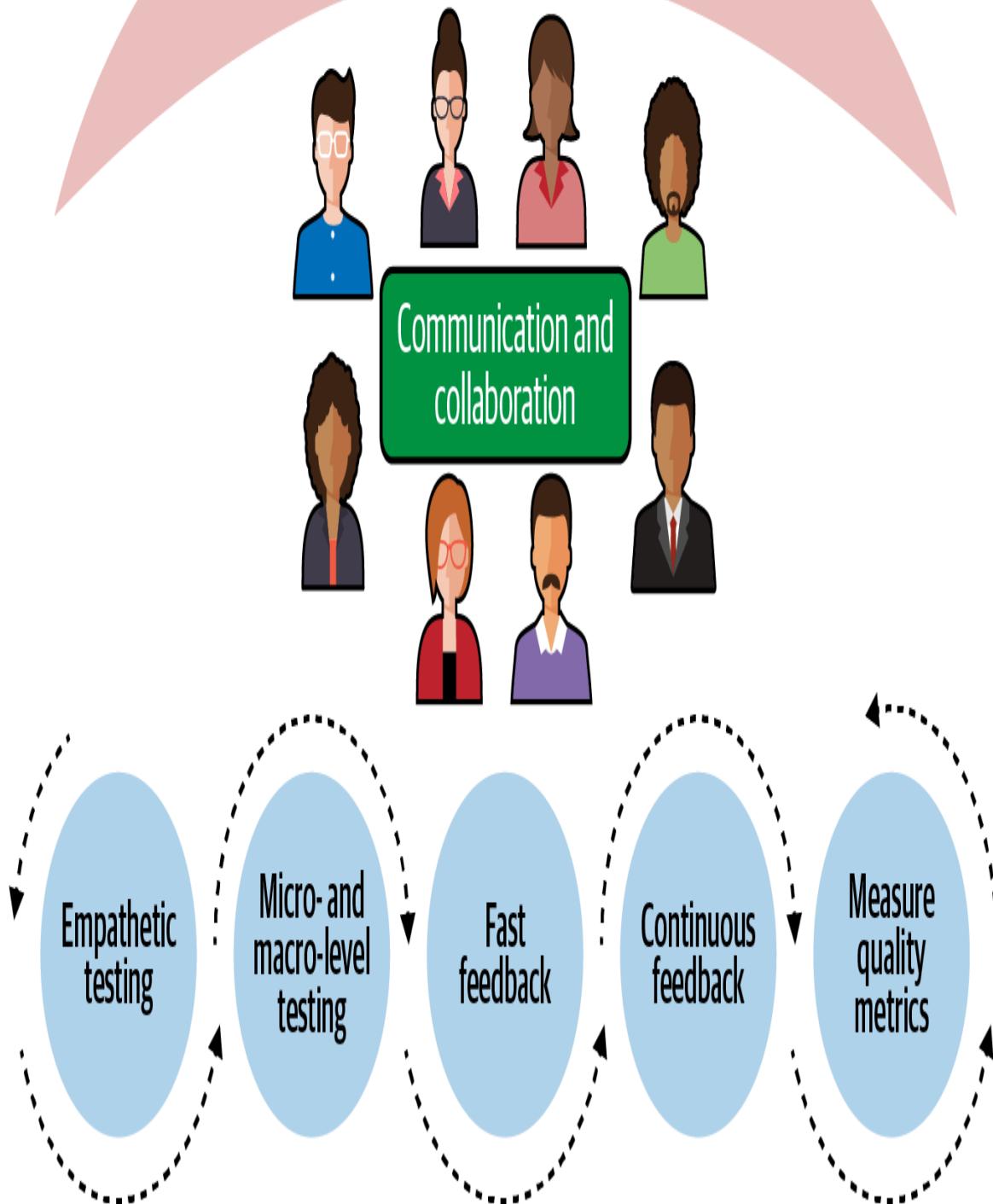


Figura 12-1. Primeros principios en las pruebas

Prevención de defectos frente a detección de defectos

Aunque el objetivo principal de las pruebas es encontrar problemas en la aplicación, la prevención de defectos debe considerarse el núcleo de las pruebas de . Una razón obvia por la que deberíamos evitar los defectos es el coste de arreglarlos. Podemos comparar estas correcciones con el enlucido de una grieta y los retoques de pintura en una pared que, por lo demás, está perfectamente pintada: a veces, el parche recién pintado no queda bien, y tenemos que volver a pintar toda la pared! Del mismo modo, los defectos del software pueden dar lugar a cambios importantes en la arquitectura, que exigen una enorme cantidad de reelaboración y costes. De ahí que este principio básico sugiera adoptar prácticas, herramientas y métodos que permitan evitar que se produzcan defectos en primer lugar, en lugar de detectarlos y repararlos.

Algunas prácticas del mundo del software actual que pretenden cumplir este principio son:

- Reuniones de planificación de la iteración (IPM), que son realizadas al principio de una iteración o sprint para debatir en detalle las historias de usuario. Una MIP es un espacio abierto para que los equipos aporten ideas sobre las integraciones que faltan y los casos de perímetro en las historias de usuario.
- El **proceso de los tres amigos**, en el que los representantes de la empresa, los desarrolladores y los probadores de estudian a fondo cada función durante la fase de análisis. El objetivo del proceso es recoger las perspectivas de las tres partes para que no se pasen por alto las integraciones, los casos extremos y otros requisitos empresariales.

- Del mismo modo, el kickoff de la historia de usuario pretende repetir el proceso de los tres amigos justo antes de que comience el desarrollo de la historia de usuario. También es una práctica habitual en las pruebas por turnos que los probadores capturen y discutan los casos de prueba durante los kickoffs de las historias, por las mismas razones.
- **Los registros de decisiones de arquitectura (ADR)** y las estrategias de prueba se debaten y documentan para dirigir al equipo hacia los objetivos colectivos de calidad del proyecto.
- Durante el desarrollo, el desarrollo dirigido por pruebas (TDD) activa los procesos de reflexión en torno a los casos de perímetro, incluso para pequeños fragmentos de código.
- La programación en parejas es otra práctica de desarrollo destinada a evitar que un código defectuoso provoque defectos y a evitar que se pierdan casos de perímetro.
- Del mismo modo, el software de linting detecta los olores de codificación incluso mientras los desarrolladores los están creando.

Como puedes ver, hay muchas prácticas que se centran en la prevención de defectos. Este enfoque puede aplicarse a cualquier dominio nuevo, como los datos, incluso cuando los roles en el equipo son diferentes.

Pruebas empáticas

Probar consiste en ponerse en el papel del usuario final, al fin y al cabo. Cuando llevamos puesto el sombrero de las pruebas, debemos actuar pensando en los intereses del usuario final, superando las distracciones cotidianas de las necesidades críticas de la empresa y los detalles técnicos de la implementación. No podemos limitarnos a verificar los criterios de aceptación de la historia de usuario y seguir adelante; tenemos que explorar la aplicación de la forma en que la

consumirá un usuario final típico. En consecuencia, comprender y empatizar con los distintos usuarios finales a los que se dirige una aplicación es vital antes de empezar a probarla. A menudo, los equipos hacen concesiones sobre las necesidades del usuario final basándose en factores como la complejidad del desarrollo y los plazos. Sin embargo, al desempeñar el papel de probador, debemos poner en primer plano la perspectiva del usuario final y negociar las compensaciones teniendo en cuenta sus intereses. Aunque trabajemos con nuestros equipos día tras día, al realizar las pruebas debemos anteponer a nuestros usuarios finales.

Pruebas a micro y macrónivel

Como se explicó en el Capítulo 1, las pruebas deben realizarse tanto a nivel micro como macro para ofrecer un software de alta calidad. Recordemos que las pruebas a nivel micro implican profundizar en una pequeña parte de la funcionalidad y probarla en detalle; por ejemplo, probar el cálculo total del pedido con varias condiciones límite (precios negativos, decimales largos, etc.). Las pruebas de macrónivel utilizan una lente más amplia para abarcar los flujos funcionales, la propagación de datos entre módulos, las integraciones entre componentes, etc. Por ejemplo, las pruebas a nivel macro podrían centrarse en probar el flujo de creación de pedidos, las integraciones de terceros, los flujos de la interfaz de usuario, los fallos en la creación de pedidos, etc.

En el Capítulo 3 vimos una elaborada estrategia de pruebas funcionales automatizadas con varios tipos de pruebas de micronivel y macrónivel. En resumen, las pruebas unitarias, de integración y contractuales se centran en las pruebas de micronivel, mientras que las pruebas de API, las pruebas funcionales de interfaz de usuario, las pruebas visuales, etc., se centran en las pruebas de macrónivel. También hemos hablado de cómo un desequilibrio en la distribución de las pruebas de micro y macrónivel puede retrasar la retroalimentación, lo que compromete la calidad.

Otra consecuencia crítica de tal desequilibrio será la detección de problemas imprevistos en producción. Esto se debe a que, en general, los equipos que sólo se centran en las pruebas a nivel macro pasan por alto los detalles. Por ejemplo, habrán probado escenarios de nivel macro, como que el pedido se cree con éxito y que se produzca un fallo en la creación del pedido debido a la falta de disponibilidad del artículo. Pero cuando los precios de los artículos en producción son negativos o tienen un número inesperado de decimales, la creación del pedido puede fallar, provocando defectos. Por lo tanto, es esencial ampliar y reducir constantemente para asegurarte de que obtienes tanto la visión general como los detalles mientras realizas las pruebas.

Respuesta rápida

Este principio consiste en encontrar los defectos pronto para que el ciclo de corrección de defectos y, en consecuencia, el ciclo de lanzamiento sea más rápido. Existe una notable correlación entre el tiempo que se tarda en solucionar un defecto y lo tarde que se encuentra. Cuando una función está en desarrollo, el desarrollador tiene todo el contexto necesario sobre el código y puede comprender fácilmente las causas de los fallos y solucionarlos rápidamente. Pero cuando el desarrollador pasa a otras funciones y la base de código sigue creciendo cada día con la refactorización, ese contexto se pierde, y depurar la causa raíz se convierte en un proceso más largo y costoso.

Además, el ciclo de seguimiento de defectos contribuye significativamente al retraso en el ciclo de corrección de defectos. Por ejemplo, imagina un fallo de alta prioridad encontrado dos semanas después del desarrollo de la característica. Habrá que dedicar tiempo y esfuerzo a crear tarjetas de errores, clasificarlos, seguirlos en iteraciones y encontrar al desarrollador adecuado con tiempo para solucionarlos. Estas tareas pueden llevar días, si no semanas! Y después de todos estos retrasos, en el peor de los casos

puede que te encuentres con que es imposible arreglar el defecto sin una refactorización importante debido al desarrollo que ha tenido lugar en el periodo intermedio, retrasando así el calendario de lanzamiento. Este es el coste final que hay que pagar por un defecto, y también la razón principal por la que debes centrarte en crear ciclos de retroalimentación más rápidos en tus equipos.

Entonces, ¿con qué antelación debe probarse un fragmento de código para obtener una respuesta rápida? Las pruebas de cambio a la izquierda tienen que ver con una retroalimentación más rápida, y hemos visto cómo ponerla en práctica en cada uno de los capítulos anteriores. Para recordar algunas de las prácticas de equipo que proporcionan una retroalimentación más rápida comentadas en los [capítulos 2 a 4](#), puedes poner en práctica las pruebas dev-box (ejecutar pruebas automatizadas tan pronto como en la máquina del desarrollador) y la pirámide de pruebas. Además, las firmas de las historias de usuario por parte de los propietarios del producto (PO) o los representantes de la empresa y las presentaciones a todas las partes interesadas al finalizar cada sprint conseguirán que el equipo obtenga una respuesta más rápida sobre los casos empresariales que faltan.

En pocas palabras, hacer pruebas para obtener una respuesta más rápida equivale a cosechar en el momento adecuado. Cuando el momento se retrasa, tienes que conformarte con una cosecha de menor calidad.

Retroalimentación continua

Una retroalimentación rápida debe estar respaldada por una retroalimentación continua . No basta con probar una función una vez y dejarla inactiva hasta el lanzamiento. Tienes que seguir realizando pruebas de regresión para obtener información sobre si las funciones actuales y sus integraciones siguen intactas mientras el equipo continúa desarrollando nuevas funciones y refactorizando el

código existente. Estos mecanismos de retroalimentación continua ayudan a detectar los problemas cuando aún son relativamente pequeños, lo que evita la interrupción de los plazos de lanzamiento. La retroalimentación continua también sitúa al equipo en el punto óptimo para poder realizar una entrega continua.

Como se ha comentado en [el Capítulo 4](#), la forma predominante de obtener una retroalimentación continua es aplicar prácticas de pruebas continuas. Para destacar un par de puntos clave sobre las pruebas continuas, ejecuta todas las pruebas funcionales a nivel micro y macro y las pruebas CFR para cada commit como parte de tu canalización CI. Esto proporcionará información continua sobre todas las dimensiones de la calidad y, por tanto, preparará al equipo para la entrega continua.

Medir las métricas de calidad

¡Todo lo que se mide tiende a mejorar! El propósito de tener KPI en cualquier campo es hacer un seguimiento de estos indicadores y mejorarlo iterativamente dando los pasos adecuados. Así pues, cuando intentamos conseguir resultados de alta calidad, también debemos medir la calidad. Dicho esto, cuando se pone un énfasis desproporcionado en las métricas, los miembros del equipo tienden a encontrar formas de engañar a las métricas y olvidan su finalidad última. Así pues, las métricas deben desplegarse sabiamente para dirigir al equipo hacia objetivos comunes de calidad.

He aquí algunas métricas de calidad que beneficiarán al equipo cuando se controlen con regularidad:

Defectos detectados por pruebas automatizadas en todas las capas

Las pruebas automatizadas crean una red de seguridad para el equipo, y cuando la mayoría de los defectos se detectan en las primeras fases, los equipos se sienten más seguros a la hora de hacer nuevos cambios. Esta métrica también refleja la fuerza de la red de seguridad.

Tiempo desde el compromiso hasta la Implementación

Como hemos visto antes, una retroalimentación más rápida es fundamental para progresar. Cuando el desarrollador realiza un commit, las pruebas automatizadas del canal de CI deben comprobar inmediatamente los nuevos cambios e implementarlos en el entorno de control de calidad para poner en marcha las pruebas exploratorias. He visto equipos en los que las canalizaciones CI tardan mucho tiempo en generar una compilación verde debido a la inestabilidad de las pruebas y a problemas del entorno, lo que retrasa la retroalimentación y provoca una pérdida de productividad.

Número de Implementaciones automatizadas en entornos de prueba

Esta métrica y la anterior mostrarán la rapidez y el éxito con que el equipo es capaz de realizar nuevos cambios. Lo ideal es que el equipo cuente con una buena red de seguridad que permita implementaciones rápidas y estables. Si observas que el número de Implementaciones automatizadas en los entornos de prueba es bajo, debido a fallos de la infraestructura, de las pruebas o de otro tipo, es señal de que tu ciclo de retroalimentación necesita mejoras.

Defectos de regresión detectados durante las pruebas de historias de usuario

Los defectos de regresión detectados durante la fase de prueba de historias de usuario indican que faltan casos de uso empresariales o que faltan pruebas automatizadas. Por ejemplo, las pruebas automatizadas en CI pasarán por alto una refactorización de una consulta SQL para utilizar equals en lugar de like si los datos de entrada a la prueba se diseñaron para que coincidieran con ambos. Como comentamos en [el Capítulo 3](#), cuando se detectan defectos de regresión durante las pruebas de historias de usuario, puede ser síntoma de que los equipos

siguen antipatrones en las pruebas automatizadas. De ahí que deban reflexionar inmediatamente sobre las causas de tales defectos y mejorar sus procesos con regularidad.

Cobertura de automatización basada en la gravedad de los casos de prueba

Lleva un registro detallado de tu cobertura de automatización con el objetivo de no tener ningún retraso. Llevar un registro de esto te ayudará a planificar tus iteraciones con antelación para cubrir el backlog, si lo hubiera.

Defectos de producción y su gravedad

El seguimiento de los defectos de producción te muestra un panorama más amplio de los casos de uso empresariales que faltan, la configuración que falta, las discordancias de datos y cualquier otro problema que el equipo pueda haber pasado por alto. Identifica sus causas profundas y automatiza las pruebas en torno a ellas. Además, construye una estrategia de pruebas viva y sigue evolucionándola a medida que la aplicación y el equipo se adentren en nuevos terrenos.

Puntuaciones de usabilidad con usuarios finales

Recoge las opiniones de los usuarios finales sobre la experiencia general del usuario durante la propia fase de desarrollo. Esto te ayudará a mejorar las métricas relacionadas con el diseño de la UX (por ejemplo, número máximo de clics para obtener información, texto frente a iconos, etc.).

Fallos debidos a problemas de infraestructura

Haz un seguimiento de los problemas de infraestructura, como la caída intermitente de los servicios en los entornos de prueba, problemas en los conductos de CI, desajustes en las configuraciones de los entornos de prueba y desarrollo, etc. A

veces, el código de la infraestructura puede necesitar que se pague la deuda técnica para que sea escalable y estable.

Métricas en torno a aspectos interfuncionales

Mide sistemáticamente los KPI de rendimiento y muestra los resultados a tus equipos. Incluye estadísticas sobre los casos de pruebas de seguridad automatizadas y las vulnerabilidades encontradas durante las exploraciones automatizadas como parte de tus vitrinas de iteración. Del mismo modo, incluye la cobertura de automatización de los casos de prueba de CFR específicos de tu proyecto y presenta las métricas pertinentes (cobertura de pruebas entre navegadores, resultados de ingeniería del caos, cobertura de pruebas de localización, etc.).

Muchas de las métricas mencionadas aquí están relacionadas con las cuatro métricas clave tratadas en [el Capítulo 4](#), que miden la calidad en términos de estabilidad del código y ritmo de entrega del equipo. Por ejemplo, recuerda que una de las cuatro métricas clave, *el tiempo de entrega* (el tiempo que transcurre desde que se confirma el código hasta que está listo para su implementación en producción), se espera que sea inferior a un día para un equipo de élite. Cuando hay una buena red de seguridad de cobertura de automatización, el equipo puede hacer cambios tan rápidos con confianza.

Del mismo modo, la métrica de *frecuencia de implementación* debe ser "a la carta" para un equipo de élite. Cuando medimos el tiempo transcurrido desde el commit hasta la implementación y el número de implementaciones en un día en entornos de prueba, nos hacemos una idea del ritmo de entrega del equipo. Los defectos de producción nos informarán sobre el *porcentaje de cambios fallidos* (porcentaje de cambios lanzados a producción que fallan), que debería ser del 0-15% para un equipo de alto rendimiento. Cuando se realiza un seguimiento de estas métricas y se debate sobre ellas

de forma coherente, el equipo persigue de forma inherente el objetivo de un software de alta calidad.

La comunicación y la colaboración son la clave de la calidad

Las pruebas no pueden realizarse como una actividad aislada. Para que las pruebas añadan valor, debe haber una comunicación adecuada sobre los requisitos empresariales, el conocimiento del dominio, la implementación técnica, los detalles del entorno, etc. Esto requiere una colaboración e interacción constantes entre todas las funciones de un equipo de proyecto. La comunicación puede realizarse a través de ceremonias ágiles como las reuniones, los kickoffs de historias, los IPM, las pruebas de caja de desarrollo y la documentación adecuada, como tarjetas de historias, ADR, estrategias de pruebas, informes de cobertura de pruebas y similares. Aunque no podemos esperar que la comunicación sea sincrónica en el mundo actual, con equipos distribuidos que trabajan en zonas horarias diferentes, debemos trabajar para garantizar que los traspasos se realicen sin problemas mediante la documentación adecuada y medios asincrónicos como grabaciones de vídeo y correos electrónicos.

En resumen, seguir estos siete primeros principios guiará a los equipos de software en el desarrollo de estrategias de pruebas eficaces, incluso cuando se aventuren en nuevas áreas del espacio tecnológico. Yo mismo he aplicado estos principios en proyectos con nuevas pilas tecnológicas y en dominios desconocidos, y he visto que producen sistemáticamente resultados de alta calidad.

Las habilidades interpersonales ayudan a crear una mentalidad de calidad ante todo

Llegados a este punto, es esencial destacar una vez más en que varios aspectos del desarrollo de software -diseño, análisis, desarrollo, infraestructura, etc.- contribuyen a producir software de alta calidad. La comprobación de la calidad es uno de esos aspectos, y además crítico. Por lo tanto, es necesario que todos los miembros del equipo trabajen juntos para conseguir una alta calidad. Ninguna persona puede ser completamente dueña de la calidad, y tampoco puede *no* serlo ninguna; no es muy distinto de cómo un equipo de relevos no puede ganar una carrera aunque un corredor vaya lento. Y las habilidades interpersonales desempeñan un papel crucial en la creación de una mentalidad de calidad en el equipo. Si eres probador de profesión o responsable de las pruebas en el trabajo, aquí tienes una lista de aptitudes interpersonales sobre las que me gustaría arrojar luz y que te ayudarán a crear en tu equipo una mentalidad colaborativa que dé prioridad a la calidad:

Capacidad para impulsar resultados

Con cada función del equipo centrada en para obtener resultados de calidad, se establece colectivamente para producir resultados de alta calidad. Por ejemplo, el diseño de un viaje intuitivo para el usuario es responsabilidad de la función de UX, la visión de un producto fácil de usar para el cliente es responsabilidad del PO/representante comercial, y garantizar una buena arquitectura y un código robusto es responsabilidad de los desarrolladores. En la misma línea, los probadores deben apropiarse principalmente de las actividades relacionadas con las pruebas e impulsar al equipo a incorporarlas a sus prácticas cotidianas. Por ejemplo, son responsables de garantizar que el equipo adopte prácticas y herramientas de prevención de defectos, corroborar que se sigan las prácticas de pruebas continuas, hacer un seguimiento de la cobertura de automatización y conseguir que se complete como

parte de cada historia de usuario, y otras prácticas descritas a lo largo del libro.

Colaboración

Inculcar la mentalidad de que la calidad es responsabilidad del equipo sólo puede conseguirse mediante una fuerte colaboración con todos los miembros del equipo y los clientes o partes interesadas de la empresa. Si somos rígidos en nuestras ideas y apáticos a la hora de llegar a otros miembros del equipo, no conseguiremos resultados de alta calidad. Por ejemplo, apropiarse de la estrategia de pruebas en colaboración con los desarrolladores contribuirá en gran medida a alcanzar este objetivo, del mismo modo que colaborar con los representantes empresariales para descubrir los casos de prueba que faltan ayudará significativamente a prevenir defectos.

Comunicación eficaz

A veces, la forma en que nos comunicamos marca la diferencia entre que una tarea se complete con éxito o no. Una comunicación eficaz también implica elegir un medio adecuado y el momento oportuno para comunicarse. En particular, debe haber una comunicación regular y clara de los probadores al equipo sobre la calidad general del producto y lo que se necesita para alcanzar el nivel de calidad deseado.

Priorización

Las pruebas pueden convertirse en una actividad interminable si no se priorizan eficazmente. A veces, lo que parece una pequeña tarea en desde el punto de vista del desarrollo exige un esfuerzo de pruebas desproporcionado y no planificado, lo que desordena los calendarios. Para evitar estas situaciones, los probadores deben priorizar la lista de actividades de prueba por historia de usuario con bastante antelación y asegurarse de que los esfuerzos que requerirán se acomodan a la capacidad de la

iteración. Esto allanará el camino para que el equipo entregue con éxito las características sin sacrificar la calidad.

Gestión de las partes interesadas

Las partes interesadas de un proyecto son los clientes, los directores, los compañeros de equipo, los jefes técnicos y cualquier otra persona que pueda cambiar el curso de acción requerido por . Tenemos que gestionar las expectativas de las partes interesadas sobre la calidad de forma coherente. Los clientes pueden esperar que la cobertura de automatización sea del 100%, lo que puede no ser un objetivo realista, y los directores pueden estar más interesados en cumplir los plazos de lanzamiento que en la calidad. Gestionar y ayudar a dar forma a estas expectativas desde el principio mediante la colaboración, la comunicación eficaz y el establecimiento de prioridades conducirá al éxito colectivo.

Coaching/tutoría

La incorporación de nuevos miembros es habitual en los equipos de , y no podemos esperar que los recién llegados conozcan todas las prácticas y herramientas del equipo desde el principio. Sin embargo, de acuerdo con la idea de que la calidad es responsabilidad del equipo, todos y cada uno de los miembros del equipo deben estar de acuerdo con las prácticas y herramientas de comprobación. Por lo tanto, en nuestra calidad de probadores (junto con todos los demás roles) deberíamos formar parejas con los nuevos miembros del equipo para compartir nuestros conocimientos sobre estos temas y ayudarles a avanzar rápidamente.

Además, ten en cuenta que la tutoría o el coaching es una actividad que va más allá de la incorporación inicial al proyecto. Debe dar lugar a un aprendizaje y mejora continuos para el mentor/coachee, especialmente en la mejora de sus habilidades

interpersonales, para que puedan actuar como campeones de calidad en el equipo.

Influencia

La influencia es importante, sobre todo cuando se trabaja con equipos grandes y clientes nuevos. Sin ella, aunque establezcamos una sabia estrategia de pruebas, puede que no se aplique de forma generalizada como nos gustaría. La influencia es clave para conseguir apoyo para la estrategia de pruebas, y para convencer a las partes interesadas del negocio de que inviertan en nuevas herramientas y prácticas de pruebas. Por supuesto, no hay una receta fija para crear influencia, pero ser capaz de producir resultados de alta calidad de forma coherente, junto con mostrar las seis aptitudes interpersonales anteriormente mencionadas, debería contribuir en gran medida a este objetivo.

Las habilidades interpersonales pueden ser más difíciles de dominar que las técnicas, y requieren una práctica diligente día tras día. Pero a medida que trabajes para dominarlas, puede que descubras para tu sorpresa que ya eres bastante bueno en algunas de ellas, y si las utilizas adecuadamente descubrirás que son muy beneficiosas para tu éxito colectivo y el de tu equipo .

Conclusión

Hemos llegado al final de una extensa exploración de las habilidades de comprobación necesarias para ofrecer aplicaciones web y móviles de alta calidad. En este punto, es mi responsabilidad señalar que las pruebas son un viaje de aprendizaje continuo. A medida que practiques activamente todo lo que hemos tratado aquí, seguirás adquiriendo más conocimientos. Además, como señalé al principio, el testing es un campo en rápida evolución, en el que aparecen continuamente nuevas herramientas, procesos y buenas prácticas.

Este rápido crecimiento puede parecer a veces abrumador. Si es así, da un paso atrás y recuerda que todas estas novedades atienden fundamentalmente a uno de los primeros principios, y aprender cómo y dónde encajan es sólo un pequeño paso. Al final, una simple mezcla de las habilidades de comprobación de la pila completa con tus habilidades interpersonales te pondrá en el buen camino para entregar eficazmente software de alta calidad.

Con esto, también hemos llegado al final del libro. Hay un capítulo extra después de éste, en el que se tratan cuatro tecnologías emergentes y algunos de los aspectos de las pruebas que les son propios. Pretende ser una lectura rápida y ágil, con la intención de que el lector piense más allá del ámbito de las aplicaciones web y móviles.

Mientras decides si te aventuras por ahí, me gustaría darte las gracias por recorrer este largo camino conmigo. Demuestra tu compromiso con la entrega de software de alta calidad, lo cual es realmente encomiable! Espero que el libro te haya orientado eficazmente en el aprendizaje de nuevas habilidades de comprobación y haya arrojado luz sobre prácticas de comprobación contemporáneas que puedas poner en práctica en tu trabajo. Hasta que nos encontraremos de nuevo en nuestro viaje de pruebas, te deseo lo mejor, y gracias por darme la oportunidad de viajar contigo a través de este libro! :)

Capítulo 13. Introducción a las pruebas en tecnologías emergentes

Este trabajo se ha traducido utilizando IA. Agradecemos tus opiniones y comentarios: translation-feedback@oreilly.com

Los rápidos cambios de la tecnología pueden ser estimulantes y vertiginosos al mismo tiempo.

La tecnología ha dado pasos de gigante en la última década. Muchas de las cosas que veíamos de niños en las películas de ciencia ficción están hoy ante nosotros: drones de vigilancia, inicios de sesión mediante huella dactilar, asistentes inteligentes, videojuegos totalmente inmersivos, y la lista continúa. Oímos muchas palabras de moda: IA, ML, IA centrada en el ser humano, blockchain, RA, RV, RM, bots, iy mucho más! Es un reto incluso asimilarlas todas a la vez. Una forma de asimilar esta vasta difusión de tecnologías es agruparlas en temas, como los siguientes:

Interacciones similares a las humanas

Durante mucho tiempo, todo lo que teníamos para interactuar con los ordenadores era un ratón y un teclado. En el mundo actual, estas interacciones se han ampliado para incluir el tacto, la voz, los gestos y mucho más. Fitbit y Alexa están aquí, interactuando con nosotros y, más concretamente, ihablando con nosotros!

Inteligencia aumentada

La tecnología se utiliza para aumentar la inteligencia humana , haciéndonos la vida mucho más fácil. Los asistentes inteligentes, las recomendaciones personalizadas y los chatbots son algunos ejemplos de cómo la tecnología ha cambiado irrevocablemente nuestras vidas.

Plataformas como normas

La tendencia actual en tecnología es que los datos, los servicios, la infraestructura y demás se abstraigan para formar **plataformas** tecnológicas con fines de reutilización y escalabilidad. Esto permite la innovación continua de nuevos productos en consonancia con las necesidades del mercado. Las llamadas **super apps** como Uber, WeChat, Grab y Gojek utilizan plataformas como base.

Cosas conectadas

Dejemos de pensar en los seres humanos por un momento. Ahora *las cosas* también están conectadas a través de Internet. Vivimos en un mundo en el que nuestros teléfonos, relojes y cafeteras hablan entre sí.

El podcast ***Seismic Shifts*** y el informe ***Looking Glass*** de Thoughtworks presentan panorámicas detalladas de los avances tecnológicos, por si quieres profundizar más.

Aunque muchas de estas tecnologías aún no se han generalizado y, por tanto, no es imprescindible tener conocimientos para probarlas, es conveniente estar preparado antes de que llegue la ola. Este capítulo pretende ofrecer una breve introducción a cuatro tecnologías emergentes -AI/ML, RA/VR, blockchain e IoT- y discutir los aspectos de las pruebas relacionados con cada una de ellas. Como puede resultar obvio, cada uno de estos temas merece un libro en sí mismo, y este capítulo sólo pretende ofrecer una visión preliminar de lo que son estas tecnologías y hacia dónde se dirigen.

Inteligencia Artificial y Aprendizaje Automático

La inteligencia artificial (IA) es un subcampo de la informática que pretende utilizar máquinas para realizar tareas que normalmente realizan los humanos, simulando la inteligencia humana. *La IA fuerte*, en particular, es una construcción teórica que puede hacer cualquier cosa que pueda hacer un humano. La IA se ejerce mediante el aprendizaje automático (AM), otro subcampo de la informática basado en la idea de que los ordenadores pueden programarse para aprender de la experiencia en lugar de programarse explícitamente para realizar una tarea de una manera determinada.

Los términos IA y ML se utilizan a menudo indistintamente. Para hacer una distinción, cualquier programa que muestre un comportamiento similar al humano puede llamarse IA, pero a menos que sus comportamientos se aprendan automáticamente de la experiencia -es decir, de datos históricos- no es aprendizaje automático. Esto quedará mucho más claro en breve, cuando hablemos del enfoque de programación del aprendizaje automático.

Introducción al Aprendizaje Automático

Normalmente, cuando desarrollamos una aplicación, codificamos en una secuencia de instrucciones para que el ordenador las ejecute; al menos, así es como hemos sabido que funcionan los ordenadores hasta ahora. Pero oír que los ordenadores pueden aprender de su experiencia sin ser programados explícitamente es sumamente intrigante. Para desmitificar lo que esto significa, consideremos un ejemplo: el filtro de contenido abusivo de una aplicación de redes sociales. Esto nos ayudará a comprender con más precisión la diferencia entre la programación tradicional y los enfoques de aprendizaje automático.

Para construir un filtro de contenido abusivo a la manera de la programación convencional, empezaríamos por enumerar los

criterios que identifican el contenido como abusivo, los codificaríamos como reglas y eliminaríamos las entradas que los activen. Por ejemplo, podríamos escribir código para buscar una lista de palabras clave como *suicidio*, *sexo*, *advertencia de activación*, etc. Del mismo modo, podríamos buscar identificadores de usuario de explotadores conocidos, marcar el contenido como abusivo y omitir los feeds automáticamente.

Pero, ¿es suficiente? Cuando codificamos una regla para marcar una lista de palabras como abusivas, los abusadores introducen rápidamente nuevas palabras para saltársela. Del mismo modo, cuando se restringen las cuentas existentes, los usuarios abusivos crean nuevas cuentas desde las que enviar contenidos. En un espacio problemático como éste, en el que las propias reglas son no deterministas, escribir una solución infalible utilizando el enfoque de programación tradicional es todo un reto. Aquí es donde el aprendizaje automático echa una mano.

Con el enfoque de programación ML, como se ve en la Figura 13-1, introducimos una enorme cantidad de datos históricos etiquetados como abusivos o no abusivos en un modelo de aprendizaje automático. Esto se llama *entrenar* el modelo. El modelo es fundamentalmente un algoritmo matemático, y aprende las diferencias entre los dos tipos de contenido a partir de los datos. Esto, en cierto modo, es similar a cómo aprende el cerebro humano. A lo largo de los años, vemos muchas manzanas de distintos tamaños, formas y colores desde distintos ángulos, y nos volvemos expertos en distinguir una manzana. Del mismo modo, también aprendemos a distinguir entre una manzana y una naranja.

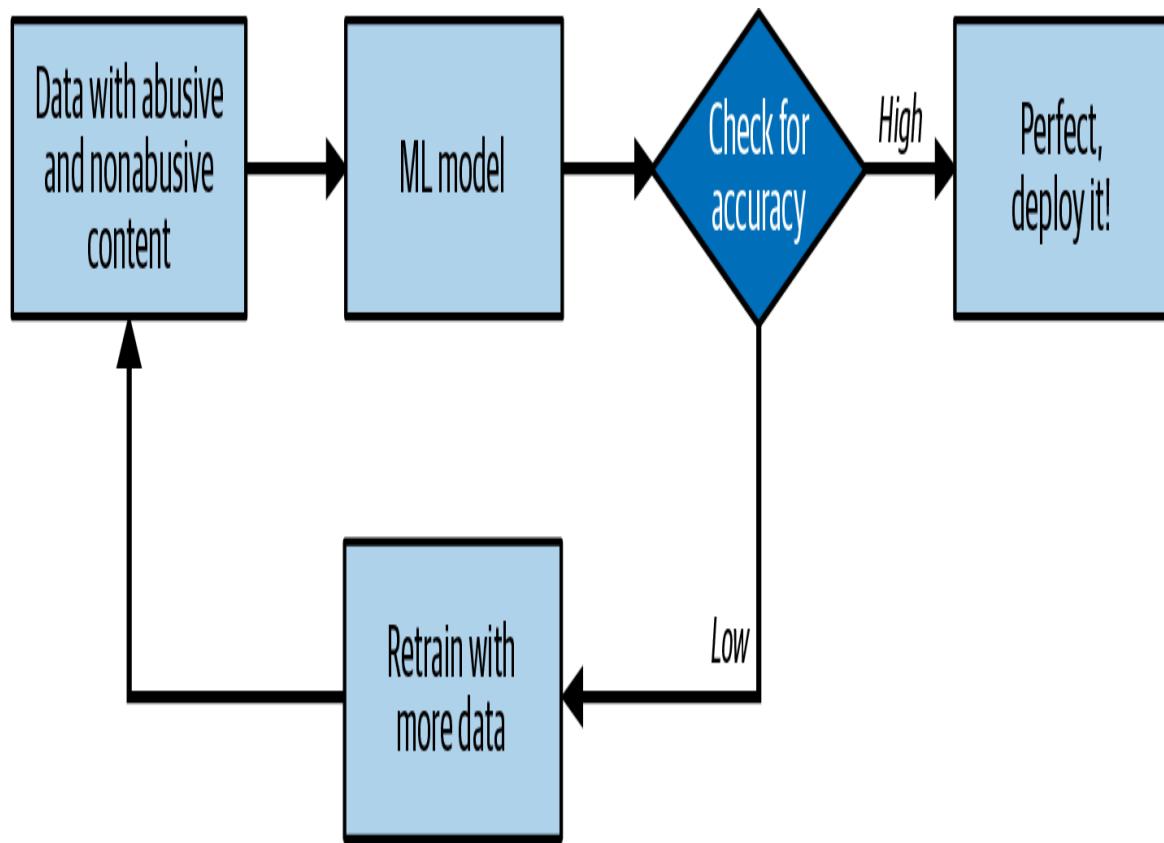


Figura 13-1. La forma ML de programar un filtro de contenidos abusivos

Una vez que el modelo está entrenado, puede decirnos si una nueva publicación es abusiva o no. Puede que no acierte todas las respuestas al principio, igual que no lo haría un niño humano. Tenemos que entrenarlo continuamente con un conjunto de datos más diverso y seguir evaluando la precisión del modelo mediante pruebas con datos sin etiquetar. Los datos sin etiquetar utilizados para comprobar la precisión del modelo se denominan *conjunto de pruebas*, y los datos utilizados para entrenarlo se denominan *conjunto de entrenamiento*. Cuando la precisión del modelo es lo suficientemente alta, se implementa en producción. El modelo también se entrena continuamente con nuevos contenidos de producción para garantizar que pueda captar nuevas palabras y variaciones.

NOTA

El aprendizaje automático con datos etiquetados se denomina aprendizaje *supervisado*. Los algoritmos de ML también pueden entrenarse con datos no etiquetados, en cuyo caso intentan aprender patrones automáticamente a partir de los datos que se les presentan. Este tipo de aprendizaje se denomina aprendizaje *no supervisado*.

En resumen, el flujo de trabajo de la programación de ML comienza con la recopilación de muchos datos, su etiquetado adecuado, su división en un conjunto de entrenamiento y un conjunto de prueba, el uso del conjunto de entrenamiento para entrenar el modelo de ML, la evaluación de la eficacia del modelo con el conjunto de prueba, la implementación y la continuación del entrenamiento. Algunos marcos populares de aprendizaje automático que ayudan en estas tareas son scikit-learn, PyTorch y TensorFlow. El aprendizaje automático ha encontrado aplicaciones en áreas como la medicina, la banca, las redes sociales, etc., y también se está explorando continuamente en nuevos dominios. A continuación abordaremos los aspectos de las pruebas.

Probar aplicaciones de ML

La mayoría de las aplicaciones ML adoptan una arquitectura típica basada en servicios, con el componente ML integrado en los servicios. En el ejemplo del filtro de contenidos, un flujo basado en servicios podría ser el siguiente: cuando un usuario crea un nuevo post, la IU lo envía a un servicio de contenidos para que compruebe con el modelo si es abusivo o no. Si el modelo identifica que el contenido es abusivo, el servicio de contenidos ordena a la IU que oculte el contenido. Así pues, junto con el enfoque habitual para probar una arquitectura orientada a servicios típica, deberíamos incluir los siguientes aspectos para cubrir las pruebas de toda la aplicación:

Validar los datos de entrenamiento

Los datos que se introducen en el modelo dictan en gran medida la calidad del modelo. Si la calidad de los datos es mala, la calidad del modelo será mala. Por tanto, centrarse en la calidad de los datos de entrada es fundamental para las aplicaciones de ML. Como necesitamos una gran cantidad de datos para entrenar el modelo, pueden obtenerse de diversas fuentes, como bases de datos públicas, raspado de sitios web públicos, entradas de usuarios de diferentes sitios web, e incluso registros del sistema. Por lo general, esto nos deja con datos de diversas formas y figuras: básicamente, un desorden desordenado y caótico. En nuestro ejemplo, nuestra fuente de datos eran las publicaciones históricas en las redes sociales. Además de texto, estas publicaciones podían contener imágenes, videos, GIFs, comentarios, etiquetas, etcétera. Algunas de ellas podrían tener a su vez diferentes tamaños, formatos de archivo, gradientes de color, etc. Si alimentamos el modelo con datos tan incoherentes, le resultará difícil centrarse en las características del contenido que lo hacen abusivo, como las palabras clave, y aprender las distinciones con precisión.

Por eso, la práctica habitual consiste en limpiar los datos de entrada, eliminar el ruido, transformarlos en un formato normalizado y, a continuación, alimentar el modelo para su entrenamiento. Esta lógica de limpieza y transformación debe probarse a fondo. Un par de casos de prueba básicos para dar una idea de cómo podría funcionar esto podrían ser:

- Cuando los datos de entrada vienen con diferentes escalas - por ejemplo, los datos numéricos podrían ir desde valores decimales a números exponencialmente grandes-, debe probarse la lógica para limpiar los datos y transformarlos a una escala uniforme.

- Cuando los datos de entrada pueden contener valores nulos o vacíos, deben sustituirse por valores por defecto o eliminarse durante la etapa de limpieza.

En general, los datos tienen muchos aspectos específicos del dominio que deben comprobarse explícitamente. Por ejemplo, las publicaciones en redes sociales pueden tener un límite de caracteres establecido, que debe validarse al comprobar la calidad de los datos de entrada. Normalmente, los equipos también escriben pruebas unitarias para la lógica de limpieza y transformación de para automatizar algunos de estos casos de prueba.

Validar la calidad del modelo

La calidad del modelo se mide en términos de varias métricas, como las tasas de error, la exactitud, las matrices de confusión, la precisión y el recuerdo. Existen métodos para calcular cada una de ellas. En nuestro ejemplo podríamos utilizar la precisión y la recuperación, como se describe aquí:

- *La precisión*, como su nombre indica, se refiere a la capacidad del modelo para predecir correctamente un resultado (es decir, el número de verdaderos positivos sobre el número total de verdaderos y falsos positivos). Por ejemplo, si el modelo identifica 100 mensajes como abusivos y 99 son realmente abusivos, su índice de precisión es de 0,99.
- Por otra parte, *la recuperación* es la métrica que nos indica cuántos de los mensajes abusivos reales fueron identificados correctamente por el modelo (es decir, el número de verdaderos positivos del número total de verdaderos positivos y falsos negativos). Si el modelo identificó correctamente como abusivas 99 entradas de un total de 110 entradas abusivas, su índice de recuerdo es de 0,90.

Los marcos de trabajo de ML mencionados anteriormente tienen funciones integradas para calcular este tipo de métricas, y podemos escribir pruebas para que fallen en la canalización de CI basándonos en estas métricas cada vez que se registra un nuevo modelo. [MLflow](#) es una herramienta de código abierto que puedes utilizar para ver el rendimiento del modelo para cada versión del modelo.

Validar el sesgo del modelo

Una cosa es tratar con datos de mala calidad , pero el sesgo en el modelo lo empeora. Recientemente, el algoritmo ML de recorte de imágenes de Twitter se enfrentó a [críticas públicas](#) porque prefería las caras de los individuos blancos a las de los negros al recortar, lo que llevó a la empresa a abandonar el enfoque de recorte automático. Estos sesgos se filtran al modelo desde los datos de entrada. Si los datos de entrada tienen un gran conjunto de muestras que representan una demografía concreta, el modelo estará sesgado hacia esa demografía. Por lo tanto, es fundamental comprobar si hay sesgos tanto en los datos de entrada como en el modelo ML. [Facets](#) es una herramienta de código abierto que puede ayudarnos con esto, al permitirnos visualizar patrones en los datos de entrada.

Validar las integraciones

Las integraciones entre las tres capas de -específicamente las capas de datos y modelo y las capas de modelo y API- deben probarse utilizando los enfoques habituales de pruebas de contrato e integración.

Centrarnos en estos aspectos debería permitirnos hacer también entrega continua. Algunos de mis colegas tratan en detalle la disciplina de la Entrega Continua para el Aprendizaje Automático (CD4ML) en su [artículo](#) en el sitio web de Martin Fowler.

Blockchain

Sir John Hargrave y Evan Karnoupakis dan una sencilla definición de blockchain en una sola línea en su informe *Qué es Blockchain*: "Blockchain es el *Internet del Dinero*". Si en consideramos que el dinero es cualquier cosa de valor, como acciones, bonos, puntos de recompensa, etc., e Internet es una plataforma para compartir información libremente con nuestros iguales, entonces blockchain puede entenderse como una plataforma para compartir cualquier cosa de valor.

El nombre deriva de su funcionamiento. Cada vez que se realiza una transacción (intercambio de valor), se crea un bloque con los datos de esa transacción, encadenado a la transacción anterior. Por "encadenado" me refiero a que cada bloque tiene un hash del contenido del bloque anterior, creando una cadena de bloques, como se ve en [la Figura 13-2](#).

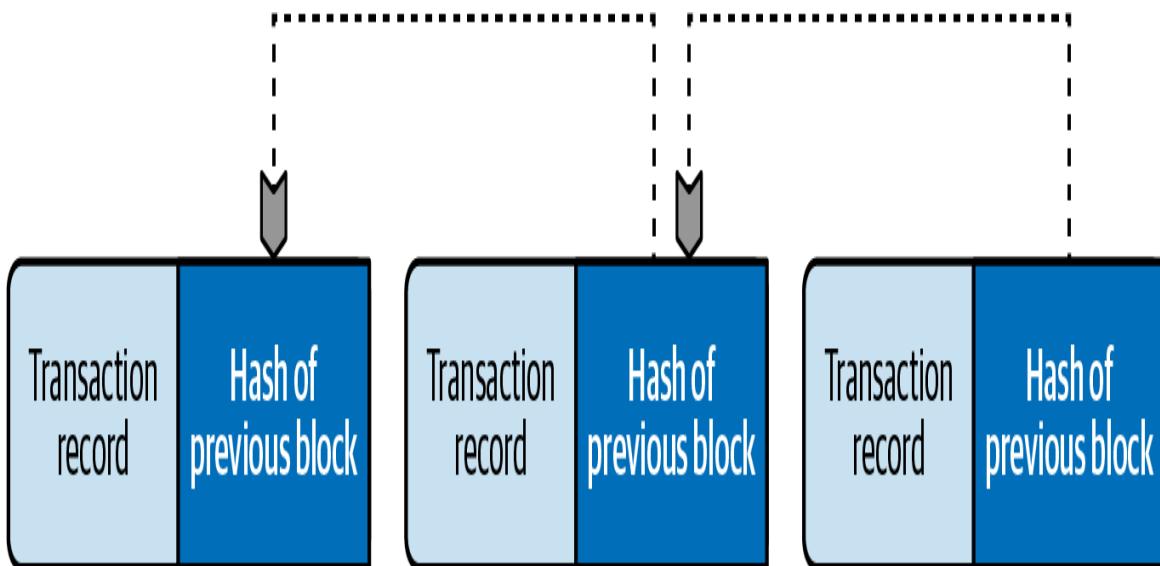


Figura 13-2. Una cadena de bloques con datos de transacciones

Así es como la cadena de bloques aporta seguridad. Si alguien altera el contenido de un bloque, el hash del siguiente bloque no coincidirá y, por tanto, se romperá la cadena. Por tanto, podemos decir que las transacciones son inmutables: se pueden añadir nuevos bloques a la

cadena, pero los bloques existentes nunca pueden alterarse. Normalmente, para el hash se utilizan algoritmos criptográficos de alto nivel, como SHA-256, que hacen que la cadena de bloques sea impenetrable para los piratas informáticos.

¿Cuál era la filosofía que había detrás de la creación de un sistema tan impenetrable? En 2008, un autor anónimo que escribía bajo el seudónimo de Satoshi Nakamoto publicó un libro blanco, "[Bitcoin: A Peer-to-Peer Electronic Cash System](#)", que hablaba de un nuevo concepto llamado *dinero digital*, o *e-cash*, que podía transferirse entre partes sin la participación de un agente centralizado, como un banco, en medio. La idea era sencilla: la gente trabaja duro para ganar dinero, y debería tener control sobre él: idinero, para la gente y por la gente! La emocionada comunidad de desarrolladores se puso rápidamente a trabajar en la aplicación del libro blanco, que evolucionó hasta convertirse en la tecnología blockchain actual. Para llamar tu atención sobre un par de puntos clave, blockchain evolucionó para ayudar a la descentralización y promover las transacciones entre iguales. Se requería que la seguridad formara parte de ella, ya que la tecnología pretendía tratar con dinero.

Introducción a los conceptos de Blockchain

Ahora, hablaremos de los bloques de construcción de blockchain para hacernos una idea de cómo se pueden implementar las pruebas:

Libros de contabilidad descentralizados

Un *libro de contabilidad* es un repositorio que contiene todos los datos contables (las entradas y salidas de una transacción).

Blockchain utiliza libros de contabilidad descentralizados; es decir, el libro de contabilidad no es propiedad de una persona, sino de todos los participantes. Cualquier parte que pretenda realizar una transacción obtendrá una copia del libro mayor. La ventaja es que es fiable, ya que ninguna persona puede manipular los registros.

Sin embargo, esto conlleva el coste adicional de mantener todos los libros de contabilidad sincronizados en todo momento.

Nodos

Un nodo es cualquier ordenador o servidor que participa en la red blockchain. Los nodos pueden pertenecer a un único individuo o a un grupo de individuos. Cada nodo almacena una copia del libro mayor descentralizado. Cuando se produce una nueva transacción, cada uno de ellos actualiza su copia de la cadena de bloques. Como se ve en [la Figura 13-3](#), los nodos se comunican entre sí para mantener sincronizados los libros de contabilidad. Este proceso se basa en algo llamado *tecnología de libro mayor distribuido (DLT)*.

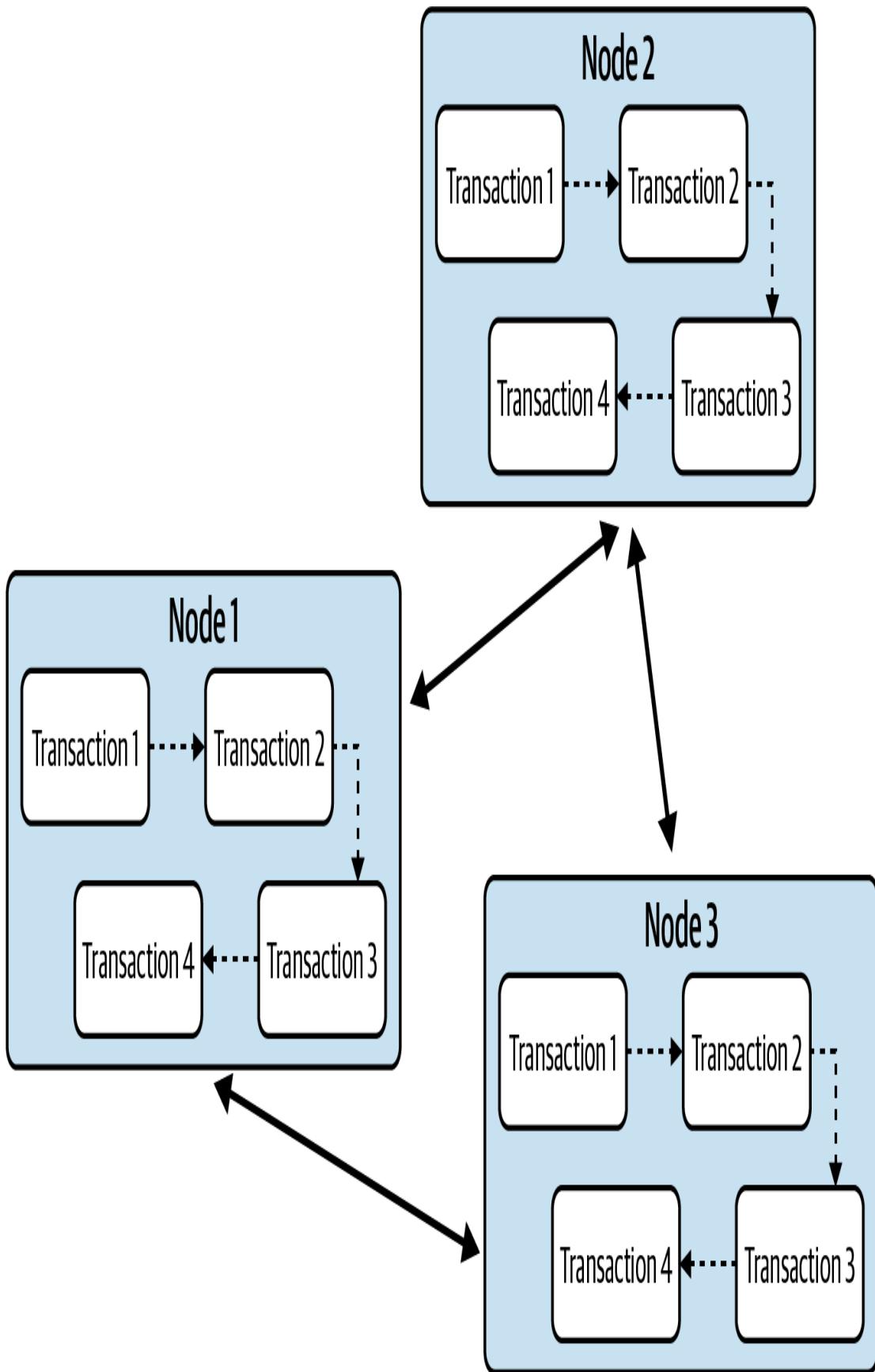


Figura 13-3. Tecnología de libro mayor distribuido con nodos que poseen cada uno una copia separada de la cadena de bloques.

Consenso

En blockchain, tenemos libros de contabilidad descentralizados con los datos contables y nodos que proporcionan la infraestructura necesaria para mantenerlos. Un banco es una autoridad centralizada que puede añadir o eliminar las transacciones de los clientes tras verificar su integridad, pero en blockchain todos los nodos son participantes en pie de igualdad: ¿quién puede añadir nuevas transacciones a la cadena? Aquí es donde entra en juego *el consenso*.

El consenso es el proceso por el cual los nodos acuerdan colectivamente añadir una transacción. Para conseguirlo programáticamente, tenemos algoritmos de consenso como *Proof of Work* y *Proof of Stake*. En el algoritmo de Prueba de Trabajo, se da a los nodos un problema matemático extremadamente complejo para que lo resuelvan. El primer nodo que obtiene la respuesta correcta recibe la autoridad para añadir el nuevo bloque. Los demás nodos de la red verifican la integridad del nuevo bloque antes de añadirlo. Cuando un nodo añade un nuevo bloque, se le recompensa con moneda digital (proceso denominado *minería*). El inconveniente de este algoritmo es la necesidad de grandes cantidades de potencia de cálculo para resolver los complejos problemas matemáticos. Con el algoritmo Proof of Stake, los nodos tienen un poder de minería proporcional a la cantidad de moneda digital que controlan. El inconveniente aquí es que los nodos con las mayores apuestas siguen teniendo el privilegio y se enriquecen.

Contratos inteligentes

Un sistema bancario tiene un conjunto de normas y condiciones establecidas para ejecutar una transacción con éxito. Por ejemplo, antes de aprobar un préstamo para vivienda, el banco

verifica primero tu salario, el saldo de la cuenta, los documentos de la vivienda, etc. En blockchain, la lógica necesaria para completar una transacción se escribe como un *contrato inteligente*. Cada nodo obtiene también una copia del contrato inteligente. Las ventajas de este enfoque son que permite realizar transacciones sin papel, elimina las comisiones de los intermediarios y facilita que las partes respectivas completen una transacción de forma independiente.

Éstos son los componentes básicos de la cadena de bloques. Para unirlo todo y hacernos una idea del flujo de trabajo general, veamos un ejemplo. Supongamos que Alicia quiere comprar unos tomates a Bob por 10 Ethereum (una criptomoneda popular). Alice inicia una transacción y transfiere el dinero. El contrato inteligente retiene el dinero de Alice hasta que Bob entrega los tomates. Como prueba de la entrega, Bob puede producir un código QR para que sea escaneado. Cuando Alice escanea el código QR, la transacción se completa y el contrato inteligente transfiere el dinero a Bob. Si Bob no entrega los tomates, el contrato inteligente devuelve el dinero a Alice tras un periodo determinado. Mientras tanto, los nodos de la red compiten para resolver el problema matemático y obtener el derecho a añadir un nuevo bloque para esta transacción. El nodo ganador también reúne los registros de transacciones del contrato inteligente para añadirlos al bloque. Una vez añadido el bloque, se sincroniza con todos los demás nodos.

Los marcos de desarrollo de blockchain que permiten todo esto incluyen Ethereum, HyperLedger Fabric y Stellar. OpenZeppelin y Solidity se utilizan para escribir contratos inteligentes. MetaMask es un monedero para guardar moneda digital (concretamente, Ethereum).

Prueba de aplicaciones Blockchain

Una vez discutido el flujo de trabajo general en la tecnología blockchain, he aquí algunas áreas de interés para las pruebas:

Pruebas funcionales

El primer paso al probar cualquier aplicación de es validar los flujos funcionales de extremo a extremo, como en el ejemplo de la compra de tomates. La lógica funcional está escrita en los contratos inteligentes, así que tenemos que buscar lagunas en ella. Los casos de prueba para validar los contratos inteligentes también pueden añadirse como pruebas unitarias.

Pruebas API

Lo más habitual es que haya APIs en la parte superior de la cadena de bloques, conectadas al frontend. Deberíamos centrarnos en las pruebas estándar de la capa API: funcionalidades, integraciones entre módulos, versionado de contratos, gestión de errores, reintentos, etc.

Pruebas de seguridad

Aquí hay muchos aspectos de seguridad implicados que hay que probar, desde la creación de cuentas y los mecanismos de autorización para las personas implicadas en una transacción hasta el intercambio de monedas, el mantenimiento del saldo de la cuenta, la comprobación de transacciones ilegítimas y los aspectos criptográficos como el hashing de los bloques.

Pruebas de rendimiento

Dado que las transacciones dependen de la disponibilidad de los nodos y de los algoritmos de consenso, el tiempo que se tarda en completar una transacción puede ser mayor que con una tecnología web estándar. Por tanto, hay que probar el

rendimiento de las transacciones y el comportamiento funcional para gestionar los retrasos.

PRUEBAS ESPECÍFICAS PARA BLOCKCHAIN

La mayoría de las aplicaciones utilizan las redes existentes de cadenas de bloques , como Ethereum, para la implementación de sus contratos inteligentes, por lo que es posible que no necesites probar las características de la cadena de bloques en particular. Sin embargo, si esto llegara a ser necesario, algunos aspectos que debes tener en cuenta son:

Adición de transacciones

Cada transacción debe registrarse sin pérdida de información. Éste es el requisito más crítico de la cadena de bloques. Los bloques deben estar correctamente encadenados y sincronizados con todos los demás nodos.

Tamaño del bloque

Las transacciones se agrupan en el mismo bloque hasta que el tamaño del bloque alcanza un límite superior (en la red Bitcoin, por ejemplo, el tamaño original del bloque era de 1 MB). Tenemos que comprobar si se crea un nuevo bloque cuando el tamaño del bloque alcanza su límite.

Tamaño de la cadena

A medida que crece el número de transacciones, la cadena se hace muy grande. Necesitamos comprobar el rendimiento de la aplicación con tamaños de cadena tan grandes.

Prueba de nodos

Los nodos son bloques fundamentales de la cadena de bloques. Los nodos deben poder participar en el consenso y

estar sincronizados con los datos más recientes en todo momento. Los nuevos nodos deben poder unirse a la red sin problemas.

Resiliencia

Cuando los nodos vuelvan a estar disponibles tras una breve interrupción, deben poder integrarse de nuevo en la red sin problemas y sin interrumpir la funcionalidad de la aplicación. Si no hay nodos disponibles durante un tiempo, la aplicación debe gestionar la interrupción con elegancia.

Colisiones

Puede haber situaciones en las que más de un nodo haya resuelto el problema matemático y luchen por el derecho a añadir una nueva transacción. Tenemos que comprobar si se dan estas situaciones de colisión.

Corrupción de datos

Un **nodo bizantino** es un nodo que se comporta mal en un sistema descentralizado. Cuando eso ocurre, los datos de los nodos pueden corromperse en . Existen formas probadas de manejar este tipo de situaciones, y es necesario probar ese comportamiento.

Herramientas como **Ethereum Tester** y **Populus** son útiles para probar aplicaciones blockchain basadas en Ethereum, y **bitcoinj** y **testnet** ayudan a probar las transacciones de Bitcoin .

Como puedes ver, la tecnología blockchain tiene importantes ventajas en términos de fuerte seguridad, transacciones totalmente digitalizadas, eliminación de intermediarios y lucha contra el monopolio. Sin embargo, también existen algunas desventajas que dificultan la adopción de esta tecnología. Por ejemplo, blockchain requiere grandes cantidades de potencia informática y eléctrica para resolver los complejos problemas matemáticos y sincronizar todos los datos del libro mayor, y debido a los algoritmos de consenso y a la disponibilidad intermitente de los nodos, puede llevar mucho tiempo completar una transacción. Según los informes, Visa gestiona unas **1.700 transacciones por segundo**, mientras que una sola transacción de blockchain puede tardar 10 minutos en confirmarse. En consecuencia, el rendimiento es un cuello de botella importante.

Internet de las cosas

El Internet de las Cosas (IoT) es la tecnología que conecta el mundo físico con el mundo digital. Permite que los dispositivos ("cosas") que nos rodean adquieran inteligencia y empiecen a comunicarse entre sí y con nosotros a través de Internet. Esta inteligencia también permite a los dispositivos reaccionar de forma autónoma a los cambios de su entorno sin intervención humana. Por ejemplo, los termostatos inteligentes se adaptan a las condiciones atmosféricas, como la humedad, y fijan la temperatura adecuada en función de las preferencias del usuario. El IoT ha demostrado ser una solución para necesidades tanto a pequeña como a gran escala. Un ejemplo famoso en el sector doméstico es la solución del hogar inteligente; se espera que el valor global del mercado del hogar inteligente supere **los 53.000 millones de dólares** en 2022. En el otro extremo del espectro están las **soluciones de IoT para ciudades inteligentes**, que se esfuerzan por mejorar la calidad de vida general de los residentes mediante la mejora de las infraestructuras, la calidad del aire, el transporte, el consumo de energía, etc.

Los dispositivos IoT suelen estar provistos de tres características: un sensor, un actuador y un medio de comunicación. Los sensores detectan estados físicos, como la temperatura, la frecuencia del pulso, el movimiento, etc. El actuador activa cambios en el entorno actual, como activar una alarma cuando se detecta humo o abrir y cerrar válvulas para controlar la temperatura. Los medios de comunicación, como las pantallas digitales y la voz, ayudan a los dispositivos IoT a presentar información al usuario.

Construir una solución IoT de extremo a extremo requiere conocimientos tanto de hardware como de software. Un componente de software está integrado en el hardware para controlar sus funcionalidades y transmitir información a los usuarios. Otro componente de software reside fuera del hardware, y agrega y analiza los datos enviados desde múltiples dispositivos para emprender acciones colectivas. Por ejemplo, para leer el pulso del usuario mediante un dispositivo de fitness, el software dentro del dispositivo activa el sensor del hardware para medir el recuento y transmitirlo a la pantalla digital. El software también envía esta información a la nube, donde un servicio realiza un análisis de patrones sobre la información de la frecuencia del pulso, los ciclos de sueño, etc., e indica al software integrado que emita una alarma cuando se detecten anomalías.

Para que todas estas tecnologías funcionen de forma cohesionada - sensores, redes, protocolos de comunicación y enrutamiento, procesadores de datos, aplicaciones de usuario final, la nube, etc.- se requiere mucha integración de extremo a extremo. Un examen más detallado de la arquitectura de cinco capas del IoT te ayudará a comprender mejor estas integraciones.

Introducción a la arquitectura de cinco capas del IoT

Existen diversos puntos de vista sobre la definición del número de capas de una arquitectura IoT: tres, **cuatro o cinco**. La arquitectura de cinco capas, como se ve en **la Figura 13-4**, ofrece una visión más amplia y profunda de las tecnologías implicadas en la construcción de una aplicación IoT de extremo a extremo. Examinaremos brevemente cada una de estas capas para comprender los aspectos de las pruebas en ellas.

Business layer

Analytics technologies like Apache Spark, Apache Kafka, Sensor ML

Application layer

App development technologies including web, mobile, and other devices

Middleware layer

Service discovery and data exchange technologies like MQTT, cloud, mDNS

Network layer

Networking technologies like IPv6, Zigbee, NFC

Perception layer

Physical devices that collect data like QR code readers, RFID scanners, wearables

Figura 13-4. Arquitectura IoT de cinco capas

Veamos brevemente cada una de estas capas para comprender los aspectos de las pruebas que intervienen en cada una de ellas:

Capa de percepción

Es la capa inferior, donde el hardware lee información del mundo físico y la transfiere a las capas siguientes. El hardware puede clasificarse como pasivo, semipasivo o activo, según admita la comunicación unidireccional o bidireccional. Por ejemplo, los escáneres de códigos QR entran en la categoría pasiva, ya que sólo pueden comunicarse en un sentido, y el alcance de la comunicación es limitado. Esto es suficiente para escenarios como el seguimiento de envíos. Además, ten en cuenta que los componentes pasivos no tienen la capacidad de realizar cálculos. Los componentes activos pueden recibir y transmitir datos, y están dotados de la capacidad energética necesaria. Algunos ejemplos son los actuadores inteligentes que realizan tareas mecánicas, los wearables con sensores integrados, las radios GPS, etc. También pueden comunicarse a mayor distancia.

Capa de red

Los dispositivos físicos tienen que identificarse en Internet para que otros dispositivos puedan comunicarse con ellos. IPv4 e IPv6 son protocolos de red populares que proporcionan direcciones IP únicas a los dispositivos (se prefiere IPv6). Para transferir eficazmente los datos a la dirección de destino, los dispositivos utilizan protocolos de encaminamiento como el Protocolo de encaminamiento para redes de baja potencia y con pérdidas (RPL). Utilizan tecnologías de comunicación estándar como WiFi, Zigbee, comunicación de campo cercano (NFC) y Bluetooth para transmitir y recibir información.

Capa de middleware

Las aplicaciones IoT deben poder acceder a los dispositivos físicos utilizando sus nombres o direcciones para solicitar sus

servicios (como leer la temperatura de la habitación, la frecuencia cardíaca del usuario, etc.) sin conocer los detalles de la infraestructura subyacente. La capa de middleware ayuda a realizar ese descubrimiento de servicios. También se encarga de extraer datos de los dispositivos físicos y comunicar la información a los usuarios. Éste es el núcleo de la solución IoT. El descubrimiento de servicios protocolos como Avahi y Bonjour y mecanismos de intercambio de datos como el Protocolo de Aplicación Restringida (CoAP) y el Transporte de Telemetría de Colas de Mensajes (MQTT) son ampliamente utilizados.

Capa de aplicación

Esta capa permite a los usuarios finales acceder a los servicios deseados de a través de una interfaz sencilla, como una web o una aplicación móvil, sin saber cómo se procesan las solicitudes de servicio en las capas subyacentes. La capa de aplicación incluye la lógica para agregar, procesar y almacenar información de múltiples dispositivos.

Capa empresarial

Esta capa analiza la información recopilada del hardware, los servicios, etc. para mejorar los servicios de la aplicación. Se utilizan tecnologías de big data como Apache Spark y Apache Kafka para analizar las enormes cantidades de datos recibidos de los diferentes dispositivos IoT. Esta capa está destinada principalmente a los administrativos internos y no a los usuarios finales.

Las plataformas IoT como AWS IoT e IBM Watson combinan muchas de estas capacidades para facilitar el desarrollo de IoT.

Prueba de aplicaciones IoT

Algunos aspectos específicos en los que hay que centrarse al probar las soluciones IoT son los siguientes:

Integración hardware/software

La funcionalidad integral de cualquier aplicación IoT depende principalmente de una integración adecuada de hardware y software, y que debe probarse con diversos casos de perímetro. Por ejemplo, la aplicación de monitoreo de los latidos del corazón en un reloj inteligente tiene que mostrar el recuento correcto de latidos del corazón registrados por el sensor, y cuando hay problemas en el registro de los latidos del corazón, el software debe manejar los errores adecuadamente. Estos casos de prueba de integración deben probarse después de nuevas instalaciones y actualizaciones de hardware o software. Además, hay que tener en cuenta las limitaciones habituales del hardware en cuanto a memoria y batería al probar las características funcionales.

Red

La conectividad de red entre dispositivos y con la nube es un aspecto importante de las soluciones IoT que debe probarse adecuadamente. Algunos dispositivos pueden admitir varios protocolos de comunicación, como WiFi y Bluetooth, y esas capacidades deben probarse independientemente.

Interoperabilidad

Interoperabilidad en una solución IoT se refiere a la capacidad de los distintos dispositivos para intercambiar información entre sí, aunque sigan normas y protocolos diferentes. Por ejemplo, en una solución IoT de transporte inteligente, los dispositivos sensoriales de tráfico, los servicios de detección de accidentes y los sistemas de encaminamiento automático deben poder intercambiar información sin problemas, aunque cada uno pueda

funcionar individualmente con diferentes conjuntos de tecnologías y protocolos. La interoperabilidad libera realmente el potencial de la IoT, pero las integraciones deben probarse cuidadosamente.

Seguridad y privacidad

Algunos protocolos de comunicación, como Z-Wave, pueden no ser siempre seguros, por lo que es necesario emplear mecanismos de seguridad ligeros adicionales, como IPsec, para evitar ataques. Además, los datos recogidos y almacenados en la nube deben ser privados por diseño. No sólo no es ético almacenar los datos biométricos y otros datos privados de las personas sin su consentimiento, sino que (como vimos en el [Capítulo 10](#)) existen requisitos legales relativos al almacenamiento de información personal, y tenemos que comprobar su cumplimiento.

Rendimiento

El rendimiento es un aspecto importante de la calidad en las soluciones IoT, ya que puede haber muchos dispositivos hablando entre sí y transmitiendo información a los servicios agregadores. Por tanto, necesitamos respuestas a preguntas como con qué rapidez responde el hardware a las órdenes del software, cuál es el tiempo de respuesta global de un servicio (como obtener la frecuencia del pulso) y cómo es el rendimiento de la recogida de datos cuando hay muchos dispositivos en la red (como en una ciudad inteligente).

Usabilidad

La usabilidad es fundamental, especialmente en el sector doméstico, por ejemplo, con dispositivos como smartwatches y televisores inteligentes. Con estos dispositivos, puede haber muchos aspectos que probar. Los smartwatches responden a los movimientos de la muñeca, tienen pantallas de distintos

tamaños, pueden manejarse con distintos botones y gestos, tienen sistemas de alerta consistentes en vibraciones y notificaciones sonoras, pueden llevarse en la muñeca derecha o en la izquierda, y mucho más. La familiarización de los usuarios con las capacidades de un dispositivo también es una parte esencial del producto. Probar el conjunto de aspectos de usabilidad es fundamental para el éxito del producto.

Por mi propia experiencia de trabajo en una cafetera inteligente, puedo atestiguar que probar soluciones IoT es increíblemente complejo debido a las variadas combinaciones de dispositivos y sus estados internos. Para ayudar a gestionar la multitud de estados y combinaciones de dispositivos y derivar casos de prueba, formulé un marco de pruebas llamado **Atlas de Pruebas IoT**, ique quizá te interese explorar más a fondo!

Realidad Aumentada y Realidad Virtual

Larealidad aumentada (RA) es una tecnología que superpone gráficos, textos, imágenes y otra información sensorial al entorno del mundo real y la presenta a los usuarios para mejorar su experiencia global. Inicialmente se inventó para ayudar a los cazas de aviones a reacción; los pilotos tenían que atacar objetivos con precisión mientras volaban, y la RA presentaba los detalles necesarios en sus pantallas frontales para ayudarles a concentrarse en ambas tareas simultáneamente. Uno de los últimos ejemplos de RA es la pantalla frontal (HUD) desarrollada por Mercedes, que proyecta información como mapas y límites de velocidad aceptables en el parabrisas del vehículo.

Hoy en día, tenemos juegos que utilizan gafas inteligentes, HUD y varios dispositivos móviles y portátiles de RA para ofrecernos la experiencia de la RA. Es posible que hayas oído hablar o probado pantallas inteligentes portátiles de fabricantes como Google, Vuzix,

Epson y Nreal. Sin embargo, los teléfonos inteligentes con RA son los más cercanos a nosotros. Tanto Android como iOS están equipados con las herramientas y marcos necesarios, como ARCore, ARKit y Unity's AR Foundation, para habilitar esta tecnología, y hay muchos teléfonos con hardware compatible con RA (como el Pixel 5, el Nokia 8, el Moto G, etc.).

Mientras que la RA aumenta tu entorno real, *la realidad virtual* (RV) transporta al usuario a un mundo virtual simulado. Además de aplicaciones populares como los juegos, esta tecnología es beneficiosa para simular entornos peligrosos, como incendios o ataques aéreos, y entrenar a los combatientes para enfrentarse a ellos. La RV también ha ganado popularidad en el espacio comercial, donde se ofrecen a los clientes funciones de personalización de productos, como diseñar los interiores de su nueva casa, y experiencias de productos en tiempo real, como probadores virtuales.

Las experiencias de RV requieren dispositivos de visualización montados en la cabeza (HMD) para una experiencia totalmente inmersiva. Algunos de los más populares del mercado son Oculus Quest, Oculus Go, HTC VIVE y Sony PlayStation VR. Una vez más, los teléfonos inteligentes con soluciones como Google Cardboard ofrecen a una opción más accesible y económica.

Además de la RA y la RV, también tenemos *la realidad mixta* (RM), que es una combinación de RA y RV que permite a los usuarios interactuar con contenidos digitales en 3D. El juego **Pokémon Go** es un ejemplo de RM. Del mismo modo, *la Realidad Extendida* (XR) es cuando los dispositivos de RA, RV y RM se integran con otros dispositivos, como electrodomésticos, sensores, etc. El espacio de la RA, la RV, la RM y la RX está en expansión, y sin duda hay que prestarle atención.

Probar aplicaciones AR/VR

La tecnología de RA y RV es fascinante y ofrece a los usuarios una experiencia estimulante, pero también es increíblemente compleja. Desarrollar y probar estos productos requiere experiencia en una amplia gama de áreas, desde la biología (percepción humana de las imágenes, mecánica de la formación de imágenes por el ojo, percepción de la profundidad, etc.) hasta las matemáticas espaciales, las tecnologías de pantallas montadas en la cabeza y mucho más. Existen plataformas de desarrollo como Unity que han abstraído hasta cierto punto estas complejidades para nuestra facilidad. Además, se han producido muchas mejoras en la calidad y el rendimiento de los HMD a lo largo de los años.

Dicho esto, todavía no hay suficientes herramientas de pruebas ni un enfoque de pruebas establecido en este ámbito. Las pruebas se personalizan contextualmente. Un desarrollo reciente de Thoughtworks es la herramienta de automatización de pruebas funcionales para Unity llamada [Arium](#). Arium es de código abierto y está disponible como paquete de Unity. Veamos brevemente algunos conceptos de Unity para entender cómo probarlos con esta herramienta.¹

Una *escena* en Unity representa un entorno de juego. Normalmente, cada nivel de un juego se denomina escena. Cada escena tiene su propia colección de objetos. Un *GameObject* es un elemento de la escena. Puede ser un accesorio, como una pelota, o un jugador. Las habilidades de estos objetos pueden programarse adjuntándoles *componentes* (un componente es cualquier característica o funcionalidad que esté vinculada a un *GameObject*). Unity proporciona muchos componentes incorporados para necesidades fundamentales como proyectar luz, colisiones, etc. Por ejemplo, podemos adjuntar un componente de luz a un *GameObject* para definir la iluminación de ese objeto. Cada *GameObject* también tiene un componente *Transform* por defecto que representa su posición, tamaño y rotación.

Arium proporciona las siguientes funciones para permitir la automatización de pruebas funcionales de aplicaciones Unity:

- `_arium.FindGameObject("Ball")` para encontrar un `gameObject` por su nombre
- `_arium.GetComponent<name_of_component>(<name_of_gameObject>)` para recuperar componentes de `gameObject`, que luego se pueden validar
- `_arium.PerformAction(new UnityPointerClick(), " <name_of_gameObject>")` para realizar acciones en `gameObjects` para navegar

Arium también puede ampliarse para realizar pruebas de usabilidad, pruebas experienciales y de inmersión, pruebas de rendimiento y pruebas de compatibilidad de las aplicaciones XR.

Esta es una lectura breve y nítida sobre las tecnologías emergentes y algunos de los aspectos de las pruebas que hay que tener en cuenta. Estas tendencias seguirán evolucionando, y puede que se generalicen antes de lo que esperamos. Así que, ¡continuemos para vigilar este espacio!

¹ Para una introducción mucho más completa, consulta *Game Programming with Unity and C# (Programación de juegos con Unity y C#)*, de Casey Hardman : *A Complete Beginner's Guide* (Apress).

Índice

A

Pruebas A/B, **Usabilidad**

fase de pruebas de aceptación, pruebas continuas, **estrategia de pruebas continuas**

accesibilidad

- Escáner de accesibilidad, **Android**
- texto alternativo, **Perceptible**
- **AndroidStudio** y, **Android**
- tecnologías de apoyo, **Ecosistema de Accesibilidad**
- ATAG (Directrices de Accesibilidad para Herramientas de Autor), **Ecosistema de Accesibilidad**
- control de audio, **perceptible**
- subtítulos, **Perceptible**
- colores, **perceptibles**
- Espresso y, **Android**
- navegación por teclado, **Operable**
- requisitos legales, **Pruebas de Accesibilidad**
- estrategia de pruebas móviles, **iOS**
- operabilidad, **Operable**
- jerarquía de páginas, **Perceptible**

- perceptibilidad, Perceptible
- robustez, Robusto
- lectores de pantalla, Ejemplo: Lectores de pantalla-Ejemplo: Lectores de pantalla
- transcripciones, Perceptible
- UAAG (Pautas de Accesibilidad del Agente de Usuario), Ecosistema de Accesibilidad
- comprensibilidad, Comprensible
- agentes de usuario, Ecosistema de Accesibilidad
- personas usuarias, Accesibilidad Personas Usuarias-AccesibilidadPersonas Usuarias
- WCAG (Pautas de Accesibilidad al Contenido en la Web), Ecosistema de Accesibilidad
- herramientas y prácticas de desarrollo web, Ecosistema de Accesibilidad
- Inspector de accesibilidad de XCode, iOS

marcos de desarrollo accesibles, marcos de desarrollo accesibles

Escáner de accesibilidad, Android, Escáner de accesibilidad

Pruebas de accesibilidad, Diez habilidades para pruebas Full Stack, Pruebas de accesibilidad

- Axe-core, Axe-core-Axe-core
- ejercicios
 - Lighthouse, Lighthouse-Flujo de trabajo

- Módulo Lighthouse Node, **Módulo Lighthouse Node-Workflow**
- WAVE, **WAVE-Flujo de trabajo**
- Módulo **Pa11y CI Node**, Módulo **Pa11y CI Node**
- estrategias, **Estrategia de pruebas de accesibilidad**
 - herramientas **automatizadas de auditoría**, **Herramientas automatizadas de auditoría de la accesibilidad**
 - listas de comprobación, **Lista de comprobación de accesibilidad**en historias de usuario-**Lista de comprobación de accesibilidad**en historias de usuario
 - pruebas manuales, **Pruebasmanuales-Pruebas manuales**
 - pruebas visuales, **Pruebas de accesibilidad**

árbol de accesibilidad, lectores de pantalla y, **Ejemplo: Lectores de pantalla**

ADRs (registros de decisión de arquitectura), **Prevención de Defectos sobre Detección de Defectos**

Desarrollo ágil

- pruebas dev-box, **repetir pruebas exploratorias por fases**
- **Prueba de mayúsculas a la izquierda** y, **Prueba de mayúsculas a la izquierda**

IA (inteligencia artificial), **Inteligencia Artificial y Aprendizaje Automático**

- Herramienta de pruebas visuales AppliTools Eyes, **AppliTools Eyes**, una herramienta potenciada por la IA-AppliToolsEyes, una herramienta potenciada por la IA

- IA visual, [Applitools Eyes](#), una herramienta potenciada por la IA
texto alternativo, accesibilidad, [Perceptible](#)
Android

- pruebas de accesibilidad, [Android](#)
- [Inspector de bases de datos](#), [Inspector de bases de datos de Android Studio](#)-[Inspector de bases de datos de Android Studio](#)
- emuladores, [Pruebas Exploratorias Manuales](#), [emulador Android](#)

Android Studio, accesibilidad y, [Android](#)
antipatrones en las pruebas funcionales automatizadas

- magdalena, [La magdalena](#)
- cucuricho de helado, [El cucuricho de helado](#)

[Apache Benchmark](#), Paso 6: Guión y ejecución de las pruebas de rendimiento mediante herramientas, [Apache Benchmark](#)

[Apache JMeter](#), [Pruebas funcionales automatizadas](#)

[Apache Spark](#), [Sistemas de procesamiento por lotes](#)

API (interfaz de programación de aplicaciones)

- RESTful, [Pruebas de API](#)
- Selenium WebDriver, [Selenium WebDriver](#)-[Selenium WebDriver](#)

[Pruebas de API](#), [Pruebas de API](#)

- aplicaciones blockchain, [Pruebas de aplicaciones blockchain](#)
- rutas de descubrimiento, [Pruebas de API](#)
- Postman, [Postman](#)-[Postman](#)
- WireMock, [WireMock](#)-[WireMock](#)

Herramientas APM (gestión del rendimiento de las aplicaciones), Paso 5: Integrar las herramientas APM

Appium, Appium

- Emulador Android, Emulador Android
- Appium 2.0 setup, Appium 2.0 setup
- Marco Java-Appium, Flujo de trabajo
- APIde rendimiento, API de rendimiento de Appium-Appium
- RPA (automatización robótica de procesos) y, Appium
- plugin de pruebas visuales, Appium Visual TestingPlug-in-Workflow
- flujo detrabajo, Workflow-Flujo de trabajo

arquitectura de la aplicación, pruebas exploratorias manuales y,
Comprender la aplicación

capa de aplicación, IoT (Internet de las Cosas), **Introducción a la arquitectura de cinco capas del IoT**

error deconfiguración de la aplicación, Error de configuración de la aplicación

herramientas de monitoreo del rendimiento de las aplicaciones(APM)
(ver APM (monitoreo del rendimiento de las aplicaciones))

vulnerabilidades de las aplicaciones

- autenticación, Autenticación y mala gestión de la sesión
- Inyeccióndecódigo, Inyecciónde código SQL-Inyección de código SQL
- vulnerabilidades conocidas, vulnerabilidades no gestionadas, vulnerabilidades conocidas no gestionadas

- Configuración errónea, Configuraciones erróneas de las aplicaciones
- Exposición de secretos, Exposición de secretos de aplicación- Applicationsecrets exposure
- gestión de sesiones, Autenticación y mala gestión de sesiones
- Inyección SQL, Inyección de código o SQL-Inyección de código SQL
- datos no encriptados, Datos privados no encriptados
- XSS (secuencias de comandos en sitios cruzados) , Secuencias de comandos en sitios cruzados

aplicaciones

- móvil, App
 - arquitectura, arquitectura de aplicaciones móviles- MobileApp Architecture
 - aplicaciones híbridas, App
 - web móvil, App
 - aplicaciones nativas, App
 - PWA (aplicaciones web progresivas), App
- Exposición de secretos, Exposición de secretos de aplicación- Applicationsecrets exposure

ApplitoolsEyes, Applitools Eyes, una herramienta potenciada por IA - ApplitoolsEyes, una herramienta potenciada por IA, Pruebas visuales Appvance, Creación de pruebas AR (realidad aumentada), Realidad Aumentada y Realidad Virtual

- Prueba de aplicaciones, Prueba de aplicaciones AR /VR-Prueba de aplicaciones AR/VR

registros de decisiones sobre arquitectura (ADR), Prevención de defectos sobre Detección de defectos

diseño de la arquitectura, rendimiento y, Factores que afectan al rendimiento de las aplicaciones

Pruebas dearquitectura, Pruebas CFR, Pruebas dearquitectura-Pruebas de arquitectura

ArchUnit, Pruebas de Arquitectura

inteligencia artificial (IA)(véase IA (inteligencia artificial))

activos, bloques de construcción

tecnologías de apoyo , Ecosistema de Accesibilidad

ATAG (Directrices de Accesibilidad para Herramientas de Autor), Ecosistema de Accesibilidad

ataques(ver ciberataques)

audio, accesibilidad, Perceptible

inteligencia aumentada, Introducción a los ensayos en tecnologías emergentes

realidad aumentada (RA)(ver RA (realidad aumentada))

servicio auth, token de acceso y, Building Blocks

autenticación

- vulnerabilidades de las aplicaciones y, Autenticación y gestión errónea de sesiones
- funcionalidades y, Aspectos interfuncionales

- GitHub, Flujo de trabajo

Directrices de Accesibilidad para Herramientas de Autor (ATAG),
Ecosistema de Accesibilidad

autorización, funcionalidades y, Aspectos interfuncionales

Pruebas funcionales automatizadas, Diez habilidades de pruebas full stack, Pruebas funcionales automatizadas, Estrategia de pruebas funcionales automatizadas

- herramientas AI/ML
 - creación de pruebas, Creación de pruebas
 - herramientas de gobernanza de pruebas, Gobernanza de pruebas
 - mantenimiento de pruebas, Mantenimiento de pruebas
 - análisis de informes de pruebas, Análisis de informes de pruebas
- antipatrones
 - magdalena, La magdalena
 - cono de helado, El cono de helado
- porcentaje de cobertura del código, Cobertura de automatización ¡100%!
- ejercicios, Ejercicios
 - pruebas de servicio, Pruebas de servicio - Configuración y flujo de trabajo
 - Pruebas funcionales de interfaz de usuario, Pruebas funcionales de interfaz de usuario - Configuración y flujo de trabajo

- Pruebas unitarias, Pruebas unitarias - Configuración flujo de trabajo
- implementación, Building Blocks
- Karate, Karate
- tipos demacropruebas, Introducción a los tipos de micropruebas y macropruebas
 - pruebas de contrato, Pruebas de contrato
 - pruebas de extremo a extremo, Pruebas de extremo a extremo
 - pruebas de integración, Pruebas de integración
 - pruebas de servicio, Pruebas de servicio-Pruebas de servicio
 - Pruebas funcionales de interfaz de usuario, Pruebas funcionales de interfaz de usuario-Pruebas funcionales de interfaz de usuario
 - pruebas unitarias, Pruebas unitarias-Pruebas unitarias
- tipos de micropruebas, Introducción a los tipos de micropruebas y macropruebas
 - Pruebas de contrato, Pruebas de contrato
 - pruebas de extremo a extremo, Pruebas de extremo a extremo
 - pruebas de integración, Pruebas de integración
 - pruebas de servicio, Pruebas de servicio-Pruebas de servicio
 - Pruebas funcionales de interfaz de usuario, Pruebas funcionales de interfaz de usuario-Pruebas funcionales de interfaz de usuario

- pruebasunitarias, Pruebasunitarias-Pruebas unitarias
 - Pacto, Pacto-Pacto
 - seguimiento de la cobertura de las pruebas de automatización, Estrategia de pruebas funcionales automatizadas
- pruebasautomatizadas, pruebas shift-lefty, Pruebas shift-left
AutoTester, Pruebas funcionales automatizadas
Avahi, Introducción a la arquitectura de cinco capas del IoT
Eje-núcleo,Eje-núcleo-Axe-núcleo

B

Aplicaciones B2C (de empresa a cliente), pruebas visuales y, Casos de uso críticos para el proyecto/empresa

pruebas de rendimiento backend, iEl rendimiento, las ventas y los fines de semana libres están correlacionados!-iEl rendimiento, las ventas y los fines de semana libres están correlacionados!

- objetivos de rendimiento, Objetivos de Rendimiento Simples

BackstopJS

- archivo de configuración backstop.json, Flujo de trabajo
- Node.js y, Configuración
- Puppeteer, BackstopJS
 - scripts, Flujo de trabajo
 - matriz de ventanas, flujo de trabajo
- Resemble.js, BackstopJS
- configuración, Configuración

- Visual Studio Code y, **Configuración**
- **flujo de trabajo, flujo de trabajo-flujo de trabajo**

estrangulador de ancho de banda, **controles ampliados:**
Estrangulador de red

Procesamiento **por lotes**, **Sistemas de procesamiento por lotes-Sistemas de procesamiento por lotes**

BDD (desarrollo basado en el comportamiento), **Configuración y flujo de trabajo**

benchmarking, **Tipos de patrones de carga**

Bitbucket, **Configuración**

blockchain, **Blockchain**

- Pruebas de API y, **Pruebas de aplicaciones Blockchain**
- consenso, **Introducción a los conceptos de Blockchain-Introducción a los conceptos de Blockchain**
- pruebas funcionales y, **Pruebas de aplicaciones Blockchain**
- libros de contabilidad, **Introducción a los conceptos de Blockchain**
- nodos, **Introducción a los conceptos de Blockchain**
- pruebas de rendimiento y, **Pruebas de aplicaciones Blockchain**
- seguridad y **Blockchain**
- pruebas de seguridad y, **Pruebas de aplicaciones Blockchain**
- contratos inteligentes, **Introducción a los conceptos de Blockchain**

Pruebas específicas de blockchain, Pruebas de aplicaciones de blockchain-Pruebas de aplicaciones de blockchain

Bonjour, Introducción a la Arquitectura de Cinco Capas del IoT

latasa de rebote, el rendimiento, las ventas y los fines de semana libres iestán correlacionados!

Límites, pruebas de valor, Análisis del valor límite-Análisis del valor límite

Análisis del valor límite, Análisis del valor límite-Análisis del valor límite construcciones rotas, empujar hacia, Principios y Etiqueta navegadores

- caching, pruebas de rendimiento y, Factores que afectan al rendimiento del frontend
- pruebas entre navegadores, pruebas entre navegadores
- compatibilidad con marcos, pruebas entre navegadores
- pruebas de interfaz de usuario web, Navegadores-Browsers

BrowserStack, Navegadores

ataques de fuerza bruta, Fuerza bruta

pruebas de errores, pruebas cruzadas entre navegadores

Imán para bichos, imán para bichos-imán para bichos

construye, rompe, Principios y Etiqueta

capa empresarial, IoT (Internet de las Cosas), Introducción a la arquitectura de cinco capas del IoT

prioridades empresariales, pruebas exploratorias manuales y, Comprender la aplicación

C

cachés, Cachés-Cachés

subtítulos, accesibilidad, Perceptible

Gráfico causa-efecto, Gráfico causa-efecto

CD (entrega continua), Pruebas continuas

- Implementación automatizada, El proceso CI/CT/CD
- frente a la CD (implementación continua), El proceso CI/CT/CD

CD (implementación continua) frente a CD (entrega continua), El proceso CI/CT/CD

CDN (redes de distribución de contenidos), Factores que afectan al rendimiento del frontend

PruebaCFR, Prueba de requisitos interfuncionales

- Pruebasdearquitectura, Pruebas dearquitectura-Pruebas de arquitectura
- ingeniería delcaos, Ingeniería delcaos-Experimento del caos
- pruebas de conformidad
 - GDPR (Reglamento General de Protección deDatos), Reglamento Generalde Protección deDatos (GDPR)- Reglamento Generalde Protección de Datos (GDPR)
 - PCI DSS (Estándar de Seguridad de Datos de la Industria de Tarjetas de Pago), PCI DSS y PSD2
 - PSD2 (Directiva de Servicios de Pago), PCIDSS y PSD2- PCIDSS y PSD2
- Pruebas de infraestructura, Pruebas de infraestructura

- conformidad, Pruebas de infraestructura
- Pruebas de extremo a extremo, Pruebas de infraestructura
- IaC (Infraestructura como código), Pruebas de infraestructura
- operatividad, Pruebas de infraestructura
- seguridad, Pruebas de infraestructura
- Terraform, Pruebas de infraestructura
- TFLint, Pruebas de infraestructura
- estrategia de pruebas móviles, CFR Testing-CFRTesting
- estrategias, CFR Testing Strategy
 - funcionalidad, CFR Testing Strategy, Funcionalidad
 - rendimiento, estrategia de pruebas del CFR, rendimiento
 - fiabilidad, estrategia de pruebas del CFR, Fiabilidad-Fiabilidad
 - compatibilidad, estrategia de pruebas del CFR, compatibilidad-soportabilidad
 - usabilidad, estrategia de pruebas del CFR, usabilidad-usabilidad

CFRs (requisitos interfuncionales), Diez habilidades de prueba de pila completa, Prueba de requisitos interfuncionales

- definiciones, Bloques de construcción-Bloques de construcción
- frente a requisitos no funcionales, Prueba de Requisitos Transfuncionales

ceguera al cambio, Introducción a las pruebas visuales

Ingeniería del caos, Fiabilidad

- Pruebas CFR, [Ingeniería del Caos-Experimento del Caos](#)

Cromático, [Libro de cuentos](#)

[Chrome DevTools](#), [Chrome DevTools](#) y [Postman](#)-[Chrome DevTools](#) y [Postman](#), [Chrome DevTools](#)-[Chrome DevTools](#)

- cookies, [Chrome DevTools](#)
- usuarios nuevos, [Chrome DevTools](#)
- número de peticiones de página, [Chrome DevTools](#)
- errores de página, [Chrome DevTools](#)
- Comportamientos de inactividad del servicio, [Chrome DevTools](#)
- integración de UI y API, [Chrome DevTools](#)
- comportamiento de la IU, redes lentas, [Chrome DevTools](#)

Chrome, [pruebas cruzadas entre navegadores](#), [Cross-Browser Testing](#)

ChromeDriver ejecutable, [Configuración y flujo de trabajo](#)

CI (integración continua), [Pruebas continuas](#), [Introducción a la integración continua](#)-[Introducción a la integración continua](#)

- descripción, [Introducción a la integración continua](#)
- JMeter, [Integración en CI](#)
- versus pruebas continuas, [Estrategia de pruebas continuas](#)

Servidor CI, [El proceso CI/CT/CD](#)

- fase de construcción y prueba, [El proceso CI/CT/CD](#)
- commits, [El proceso CI/CT/CD](#)

CI/CD (Integración Continua/Entrega Continua), **pruebas por turnos**, **Pruebas por turnos**

Proceso CI/CT/CD

- Etiqueta, **Principios** y Etiqueta-Principios y Etiqueta
- principios, **Principios** y Etiqueta-Principios y Etiqueta
- VCS (sistema de control de versiones), **El proceso CI/CT/CD**

plataformas de pruebas alojadas en la nube, **Navegadores**

coaching, habilidades **blandas**, Las habilidades blandas ayudan a **crear una mentalidad de calidad ante todo**

CoAP (Constrained Application Protocol), **Introducción a la arquitectura de cinco capas del IoT**

complejidad del código, rendimiento y, **Factores que afectan al rendimiento de las aplicaciones**

inyección de código, inyección de código o SQL

colaboración

- pruebas continuas y, **Beneficios**
- primeros principios y, **La comunicación y la colaboración son la clave de la calidad**
- habilidades **blandasy**, Las habilidades blandas ayudan a crear una mentalidad de calidad ante todo

colores, accesibilidad, **Perceptible**

comentar pruebas fallidas, **Principios y etiqueta**

comete

- frecuencia, **principios y etiqueta**

- Git VCS, El proceso CI/CT/CD
- código autocomprobado, Principios y Etiqueta

comunicación

- primeros principios y, La comunicación y la colaboración son la clave de la calidad
- habilidades blandas, Las habilidades blandas ayudan a crear una mentalidad de calidad ante todo

pruebas de conformidad, pruebas CFR

- Reglamento General de Protección de Datos (RGPD),
Reglamento General de Protección de Datos (RGPD)-Reglamento
General de Protección de Datos (RGPD)
- PCIDSS (Payment Card Industry Data Security Standard), PCI
DSS y PSD2
- PSD2 (Directiva de Servicios de Pago), PCIDSS y PS D2-PCIDSS
y PSD2

compromisos, seguridad, Building Blocks

configuración, errores de configuración de la aplicación, errores de
configuración de la aplicación

certificación de conformidad, accesibilidad, Pruebas manuales

cosas conectadas, Introducción a las Pruebas en Tecnologías
Emergentes

consenso, blockchain y, Introducción a los conceptos de block chain-
Introducción a los conceptos de blockchain

modelos de coherencia, Bases de datos

Protocolo de Aplicación Restringida (CoAP), [Introducción a la Arquitectura de Cinco Capas del IoT](#)

Contenedores, [Contenedores de prueba](#), [Contenedores de prueba-Contenedores de prueba](#)

Redes de distribución de contenidos (CDN), [Factores que afectan al rendimiento del frontend](#)

entrega continua(CD)(ver [CD \(entrega continua\)](#))

integración continua (IC)(ver [IC \(integración continua\)](#))

Prueba de Certificación de Integración Continua, [Principios y Etiqueta](#)

Pruebas continuas, [Diez habilidades de pruebas full stack](#)

- fase de aceptación, [estrategia de pruebas continuas](#)
- etapa de construcción-prueba, [Estrategia de Pruebas Continuas](#), [Estrategia de Pruebas Continuas](#)
- porcentaje de cambios fallidos, [Las cuatro métricas clave](#)
- colaboración y, [Beneficios](#)
- objetivos comunes de calidad, [Beneficios](#)
- propiedad de la entrega, [Ventajas](#)
- fase de implementación, [Estrategia de Pruebas Continuas](#)
- Implementación y, [Beneficios](#)
- frecuencia de implementación, [Las cuatro métricas clave](#)
- detección precoz de defectos, [Ventajas](#)
- ejercicios
 - Git, [Flujo de trabajoGit](#)

- Jenkins, Jenkins
- fase de pruebas funcionales, Estrategia de Pruebas Continuas
- tiempo de espera, Las cuatro métricas clave
- tiempo medio de restauración, Las cuatro métricas clave
- métricas, LasCuatro Métricas Clave-LasCuatro Métricas Clave
- etapa de regresión nocturna, Estrategia de Pruebas Continuas
- pruebas de humo, Estrategia de Pruebas Continuas
- estrategias, Estrategia de pruebas continuas-Beneficios
- frente a CI (integración continua), Estrategia de pruebas continuas

pruebas de contrato, Pruebas de contrato

forja de galletas, Forja de galletas

muestreo por criterios específicos, Muestreo

pruebas entre navegadores

- desde la izquierda, Cross-Browser Testing
- Retroalimentación funcional, Pruebas cruzadas, Pruebas cruzadas
- pruebas visuales, pruebas entrenavegadores-pruebas entre navegadores

requisitos interfuncionales(CFRS)(ver CFR (requisitos interfuncionales))

pruebas de requisitos interfuncionales(ver pruebas CFR)

Cross-site scripting (XSS), Secuencias de comandos en sitios cruzados

Operaciones CRUD, Bases de datos

cryptojacking, Criptopiratería

CSS (Hojas de estilo en cascada), pruebas e, Introducción a las pruebas visuales

TC (pruebas continuas), Pruebas continuas

Pepino, Karate

antipatrón de la magdalena, La magdalena

impacto en el cliente, pruebas visuales y, Casos de uso críticos para el proyecto/negocio

ciberataques

- fuerza bruta, Fuerza bruta
- forja de galletas, Forja de galletas
- cryptojacking, Criptopiratería
- phishing, suplantación de identidad
- ransomware, ransomware
- ingeniería social, Ingeniería social
- raspado web, Web scraping
- XSS (secuencias de comandos en sitios cruzados) , Secuencias de comandos en sitios cruzados

ciberdelincuencia, Pruebas desseguridad-Pruebas de seguridad

Ciprés, Ciprés-Workflow

D

DAST (Pruebas Dinámicas de Seguridad de las Aplicaciones),
Estrategia de Pruebas de Seguridad

desviación de datos, sistemas de procesamiento por lotes

Prueba de datos, Diez habilidades de prueba de pila completa,
Prueba de datos

- procesamiento por lotes, Sistemas de procesamiento por lotes-Sistemas de procesamiento por lotes
- cachés, Cachés-Cachés
- bases de datos, Bases de datos
 - valores límite, Bases de datos
 - concurrencia, Bases de datos
 - coherencia de orden, Bases de datos
 - lectura-escritura, Bases de datos
 - bases de datos relacionales, Bases de datos
 - replicación, Bases de datos
 - esquema, Bases de datos
 - SQL, Bases de datos
 - casos de prueba, Bases de datos
 - viajes en el tiempo, Bases de datos
 - conflictos de escritura, Bases de datos
- Deequ, Deequ-Deequ
- flujos de eventos, flujos de eventos

- ejercicios
 - JDBC, JDBC-Configuración y flujo de trabajo
 - Kafka, Apache Kafka y Zerocode-Flujo de trabajo
 - SQL, SQL-Actualización y eliminación
 - Zerocode, Apache Kafka y Zerocode-Workflow
- pruebas funcionales y, Pruebas de datos
- estrategia de pruebas móviles, Pruebas de datos-Pruebas de datos
- pirámide y, JDBC
- estrategias, Estrategia de Pruebas de Datos
 - pruebas funcionales automatizadas, Estrategia de Pruebas de Datos
 - prueba exploratoria manual, Estrategia de prueba de datos
 - pruebas de rendimiento y Estrategia de pruebas de datos
 - seguridad y privacidad, Estrategia de Pruebas de Datos
- Contenedores de Pruebas, Contenedores de Pruebas-Contenedores de Pruebas

transferencias de datos, pruebas de rendimiento y, Factores que afectan al rendimiento del frontend

pruebas de rendimiento basadas en datos

- JMeter, Pruebas de rendimiento basadas en datos

Inspector de bases de datos, Inspector de bases de datos de Android Studio-Inspector de bases de datos de Android Studio

BD (bases de datos), Introducción a los micro y macro tipos de pruebas, Bases de datos, Bases de datos

- valores límite, Bases de datos
- concurrencia, Bases de datos
- operaciones CRUD, Bases de datos
- coherencia de ordenación, Bases de datos
- rendimiento y, Factores que afectan al rendimiento de las aplicaciones
- bases de datos relacionales, Bases de datos
 - UUIDs, Bases de datos
- replicación, Bases de datos
- escalabilidad, Bases de datos
- esquema, Bases de datos
- casos de prueba, Bases de datos
- viajes en el tiempo, Bases de datos
- conflictos de escritura, Bases de datos
- escritura, lectura, Bases de datos

Ataque DDoS (denegación de servicio distribuida), Denegación de servicio

cola de letra muerta, flujos de eventos

Tabla de decisión, Tabla de decisión-Tabla de decisión

Deequ, Deequ-Deequ

propiedad de la entrega, pruebas continuas y, Beneficios

implementación

- pruebas continuas y, **Beneficios**

- frecuencia, **Pruebas continuas**, **Las cuatro métricas clave**

sistemas de diseño, **Casos de uso críticos para el proyecto/empresa**

diseño, **pruebas de desplazamiento a la izquierday**, **Pruebas de desplazamiento a la izquierda**

pruebas dev-box, **repetir pruebas exploratorias por fases**

desarrollo, **pruebas shift-izquierday**, **Pruebas shift-izquierda**

dispositivos

- IoT (Internet de los objetos), **Internet de los objetos**

- móvil

- fabricante de **dispositivos**, **Dispositivos**

- hardware, **Dispositivos**

- sistema operativo, **Dispositivos**

- densidad de píxeles, **Dispositivos**

- resolución de pantalla, **Dispositivos**

- tamaño de pantalla, **Dispositivos**

digitalización, **Introducción al Full Stack Testing**

Consultas DNS (Servicio de Nombres de Dominio), **Factores que afectan al rendimiento del Frontend**

Docker, **Configuración**

dominios, pruebas exploratorias manuales y, **Comprender la aplicación**

Ataque DoS (denegación de servicio), Denegación de servicio controladores, Selenium WebDriver, Selenium WebDriver Pruebas dinámicas de seguridad de las aplicaciones (DAST), Estrategia de pruebas de seguridad

E

detección precoz de defectos, pruebas continuas, Beneficios ecommerce UI, Introducción a los tipos de pruebas micro y macro perímetro, Prueba Exploratoria Manual

Pruebas empáticas, Pruebas empáticas emuladores, Pruebas Exploratorias Manuales

- Android, emulador de Android cifrado, Bloques de construcción
- datos no encriptados y, Datos privados no encriptados

Pruebas de extremo a extremo, Pruebas de extremo a extremo partición de clases de equivalencia, Partición de clases de equivalencia

Método de adivinación de errores, Método de adivinación de errores- Método de adivinación de errores

Tratamiento de errores, Fallos y tratamiento de errores

Escalada de privilegios, Escalada de privilegios, Ejercicio de modelado de amenazas

Espresso, accesibilidad y, Android flujos de eventos, Event Streams

- Apache Kafka, **flujos de eventos**
- Google Cloud Pub/Sub, **flujos de eventos**
- casi en tiempo real, **flujos de eventos**
- editor, **flujos de eventos**
- RabbitMQ, **flujos de eventos**
- suscriptores, **flujos de eventos**
- temas, **Event Streams**

eventos, **flujos de eventos**

coherencia eventual, **Bases de datos**

estrategia de espera explícita, **Selenium WebDriver**

Prueba exploratoria, Prueba exploratoria manual

- (véase también Prueba exploratoria manual)
- marcos, **Marcos de pruebas exploratorias**
 - análisis del**valor límite**, **Análisis del valor límite-Análisis del valor límite**
 - gráfico causa-efecto, **Gráfico causa-efecto**
 - tabla de**decisión**, **Tabla de decisión-Tabla de decisión**
 - partición de **clases de equivalencia**, **Partición de clases de equivalencia**
 - método de **suposición de errores**, **Método de suposición de errores-Método de suposición de errores**
 - prueba por **pares**, **Prueba por pares-Prueba por pares**
 - muestreo, **Muestreo-Muestreo**

- transición de estado, Transición de estado-Transición de estado
- prueba por monos, Comprender la aplicación expresiones, SQL, Expresiones y predicados Programación Extrema (XP), Pruebas por turnos

F

capturas de pantalla de fallos, configuración y flujo de trabajo fallos, Fallos y gestión de errores

- Apropiación, principios y etiqueta
- pruebas decaracterísticas, accesibilidad, Pruebas manuales características, Pruebas Exploratorias Manuales retroalimentación, primeros principios de las pruebas, Retroalimentación rápida, Retroalimentación continua primeros principios

- la colaboración, la comunicación y la colaboración son la clave de la calidad
- la comunicación, la comunicación y la colaboración son la clave de la calidad
- retroalimentación continua, retroalimentación continua
- defectos, prevención sobre detección, Prevención de Defectos sobre Detección de Defectos-Prevención de Defectos sobre Detección de Defectos
- Pruebas empáticas, Pruebas empáticas
- respuesta rápida, respuesta rápida

- Pruebas a nivel macro, Pruebas a nivel micro y macro
- métricas, Medir las métricas de calidad-Medir las métricas de calidad
- Pruebas demicronivel, Pruebas de micronivel y macronivel

Flipkart, Introducción al Full Stack Testing

estrategia de espera fluida, Selenium WebDriver

FORTRAN, pruebas y, Pruebas funcionales automatizadas

Ataque FriendFinder, Fuerza bruta

Pruebas de rendimiento del frontend, Bloques de construcción de las pruebas de rendimiento del frontend-Bloques de construcción de las pruebas de rendimiento del frontend

- caché del navegador, Factores que afectan al rendimiento del frontend
- CDN (redes de distribución de contenidos), Factores que afectan al rendimiento del frontend
- complejidad del código, factores que afectan al rendimiento del frontend
- transferencia de datos, factores que afectan al rendimiento del módulo de acceso
- Consultas DNS, Factores que afectan al rendimiento del Frontend
- ejercicios, Ejercicios
 - Lighthouse, Lighthouse-Flujo de trabajo
 - WebPageTest, WebPageTest-Workflow
- macropruebas, Estrategia de pruebas del módulo de acceso

- métricas, Métricas de rendimiento del Frontend-Métricas de rendimiento del Frontend
- pruebas de micronivel, Estrategia de pruebas del frontend
- latencia de la red, Factores que afectan al rendimiento del frontend
- pruebas visuales, estrategia de pruebas del frontend, pruebas de rendimiento del frontend
 - pruebas de accesibilidad, Pruebas de accesibilidad
 - pruebas cross-browser, Pruebas cross-browser-Pruebas cross-browser
 - Pruebas de rendimiento del frontend, Pruebas de rendimiento del frontend
 - pruebas funcionales de extremo a extremo, Pruebas funcionales de extremo a extremo
 - pruebas de integración/componentes, Pruebas de integración/componentes-Pruebas de integración/componentes
 - pruebas de instantáneas, Snapshot Tests -Pruebas de instantáneas
 - pruebas unitarias, Pruebas unitarias
 - pruebas visuales, pruebas visuales

uniones externas completas, uniones

pruebas funcionales automatizadas

- prueba de datos y, Estrategia de prueba de datos

- estrategia de pruebas móviles, **Pruebas Funcionales Automatizadas**

pruebas funcionales de extremo a extremo, pruebas visuales y, **Pruebas funcionales de extremo a extremo**

feedback funcional, **pruebas entre navegadores**, pruebas entre navegadores

automatización de pruebas funcionales, **estrategia de pruebas de seguridad**

pruebas funcionales

- pruebas funcionales automatizadas, **Ten Full Stack Testing Skills**
- aplicaciones **blockchain**, probar aplicaciones blockchain
- **Prueba de datos**, Prueba de datos

etapa de pruebas funcionales, pruebas continuas, **Estrategia de Pruebas Continuas**

funcionalidades

- **Aspectos interfuncionales**, **Aspectos interfuncionales-Aspectos interfuncionales**
- rutas de descubrimiento, **flujos de usuario funcionales**, **aspectos interfuncionales**
- Tratamiento de errores, **Fallos y tratamiento de errores**
- fallos, **Fallos y tratamiento de errores**
- Flujo **funcional de usuario**, **Flujos funcionales de usuario-Flujos funcionales de usuario**
- IU (interfaz de usuario)
 - apariencia, **La apariencia de la IU**

funcionalidad

- Pruebas CFR, Funcionalidad
- definición, Prueba Exploratoria Manual

Functionize, Creación de pruebas

Funciones, SQL, Funciones y operadores

G

Gatling, Paso 6: Guión y ejecución de las pruebas de rendimiento mediante herramientas

Reglamento General de Protección de Datos (RGPD), Reglamento General de Protección de Datos (RGPD)-Reglamento General de Protección de Datos (RGPD)

geolocalización, rendimiento y, Factores que afectan al rendimiento de las aplicaciones

Declaraciones de Gherkin, Karate

Git, Git

Sistema VCS Git, El proceso CI/CT/CD

GitHub, Configuración

- autenticación, Flujo de trabajo
- repositorios, Configuración

Google Cardboard, Realidad Aumentada y Realidad Virtual

Gradle, Maven

graficar, causa-efecto, Graficar causa-efecto

H

colgar porcentaje de fallos, **Pruebas continuas**
hardware, dispositivos móviles, **Dispositivos**
hashing, **Bloques de construcción**
forma de panal de abeja para pruebas, **Estrategia de pruebas funcionales automatizadas**
HTC VIVE, **Realidad Aumentada y Realidad Virtual**
HTML, **pruebas instantáneas**, **Pruebas instantáneas**
interacción similar a la humana, **Introducción a los ensayos en tecnologías emergentes**
aplicaciones híbridas, **App**

I

IaC (Infraestructura como Código), **Pruebas de Infraestructura**
IAST (Pruebas Interactivas de Seguridad de las Aplicaciones),
Estrategia de Pruebas de Seguridad
antipatrón del cono de helado, **El cono de helado**
escaneado de imágenes, **estrategia de pruebas de seguridad**
estrategia de espera implícita, **Selenium WebDriver**
influencia, habilidades **blandas**, **Las habilidades blandas ayudan a crear una mentalidad de calidad ante todo**
divulgación **de información**, **Divulgación de información**
Infraestructura como Código (IaC), **Pruebas de Infraestructura**
pruebas de infraestructura, **Fiabilidad**

pruebas de infraestructura, pruebas CFR, Pruebas de infraestructura

- conformidad, Pruebas de infraestructura
- Pruebas de extremo a extremo, Pruebas de infraestructura
- IaC (Infraestructura como código), Pruebas de infraestructura
- operatividad, Pruebas de Infraestructura
- Seguridad, Pruebas de infraestructura
- Terraform, Pruebas de infraestructura
- TFLint, Pruebas de infraestructura

infraestructura, rendimiento y, Factores que afectan al rendimiento de las aplicaciones

Manipulación de entradas, Manipulación de entradas

pruebas de integración, Pruebas de integración

pruebas de integración/componentes, pruebas visuales, Pruebas de integración/componentes-Pruebas de integración/componentes

IntelliJ, proyecto Maven, configuración y flujo de trabajo

Pruebas interactivas de seguridad de las aplicaciones (IAST), Estrategia de pruebas de seguridad

internacionalización, pruebas de usabilidad y, Usabilidad

Internet de las Cosas(IoT) (ver IoT (Internet de las Cosas))

Pruebas de accesibilidad de iOS, iOS

IoT (Internet de las Cosas), Internet de las Cosas

- capa de aplicación, Introducción a la arquitectura de cinco capas del IoT

- pruebas de aplicaciones
 - integración hardware/software, Pruebas de aplicaciones IoT
 - interoperabilidad, Pruebas de aplicaciones IoT
 - conectividad de red, Pruebas de aplicaciones IoT
 - rendimiento, Pruebas de aplicaciones IoT
 - privacidad, Pruebas de aplicaciones IoT
 - seguridad, Pruebas de aplicaciones IoT
 - usabilidad, Pruebas de aplicaciones IoT
- capa empresarial, Introducción a la arquitectura de cinco capas de IoT
- dispositivos, Internet de las Cosas
- capa de middleware, Introducción a la Arquitectura de Cinco Capas de IoT
- capa de red, Introducción a la Arquitectura de Cinco Capas de IoT
- capa de percepción, Introducción a la Arquitectura de Cinco Capas de IoT

IPMs (reuniones de planificación de iteraciones), Pruebas de Desplazamiento a la Izquierda, Prevención de Defectos sobre Detección de Defectos

J

Marco Java-Appium, Flujo de trabajo

Java-REST Assured Framework, Java-REST Assured Framework-Configuración y flujo de trabajo

Java-Selenium WebDriver, JDBC

- Maven, Maven-Maven
- Modelo de objetos de página, Modelode objetosde página-Modelo de objetos de página
- Requisitos previos, Requisitos previos
- Selenium WebDriver, Selenium WebDriver
 - componentes, Selenium WebDriver
- configuración, Configuración y flujo de trabajo, Configuración y flujo de trabajo
- TestNG, TestNG

JavaScript, compatibilidad con versiones anteriores, pruebas entre navegadores

Marco JavaScript-Cypress, Marco JavaScript-Cypress

- Cypress, Cypress-Cypress
- requisitos previos, Requisitos previos
- configuracióny flujo de trabajo, Configuración y flujo de trabajo -Configuracióny flujo de trabajo

JDBC (Java Database Connectivity), JDBC-ApacheKafka y Zerocode

Jenkins

- crear activadores, flujo de trabajo
- salpicadero, Configuración
- configuración, Configuración
- flujo de trabajo, flujo de trabajo-flujode trabajo

Jest, Pruebas instantáneas

Jira, Estrategia de pruebas funcionales automatizadas

JMeter, Pruebas funcionales automatizadas, Paso 6: Guión y ejecución de las pruebas de rendimiento mediante herramientas, Paso 6: Guión de los casos de prueba y ejecución mediante JMeter

- Informe agregado, Flujo de trabajo
- Integración en CI, Integración en CI
- Pruebas de rendimiento basadas en datos, Pruebas de rendimiento basadas en datos
- GUI, Grupo de hilos, Flujo de trabajo
- escuchadores, flujo de trabajo
- Pruebas de carga, Flujo de trabajo
- diseño de casos de prueba de rendimiento, Diseño de otros casos de prueba de rendimiento-Diseño de otros casos de prueba de rendimiento
- configuración, Configuración
- pruebas decarga, Diseño de otros casos de prueba de rendimiento
- Ver resultados Vista en árbol, Flujo de trabajo
- flujo de trabajo, Flujo de trabajo-Flujo de trabajo

uniones, Uniones

JUnit, Pruebas unitarias, JUnit-Setup y flujo de trabajo

- Spring Data JPA, Pruebas de integración

K

k6, Paso 6: Guión y ejecución de las pruebas de rendimiento utilizando herramientas

Kafka

- brokers, Apache Kafka y Zerocode
- instalación, con Docker, Setup
- mensajes, Apache Kafka y Zerocode
- offset, Apache Kafka y Zerocode
- particiones, Apache Kafka y Zerocode
- retención, Apache Kafka y Zerocode
- esquemas, Apache Kafka y Zerocode
- configuración, Configuración
- temas, Apache Kafka y Zerocode
- Zerocode y, Flujo de trabajo

Kárate, Kárate

navegación con teclado, accesibilidad, Operable

KPI (indicadores clave de rendimiento), Indicadores clave de rendimiento-Tipos de pruebas de rendimiento

- objetivo, Paso 1: Definir los KPI objetivo-Paso 1: Definir los KPI objetivo
- casos de prueba, Paso 2: Definir los casos de prueba

L

tiempo de espera, **Pruebas continuas**

- pruebas continuas, **Las cuatro métricas clave**

uniones izquierdas, **Uniones**

bibliotecas, Selenium WebDriver, **Selenium WebDriver**

Lighthouse,Lighthouse-Flujo de trabajo

Herramienta de evaluación de la accesibilidad

Lighthouse,Lighthouse-Workflow

Herramienta de evaluación de la accesibilidad del **módulo Lighthouse Node, Lighthouse NodeModule-Workflow**

patrones de carga, pruebas de rendimiento y

- patrón pico-descanso, **Tipos de patrones de carga**
- patrón de rampa de subida constante, **Tipos de patrones de carga**
- patrón de rampa escalonada, **Tipos de patrones de carga**

pruebas de carga, Scala, **Gatling**

pruebas de carga/volumen, **Tipos de pruebas de rendimiento**

localización, pruebas de usabilidad y, **Usabilidad**

M

aprendizaje automático(ML)(véase ML (aprendizaje automático))

tipos de macropruebas, **Introducción a los tipos de micropruebas y macropruebas**

- pruebas de contrato, **Pruebas de contrato**

- pruebas de extremo a extremo, Pruebas de extremo a extremo
- pruebas de integración, Pruebas de integración
- pruebas de servicio, Pruebas de servicio-Pruebas de servicio
- pirámide de pruebas y, Estrategia de pruebas funcionales automatizadas
- pruebas funcionales de interfaz de usuario, pruebas funcionales de interfaz de usuario-pruebas funcionales de interfaz de usuario
- pruebas unitarias, Pruebas unitarias-Pruebas unitarias

pruebas a nivel macro, Pruebas a nivel micro y macro

- estrategia de pruebas frontales, Estrategia de pruebas frontales

Pruebas exploratorias manuales, Diez habilidades de pruebas full stack, Pruebas exploratorias manuales, Estrategia de pruebas de seguridad

- (ver también pruebas exploratorias)
- aplicación y, Comprender la aplicación
 - arquitectura de la aplicación, Comprender la aplicación
 - prioridades empresariales, Comprender la aplicación
 - configuración y, Comprender la aplicación
 - dominio, Comprender la aplicación
 - infraestructura, Comprender la aplicación
 - personas usuarias, Comprender la aplicación
- Prueba de datos y, Estrategia de prueba de datos
- ejercicios
 - Pruebas de API, Pruebas de API-WireMock

- Pruebas de interfaz de usuario web, Pruebas de interfaz de usuario web-ChromeDevTools
- por partes, Explorar por partes-Explorar por partes
- estrategia de pruebas móviles, Pruebas exploratorias manuales-Pruebas exploratorias manuales
- repetir, fases, Repetir las pruebas exploratorias en fases-Repetir las pruebas exploratorias en fases

pruebas manuales, pruebas de accesibilidad, Pruebas manuales

- Pruebas de certificación de conformidad, Pruebas manuales
- Pruebas de características, Pruebas manuales
- Pruebas de lanzamiento, Pruebas manuales
- pruebas de historias de usuario, Pruebas manuales

Maven, Maven-Maven

tiempo medio de restauración, Pruebas continuas

- pruebas continuas y, Las cuatro métricas clave

mentoring, habilidades blandas, Las habilidades blandas ayudan a crear una mentalidad de calidad ante todo

Mercury Interactive, Pruebas funcionales automatizadas

Message Queuing Telemetry Transport (MQTT), Introducción a la arquitectura de cinco capas del IoT

métricas, primeros principios de las pruebas, Medir las métricas de calidad-Medir las métricas de calidad

microtipos de pruebas, Introducción a los microtipos y macrotipos de pruebas

- pruebas de contrato, Pruebas de contrato
- pruebas de extremo a extremo, Pruebas de extremo a extremo
- pruebas de integración, Pruebas de integración
- pruebas de servicio, Pruebas de servicio-Pruebas de servicio
- pirámide de pruebas y, Estrategia de pruebas funcionales automatizadas
- pruebas funcionales de interfaz de usuario, pruebas funcionales de interfaz de usuario-pruebas funcionales de interfaz de usuario
- pruebas unitarias, Pruebas unitarias-Pruebas unitarias

Pruebas de micronivel, Pruebas de micronivel y macronivel

- estrategia de pruebas frontales, Estrategia de pruebas frontales

capa de middleware, IoT (Internet de las cosas), Introducción a la arquitectura de cinco capas del IoT

ML (aprendizaje automático), Introducción al aprendizaje automático

- pruebas de aplicaciones
 - validación de la integración, Prueba de aplicaciones de ML
 - validación del sesgo del modelo, Prueba de aplicaciones de ML
 - validación de la calidad del modelo, Probar aplicaciones ML
 - validación de datos de entrenamiento, Prueba de aplicaciones ML-Prueba de aplicaciones ML
- entrenamiento del modelo, Introducción al Aprendizaje Automático
- conjunto de pruebas, Introducción al Aprendizaje Automático

- conjunto de entrenamiento, Introducción al Aprendizaje Automático

arquitectura de aplicaciones móviles, Arquitectura de aplicaciones móviles-MobileApp Architecture

paisaje móvil, Introducción al paisaje móvil

- aplicaciones, App
 - híbrido, App
 - web móvil, App
 - nativa, App
 - PWA (aplicaciones web progresivas), App
- dispositivos
 - fabricante de dispositivos, Dispositivos
 - hardware, Dispositivos
 - sistema operativo, Dispositivos
 - densidad de píxeles, Dispositivos
 - tamaño de pantalla, Dispositivos
- red, Red

redes móviles, Red

pirámide de pruebas móviles, Perspectivas: La pirámide de pruebas móviles

Pruebas móviles, Diez habilidades de pruebas full stack, Pruebas móviles

- Escáner de accesibilidad, Escáner de accesibilidad

- Inspector de bases de datos, Inspector de bases de datos de Android Studio-Inspector de bases de datos de Android Studio
- ejercicios
 - Appium, Appium-Workflow
 - Appium Visual Testing Plug-in, Appium Visual Testing Plug-in-Workflow
- paisaje móvil, Introducción al paisaje móvil
 - dispositivos, Dispositivos-Devices
- herramientas de pruebas de rendimiento
 - API de rendimiento de Appium, API de rendimiento de Appium-Appium's performance API
 - MobSF, MobSF-MobSF
 - Mono, Mono
 - Estrangulador de red, Controles ampliados: Estrangulador de red
 - Qark, Qark
- estrategias, Estrategia de pruebas móviles-Pruebas exploratorias manuales
 - pruebas de accesibilidad, Pruebas de accesibilidad-Android
 - Pruebas CFR, Pruebas CFR-Pruebas CFR
 - pruebas de datos, Pruebas de datos-Pruebas de datos
 - pruebas automatizadas funcionales, Pruebas automatizadas funcionales

- Prueba exploratoria manual, Prueba exploratoria manual-
Prueba exploratoria manual
- pruebas de rendimiento, Pruebas de rendimiento-Pruebas de rendimiento
- pruebas desseguridad, Pruebas desseguridad-Pruebas de seguridad
- pruebas visuales, pruebas visuales

aplicaciones web móviles, App

MobSF (Marco de Seguridad Móvil), Pruebas de seguridad, MobSF-
MobSF

prueba del mono, comprender la aplicación

Herramienta de pruebas Monkey, Monkey

MQTT (Message Queuing Telemetry Transport), Introducción a la arquitectura de cinco capas del IoT

flujos de usuarios múltiples, flujos de usuarios funcionales

pruebas de mutación, cobertura del código y, cobertura 100% de automatización!

N

aplicaciones nativas, App

casi en tiempo real, flujos de eventos

Consultas anidadas en SQL, Consultas anidadas

NetArchTest, Pruebas de Arquitectura

latencia de la red, pruebas de rendimiento, Factores que afectan al rendimiento de las aplicaciones, Factores que afectan al rendimiento

del frontend

capa de red, IoT (Internet de las Cosas), [Introducción a la arquitectura de cinco capas del IoT](#)

estrangulador de red, pruebas móviles, [controles ampliados: Estrangulador de red](#)

redes, móvil, [Red](#)

NFR (requisitos no funcionales), [Diez habilidades para las pruebas de pila completa](#)

- frente a los RFC (requisitos interfuncionales), [Pruebas de requisitos interfuncionales](#)

etapa de regresión nocturna, pruebas continuas, [Estrategia de Pruebas Continuas](#)

nodos, blockchain, [Introducción a los Conceptos de Blockchain](#)

requisitos no funcionales (NFR), [Diez habilidades de prueba de pila completa](#)

NUnit, [Pruebas unitarias](#)

O

OAuth 2.0, [bloques de construcción](#)

Oculus Go, [Realidad Aumentada y Realidad Virtual](#)

Oculus Quest, [Realidad Aumentada y Realidad Virtual](#)

operabilidad, accesibilidad, [Operable](#)

sistemas operativos, dispositivos móviles, [Dispositivos](#)

operadores, SQL, [Funciones y operadores](#)

resultados, habilidades blandas, Las habilidades blandas ayudan a crear una mentalidad de calidad ante todo

OWASPDependency-Check, OWASP Dependency-Check-Setup y Workflow

OWASP ZAP (Proxy de Ataque Zed)

- Integración de CI, IntegrarZAP con CI -IntegrarZAP con CI
- Escaneado, Escaneado-Escaneado
- configuración, Configuración
- flujo de trabajo, flujo de trabajo-flujo de trabajo
- Araña ZAP, Araña ZAP

P

Módulo Nodo Pa11y CI, Módulo Nodo Pa11y CI

Pacto, Pacto-Pacto

Modelode objetos de página, Modelode objetos depágina-Modelo de objetos de página

PageSpeedInsights, PageSpeed Insights-PageSpeedInsights

Pruebaspor pares, Pruebas por pares-Pruebas por pares

partición, clase de equivalencia, Partición de clase de equivalencia

Norma de Seguridad de Datos del Sector de las Tarjetas de Pago (PCI DSS) , PCI DSS y PSD2

Directiva de Servicios de Pago (PSD2), PCIDSS y PS D2-PCIDSS y PSD2

PCI DSS(Norma de Seguridad de Datos del Sector de las Tarjetas de Pago (PCI DSS), PCI DSS y PSD2

patrón de carga pico-descanso, **Tipos de patrones de carga**
pruebas de penetración (pen), **Estrategia de pruebas de seguridad**
perceptibilidad, accesibilidad, **Perceptible**
capa de percepción, IoT (Internet de las Cosas), **Introducción a la arquitectura de cinco capas del IoT**
rendimiento

- diseño de la arquitectura y, **Factores que afectan al rendimiento de las aplicaciones**
- Pruebas CFR, **Estrategia de pruebas CFR**, Rendimiento
- complejidad del código y, **Factores que afectan al rendimiento de la aplicación**
- bases de datos y, **Factores que afectan al rendimiento de las aplicaciones**
- geolocalización y, **Factores que afectan al rendimiento de las aplicaciones**
- infraestructura y, **Factores que afectan al rendimiento de las aplicaciones**
- latencia de la red y, **Factores que afectan al rendimiento de las aplicaciones**
- pila tecnológica y, **Factores que afectan al rendimiento de la aplicación**
- componentes de terceros y, **Factores que afectan al rendimiento de la aplicación**

Pruebas de Rendimiento, Diez Habilidades Full Stack Testing,
Pruebas de Rendimiento

- Apache Benchmark, [Apache Benchmark](#)
- pruebas de rendimiento de backend, El rendimiento, las ventas y los fines de semana libres iestán correlacionados!
 - objetivos de rendimiento, Objetivos de Rendimiento Simples
- aplicaciones blockchain, Pruebas de aplicaciones blockchain
- Chrome DevTools, [Chrome DevTools-ChromeDevTools](#)
- pruebas de datos y, [Estrategia de pruebas de datos](#)
- ejercicios
 - preparación de datos, [Pasos 3-5: Preparar los datos, el entorno y las herramientas-Paso6: Guionizar los casos de prueba y ejecutarlos con JMeter](#)
 - preparación del entorno, [Pasos 3-5: Preparar los datos, el entorno y las herramientas-Paso6: Guionizar los casos de prueba y ejecutarlos con JMeter](#)
 - KPI objetivo, [Paso 1: Definir los KPI objetivo-Paso 1: Definir los KPI objetivo](#)
 - guionización de casos de prueba, [Paso 6: Guioniza los casos de prueba y ejecútalos utilizando JMeter-Integraciónen CI](#)
 - casos de prueba, [Paso 2: Definir los casos de prueba](#)
 - casos de prueba, JMeter y, [Paso 6: Guioniza los casos de prueba y ejecútalos utilizando JMeter-Integraciónen CI](#)
 - preparación de herramientas, [Pasos 3-5: Preparar los datos, el entorno y las herramientas-Paso6: Guionizar los casos de prueba y ejecutarlos con JMeter](#)
- frontend, Bloques de construcción de pruebas de rendimiento del frontend-Bloques de construcción de pruebas de rendimiento

del frontend

- almacenamiento en caché del navegador, Factores que afectan al rendimiento del frontend
- CDN (redes de distribución de contenidos), Factores que afectan al rendimiento del frontend
- complejidad del código, factores que afectan al rendimiento del frontend
- transferencias de datos, factores que afectan al rendimiento del módulo de acceso
- búsquedas DNS, factores que afectan al rendimiento del módulo de acceso
- latencia de la red, Factores que afectan al rendimiento del Frontend
- Gatling, [Gatling](#)
- objetivos, [Objetivos simples de rendimiento](#)
- Aplicaciones IoT (Internet de las Cosas), [Prueba de aplicaciones IoT](#)
- KPI (indicadores clave de rendimiento)
 - Pruebas de carga/volumen, [Tipos de pruebas de rendimiento](#)
 - Pruebas de inmersión, [Tipos de pruebas de rendimiento](#)
 - Pruebas de estrés, [Tipos de pruebas de rendimiento](#)
- duración, [Paso 6: Guión y ejecución de las pruebas de rendimiento mediante herramientas](#)
- patrones de carga

- patrón pico-descanso, [Tipos de patrones de carga](#)
- patrón de aceleración constante, [Tipos de patrones de carga](#)
- patrón de aumento escalonado, [Tipos de patrones de carga](#)
- aplicaciones móviles
 - [APId de rendimiento de Appium](#), [APId de rendimiento de AppiumAPI de rendimiento de Appium](#)
 - Mono, [Mono](#)
 - estrangulador de red, [Controles ampliados: Estrangulador de red](#)
- estrategia de pruebas móviles, [Pruebas de rendimiento-Pruebas de rendimiento](#)
 - Android, [Android](#)
- PageSpeed Insights, [PageSpeed Insights-PageSpeed Insights](#)
- [Modelo RAIL](#), [Modelo RAIL](#)
- pruebas de desplazamiento a la izquierda
 - fase de desarrollo, [Estrategia de pruebas de rendimiento](#)
 - en CI, [Estrategia de pruebas de rendimiento](#)
 - fase de planificación, [Estrategia de Pruebas de Rendimiento](#)
 - fase de pruebas de lanzamiento, [Estrategia de Pruebas de Rendimiento](#)
 - fase de prueba de la historia de usuario, [Estrategia de Pruebas de Rendimiento](#)
- pasos

- herramientas APM, Paso 5: Integrar las herramientas APM
 - preparación del entorno, Paso 3: Preparar el entorno de pruebas de rendimiento
 - scripting, Paso 6: Scripting y ejecución de las pruebas de rendimiento mediante herramientas
 - KPI objetivo, Paso 1: Definir los KPI objetivo
 - casos de prueba, Paso 2: Definir los casos de prueba
 - datos de prueba, Paso 4: Preparar los datos de prueba
 - herramientas, Paso 6: Guión y ejecución de las pruebas de rendimiento con herramientas
- estrategias, Estrategia de pruebas de rendimiento-Estrategia de pruebas de rendimiento

ataques de phishing, Phishing

densidad de píxeles, dispositivos móviles, Dispositivos

plataformas como normas, Introducción a las pruebas en tecnologías emergentes

POM (Modelo de Objetos del Proyecto)

- Archivo XML, Maven-Maven

pruebas de portabilidad, Contenedores de pruebas

Postman, Chrome DevTools y Postman-ChromeDevTools y Postman

Herramienta de prueba de la API de Postman, Postman-Postman

predicados, SQL, Expresiones y predicados

priorización, habilidades blandas y, Las habilidades blandas ayudan a crear una mentalidad de calidad ante todo

privacidad

- prueba de datos y, [Estrategia de prueba de datos](#)
- funcionalidades y, [Aspectos interfuncionales](#)

aplicaciones web progresivas (PWA), [App](#)

Modelo de objetos del proyecto (POM), [Maven](#)

- (ver también POM (Modelo de Objetos de Proyecto))

PSD2 (Directiva de Servicios de Pago), [PCIDSS y PS D2-PCIDSS y PSD2](#)

Titiritero, [BackstopJS](#)

PWA (aplicaciones web progresivas), [App](#)

Q

Qark, [Qark](#)

consultas, anidadas, [Consultas anidadas](#)

QuickTest, [Pruebas funcionales automatizadas](#)

R

Modelo RAIL, rendimiento del frontend y

- animación, [Modelo RAIL](#)
- inactivo, [modelo RAIL](#)
- carga, [Modelo RAIL](#)
- respuesta, [Modelo RAIL](#)

muestreo aleatorio, [Muestreo](#)

ransomware, [ransomware](#)

RASP (Autoprotección de Aplicaciones en Tiempo de Ejecución),
Estrategia de Pruebas de Seguridad

limitación de la tasa, comprender la aplicación

react-test-renderer, Pruebas instantáneas

Bases de datos relacionales, Bases de datos, Bases de datos, Bases de datos

pruebas de lanzamiento, accesibilidad, Pruebas manuales

fiabilidad, pruebas CFR, Estrategia de pruebas CFR, Fiabilidad, Fiabilidad

flujos de repetición, flujos de usuarios funcionales

replicación, Bases de datos

herramientas de análisis de informes, Análisis de informes de pruebas ReportPortal, Análisis de informes de pruebas

repudio, Repudio de acciones

análisis de requisitos, pruebas de shift-left y, pruebas de shift-left, pruebas de shift-left

Resemble.js, BackstopJS

API RESTful, Pruebas de API

Servicios RESTful, Introducción a los Micro y Macro Tipos de Pruebas uniones derechas, Uniones

robustez, accesibilidad, Robusto

Autoprotección de Aplicaciones en Tiempo de Ejecución (RASP),
Estrategia de Pruebas de Seguridad

Herramienta RXVP, pruebas **automatizadas**, Pruebas funcionales automatizadas

S

SaaS (software como servicio), IA visual, **Applitools Eyes**, una herramienta potenciada por IA

Safari, pruebas entre navegadores y, **Pruebas entre navegadores** muestreo, **Muestreo-Muestreo**

- criterios específicos, **Muestreo**
- muestreo aleatorio, **Muestreo**

SAST (Pruebas estáticas de seguridad de las aplicaciones), **Estrategia de pruebas de seguridad**, **Pruebas de seguridad**

Herramientas SCA (Análisis de la Composición del Software), **Estrategia de Pruebas de Seguridad**

Script Scala, pruebas de carga, **Gatling**
escalabilidad, **bases de datos**

SCCS (Sistema de Control del Código Fuente), **El Proceso CI/CT/CD**
lectores de pantalla, **Ejemplo: Lectores de pantalla-Ejemplo: Lectores de pantalla**

- TalkBack, **Android**
- VoiceOver, **iOS**

resolución de pantalla, **Dispositivos**

tamaño de pantalla, dispositivos móviles, **Dispositivos**

capturas de pantalla, capturas de pantalla de fallos, **Configuración y flujo de trabajo**

seguridad

- como hábito, **Perspectivas: La seguridad es un hábito**
- activos, **bloques de construcción**
- ataques, **bloques de construcción**
- blockchain y, **Blockchain**
- compromisos, **bloques de construcción**
- ciberataques
 - fuerza bruta, **Fuerza bruta**
 - forja de galletas, **Forja de galletas**
 - cryptojacking, **Criptopiratería**
 - phishing, **Suplantación de identidad**
 - ransomware, **Ransomware**
 - ingeniería social, **Ingeniería social**
 - raspado web, **Web scraping**
 - XSS (secuencias de comandos en sitios cruzados) ,
Secuencias de comandos en sitios cruzados
- prueba **dedatos** y, **Estrategia de prueba de datos**
- encriptación, **bloques de construcción**
- funcionalidades y, **Aspectos interfuncionales**
- hashing, **Bloques de construcción**
- amenazas, **bloques de construcción**
- vulnerabilidades, **Building Blocks**

Casos de prueba de seguridad, Casos de prueba de seguridad a partir del modelo de amenaza-Casos de prueba de seguridad a partir del modelo de amenaza

Pruebas de seguridad, Diez habilidades para pruebas full stack, Pruebas de seguridad

- vulnerabilidades de las aplicaciones
 - autenticación, Autenticación y mala gestión de la sesión
 - inyección de código, Inyección de código SQL
 - vulnerabilidades conocidas, Unhandled, Vulnerabilidades conocidas no gestionadas
 - configuración errónea, Configuración errónea de la aplicación
 - exposición de secretos, Exposición de secretos de la aplicación-Exposición de secretos de la aplicación
 - gestión de sesiones, Autenticación y gestión errónea de sesiones
 - inyección SQL, Inyección de código SQL-Inyección de código SQL
 - datos no encriptados, Datos privados no encriptados
 - XSS (secuencias de comandos en sitios cruzados), Secuencias de comandos en sitios cruzados
- aplicaciones blockchain, Prueba de aplicaciones blockchain
- ejercicios
 - OWASP Dependency-Check, Configuración y flujo de trabajo de OWASP Dependency-Check
 - OWASP ZAP, OWASP ZAP-Integración de ZAP con CI

- Aplicaciones IoT (Internet de las Cosas), Pruebas de aplicaciones IoT
- aplicaciones móviles
 - Escáner de accesibilidad, Escáner de accesibilidad
 - MobSF, MobSF-MobSF
 - Qark, Qark
- estrategia de pruebas móviles, Pruebas desseguridad-Pruebas de seguridad
- estrategias
 - DAST (Pruebas dinámicas de seguridad de aplicaciones), Estrategia de pruebas de seguridad
 - automatización de pruebas funcionales, estrategia de pruebas de seguridad
 - IAST (Pruebas Interactivas de Seguridad de las Aplicaciones), Estrategia de Pruebas de Seguridad
 - escaneado de imágenes, estrategia de pruebas de seguridad
 - pruebas exploratorias manuales, estrategia de pruebas de seguridad
 - pruebas de penetración (pen), Estrategia de pruebas de seguridad
 - RASP (Autoprotección de Aplicaciones en Tiempo de Ejecución), Estrategia de Pruebas de Seguridad
 - SAST (Pruebas estáticas de seguridad de las aplicaciones), Estrategia de pruebas de seguridad

- Herramientas SCA (Análisis de Composición de Software), Estrategia de Pruebas de Seguridad
- Complemento Synk IDE, Complemento Snyk IDE
- modelado de amenazas, El modelo de amenazas STRIDE
 - DDoS (denegación de servicio distribuida), Denegación de servicio
 - DoS (denegación de servicio), Denegación de servicio
 - escalada de privilegios, Escalada de privilegios
 - revelación de información, Revelación de información
 - manipulación de entradas, Manipulación de entradas
 - repudio, Repudio de acciones
 - suplantación de identidad, Suplantación de identidad

Selenium, Pruebas funcionales automatizadas

Selenium Grid, Configuración y flujo de trabajo

SeleniumWebDriver, Selenium WebDriver

- Clase Acciones, Selenium WebDriver
- API, Selenium WebDriver-SeleniumWebDriver
- componentes, Selenium WebDriver
- estrategia de espera explícita, Selenium WebDriver
- estrategia de espera fluida, Selenium WebDriver
- estrategia de espera implícita, Selenium WebDriver
- localizadores relativos, Selenium WebDriver

SEO (optimización de motores de búsqueda) , pruebas de rendimiento, **El rendimiento, las ventas y los fines de semana libres están correlacionados!**

Pruebas de servicio, Pruebas de servicio-Pruebas de servicio

- Java-REST Assured Framework, **Java-REST Assured Framework-Creación y flujo de trabajo**

gestión de **sesiones**, Autenticación y mala gestión de sesiones

Pruebas por turnos, Pruebas por turnos

- Desarrollo ágil y **Prueba de Turno a la Izquierda**
- fase de análisis, **Pruebas de Turno a la Izquierda**
- Pruebas automatizadas y, **Pruebas de Turno a la Izquierda**
- CI/CD y, **Pruebas de Turno a la Izquierda**
- pruebas de rendimiento
 - fase de desarrollo, **Estrategia de Pruebas de Rendimiento**
 - en CI, **Estrategia de Pruebas de Rendimiento**
 - fase de planificación, **Estrategia de Pruebas de Rendimiento**
 - fase de pruebas de lanzamiento, **Estrategia de Pruebas de Rendimiento**
 - fase de prueba de la historia de usuario, **estrategia de pruebas de rendimiento**
- inicio de la historia, **Prueba de Desplazamiento a la Izquierda**
- proceso de los tres amigos, **Prueba de Desplazamiento a la Izquierda**

simuladores, **Pruebas Exploratorias Manuales**

pruebas de humo, pruebas continuas, **estrategia de pruebas continuas**

Snapdeal, **Introducción al Full Stack Testing**

pruebas instantáneas, pruebas visuales y, **Pruebas Instantáneas-Pruebas Instantáneas**

Plugin Snyk JetBrains IDE, **Estrategia de pruebas de seguridad**

pruebas **de rendimiento**, **Tipos de pruebas de rendimiento**

- JMeter, **Diseño de otros casos de pruebas de rendimiento**

ingeniería social, Ingeniería social

habilidades **blandas**, Las habilidades blandasayudan a crear una mentalidad de calidad ante todo-Las habilidades blandasayudan a crear una mentalidad de calidad ante todo

Análisis de la Composición del Software (SCA)(ver herramientas SCA (Análisis de la Composición del Software))

Sony Playstation VR, **Realidad Aumentada y Realidad Virtual**

Sistema de Control del Código Fuente (SCCS), **El Proceso CI/CT/CD**

Suplantación de identidad, **Ejercicio de modelado de amenazas**

Spring Batch, **Sistemas de procesamiento por lotes**

Spring Data JPA, **Pruebas de integración**

SQL (Lenguaje de consulta estructurado), **Bases de datos, SQL, Filtrary agrupar-Filtrary agrupar**

- crear tablas, **Crear-Crear**
- eliminar, **Actualizar y eliminar**
- expresiones, **Expresiones y predicados**

- funciones, [Funciones y operadores](#)
- uniones, [Uniones](#)
- valores nulos, [Uniones](#)
- operadores, [Funciones y operadores](#)
- llenar tablas, [Insertar](#)
- predicados, [Expresiones y predicados](#)
- requisitos previos, [Requisitos previos](#)
- consultas, [Filtrado y agrupación](#)
 - consultas anidadas, [Consultas anidadas](#)
- lecturas, [Seleccionar](#)
- ordenar, [Ordenar](#)
- actualizaciones, [Actualizar y eliminar](#)

Inyección SQL, [Inyección de código o SQL-Inyección de código](#)

partes interesadas, habilidades [blandas](#), [Las habilidades blandas ayudan a crear una mentalidad de calidad ante todo](#)

transición de estado, [Transición de estado-Transición de estado](#)

Herramientas de pruebas estáticas de seguridad de las aplicaciones (SAST), [Estrategia de pruebas de seguridad](#)

patrón de carga de rampa constante, [Tipos de patrones de carga](#)

patrón de carga de rampa escalonada, [Tipos de patrones de carga](#)

inicio de la historia, [Prueba de Desplazamiento a la Izquierda](#)

Libro [de cuentos](#), [Libro de cuentos-Libro de cuentos](#)

flujos, [flujos de eventos](#)

pruebas de resistencia, [Tipos de pruebas de rendimiento](#)

Modelo [STRIDE](#), amenazas y, [El modelo STRIDE de amenazas](#)

stubs, Pruebas [de contrato](#), Pruebas de contrato

soportabilidad, pruebas CFR, [estrategia de pruebas CFR](#)

- pruebas de arquitectura, [Soportabilidad](#)
- analizador estático de código, [Soportabilidad](#)

[Plug-in IDESynk](#), [Plug-in IDE Snyk](#)

T

[Talisman](#), [Estrategia de pruebas de seguridad](#), [Talisman Pre-Commit Hook](#)-[TalismanPre-Commit Hook](#)

[TalkBack](#), [Android](#)

pila tecnológica, rendimiento y, [Factores que afectan al rendimiento de las aplicaciones](#)

tecnologías

- inteligencia aumentada, [Introducción a los ensayos en tecnologías emergentes](#)
- cosas conectadas, [Introducción a las Pruebas en Tecnologías Emergentes](#)
- interacción similar a la humana, [Introducción a los ensayos en tecnologías emergentes](#)
- plataformas como normas, [Introducción a las pruebas en tecnologías emergentes](#)

[Terraform](#), [Pruebas de infraestructura](#)

herramientas de creación [depruebas](#), [Creación de pruebas](#)

caso de prueba, **Prueba Exploratoria Manual**

higiene del entorno de pruebas, **Perspectivas: Higiene del entorno de pruebas**

- equipos autónomos, **Perspectivas: Higiene del entorno de pruebas**
- higiene de datos, **Perspectivas: Higiene del entorno de pruebas**
- Implementación y, **Perspectivas: Higiene del entorno de pruebas**
- compartido frente a dedicado, **Perspectivas: Higiene del entorno de pruebas**
- servicios de terceros, **Perspectivas: Higiene del entorno de pruebas**

herramientas de **gobernanza de pruebas**, **Gobernanza de pruebas**

herramientas de mantenimiento **de pruebas**, **Mantenimiento de pruebas**

pirámide de pruebas , **Estrategia de Pruebas Funcionales Automatizadas**

- aplicación web orientada a servicios, **Estrategia de Pruebas Funcionales Automatizadas**

herramientas de análisis **de informes de pruebas**, **Análisis de informes de pruebas**

ejecutores de pruebas, **TestNG**

trofeo de prueba forma de prueba, **Estrategia de Prueba Funcional Automatizada**

Test.ai, Creación de tests

Contenedores de prueba, Contenedores de prueba-Contenedores de prueba

TestCraft, Creación de pruebas

Testim, Creación de pruebas

Pruebas de pila completa para una alta calidad

- ADR (registros de decisiones de arquitectura), Prevención de defectos frente a detección de defectos
- específico de blockchain, Pruebas de aplicaciones blockchain-Pruebas de aplicaciones blockchain
- plataformas alojadas en la nube, Navegadores
- primeros principios
 - colaboración, La comunicación y la colaboración son la clave de la calidad
 - comunicación, La comunicación y la colaboración son la clave de la calidad
 - retroalimentación continua, Retroalimentación continua
 - prevención de defectos sobre detección, Prevención de defectos sobre detección de defectos-Prevención de defectos sobre detección de defectos
 - pruebas empáticas, Pruebas empáticas
 - retroalimentación rápida, Retroalimentación rápida
 - pruebas a nivel macro, Pruebas a nivel micro y macro
 - métricas, Medición de métricas de calidad-Medición demétricas de calidad
 - Pruebas demicronivel, Pruebas de micronivel y macronivel

- IPM (reuniones de planificación de iteraciones), Prevención de defectos en lugar de detección de defectos
- Pruebas deportabilidad, Contenedores de pruebas
- Pruebas de desplazamiento a la izquierda, Pruebas de desplazamiento a la izquierda
- Prueba por turnos a la izquierda y Prueba por turnos a la izquierda
- proceso de los tres amigos, Prevención de Defectos sobre Detección de Defectos
- inicio de la historia de usuario, Prevención de Defectos sobre Detección de Defectos

habilidades de prueba, Diez habilidades de prueba de pila completa

- Pruebas de accesibilidad, Diez habilidades de pruebas de pila completa
- Pruebas funcionales automatizadas, Diez Técnicas de Pruebas Completas
- Pruebas continuas, Diez competencias en pruebas Full Stack
- pruebas de requisitos interfuncionales, Diez competencias en pruebas Full Stack
- pruebas de datos, Habilidades de evaluación de diez pilas completas
- Pruebas exploratorias manuales, Diez Full Stack Testing Skills
- Pruebas móviles, Diez competencias en pruebas Full Stack
- Pruebas de rendimiento, Técnicas de Pruebas Full Stack 10
- Pruebas de seguridad, Diez competencias en pruebas Full Stack

- pruebas visuales, diez competencias Full Stack Testing

TestNG, Pruebas unitarias, TestNG, Configuración y flujo de trabajo

TestRail, Estrategia de Pruebas Funcionales Automatizadas

TFLint, Pruebas de Infraestructura

componentes de terceros, rendimiento y, Factores que afectan al rendimiento de las aplicaciones

modelado de amenazas, El modelo de amenazas STRIDE, Modelado de amenazas

- activos, Pasos del modelado de amenazas
- Pensamiento de sombrero negro, Pasos del modelado de amenazas
- DDoS (denegación de servicio distribuida), Denegación de servicio
- DoS (denegación de servicio), Denegación de servicio
- escalada de privilegios, Escalada de privilegios, Ejercicio de modelado de amenazas
- ejercicio de modelado de amenazas, Ejercicio de modelado de amenazas
- características, definición, Pasos del modelado de amenazas
- revelación de información, Revelación de información, Ejercicio de modelado de amenazas
- Manipulación de entradas, Manipulación de entradas, Ejercicio de modelado de amenazas
- Priorización, Pasos del modelado de amenazas

- repudio, Repudio de acciones, Ejercicio de modelado de amenazas
- identidadsuplantada, Identidad suplantada, Ejercicio de modelado de amenazas
- pasos, Pasos del modelado de amenazas
- modelo STRIDE, El modelo de amenazas STRIDE
- casosdeprueba, Casos de prueba de seguridaddel modelo de amenazas-Casos de prueba de seguridaddel modelo de amenazas

amenazas, seguridad, Building Blocks

proceso de los tres amigos, Pruebas de Desplazamiento a la Izquierda, Prevención de Defectos sobre Detección de Defectos

transcripciones, accesibilidad, Perceptible

U

UAAG (Pautas de Accesibilidad del Agente de Usuario), Ecosistema de Accesibilidad

Entorno UAT, El Proceso CI/CT/CD

IU (interfaz de usuario)

- ecommerce UI, Introducción a los tipos de pruebas micro y macro
- apariencia, La apariencia de la IU

Pruebas funcionales deinterfaz de usuario,Pruebas funcionales de interfaz deusuario-Pruebas funcionales de interfaz de usuario

- Java-Selenium WebDriver

- Maven, **Maven-Maven**
- Modelodeobjetosdepágina, **Modelode objetosde página**-
Modelo de objetos de página
- Requisitos previos, **Requisitos previos**
- Selenium WebDriver, **Selenium WebDriver**-
SeleniumWebDriver
- configuración, **Configuracióny flujo de trabajo -**
Configuracióny flujo de trabajo
- TestNG, **TestNG**
- JavaScript-Cypress, **Marco JavaScript-Cypress**
 - Cypress,Cypress-Cypress
 - requisitos previos, **Requisitos previos**
 - configuraciónyflujo de trabajo, **Configuracióny flujo de trabajo**-**Configuracióny flujo de trabajo**

Capa de interfaz de usuario, **Building Blocks**

- servicio de autenticación, **Building Blocks**

Pruebas automatizadas basadas en la interfaz de usuario,
Introducción a las pruebas visuales

UiAutomator, **Appium**

UN CRPD (Convención de las Naciones Unidas sobre los Derechos de las Personas con Discapacidad), **Pruebas de accesibilidad**

comprendibilidad, accesibilidad, **Comprensible**

datos no encriptados, vulnerabilidades de las aplicaciones y, **Datos privados no encriptados**

Pruebas unitarias, Pruebasunitarias-Pruebas unit arias

- JUnit, JUnit-Configuracióny flujo de trabajo
- pruebas visuales, Pruebas unitarias

Convención de las Naciones Unidas sobre los Derechos de las Personas con Discapacidad (CDPD), Pruebas de accesibilidad

Unity, Probando Aplicaciones AR/VR

pruebas de usabilidad

- Pruebas CFR, Estrategia de pruebas CFR
 - internacionalización, Usabilidad
 - localización, Usabilidad
 - experiencia de usuario, usabilidad
- Aplicaciones IoT (Internet de las cosas), Probar aplicaciones IoT
- UX (experiencia de usuario) y, Pruebas visuales

Pautas de Accesibilidad para el Agente de Usuario (UAAG), Ecosistema de Accesibilidad

flujo de usuarios, Pruebas Exploratorias Manuales

personasusuarias, accesibilidad, Accessibility User Persons-AccessibilityUser Persons

personas usuarias, pruebas exploratorias manuales y, Comprender la aplicación

inicio de la historia de usuario, Prevención de Defectos sobre Detección de Defectos

UX (experiencia de usuario), Pruebas por turnos

- Pruebas CFR, Usabilidad

- pruebas deusabilidad y, Pruebas visuales

V

VCS (sistema de control de versiones), El proceso CI/CT/CD

- beneficios, El proceso CI/CT/CD
- Git, El proceso CI/CT/CD

sistema de control deversiones (VCS)(ver VCS (sistema de control de versiones))

realidad virtual (RV)(ver RV (realidad virtual))

Visual AI, AppliTools Eyes, una herramienta potenciada por la IA

pruebas visuales, diez habilidades de pruebas full stack, pruebas visuales

- AppliTools Eyes, AppliTools Eyes , una herramienta potenciada por la IA, AppliTools Eyes, una herramienta potenciada por la IA
- Retos, Perspectivas: Retos del Testing Visual
- ceguera al cambio, Introducción al Testing Visual
- nivel de componente, Casos de uso críticos para el proyecto/empresa
- ejercicios
 - BackstopJS, BackstopJS-Flujo de trabajo
 - Cypress,Cypress-Flujo de trabajo
- estrategia de pruebas frontales, Estrategia de pruebas frontales
 - Pruebas de accesibilidad, Pruebas de accesibilidad

- pruebas cross-browser, pruebas cross-browser-pruebas cross-browser
- Pruebas de rendimiento del frontend, Pruebas de rendimiento del frontend
- pruebas funcionales de extremo a extremo, Pruebas funcionales de extremo a extremo
- pruebas de integración/componentes, Pruebas de integración/componentes-Pruebas de integración/componentes
- pruebas de instantáneas, Snapshot Tests -Pruebas de instantáneas
- pruebas unitarias, Pruebas unitarias
- pruebas visuales, pruebas visuales
- estrategia de pruebas móviles, Pruebas visuales
- casos de uso críticos para el proyecto/negocio, Casos de uso críticos para el proyecto/negocio-Casos de uso críticos para el proyecto/negocio
- Storybook, Storybook-Storybook
- consejos para la selección de herramientas, Perspectivas: Retos de las pruebas visuales
- frente a las pruebas instantáneas, Pruebas visuales

Lector de pantalla VoiceOver, iOS

RV (realidad virtual), Realidad Aumentada y Realidad Virtual

- Prueba de aplicaciones, Prueba de aplicaciones AR/VR-Prueba de aplicaciones AR/VR

- Google Cardboard, **Realidad Aumentada y Realidad Virtual**
 - HMD (pantalla montada en la cabeza), **Realidad Aumentada y Realidad Virtual**
 - HTC VIVE, **Realidad Aumentada y Realidad Virtual**
 - Oculus Go, **Realidad Aumentada y Realidad Virtual**
 - Oculus Quest, **Realidad Aumentada y Realidad Virtual**
 - Sony Playstation VR, **Realidad Aumentada y Realidad Virtual**
- vulnerabilidades, seguridad, **Building Blocks**
- **vulnerabilidades conocidas no gestionadas**

W

W3C (Consorcio de la World Wide Web), **Pruebas de accesibilidad**
WAI (Iniciativa de Accesibilidad a la Web), **Pruebas de accesibilidad**
WAI-ARIA (Aplicaciones ricas de Internet accesibles de la WAI),
Robusto

Herramienta WAVE de evaluación de la accesibilidad, **WAVE-Workflow**

WCAG (Pautas de Accesibilidad al Contenido en la Web), **Ecosistema de Accesibilidad**

- principios rectores, **WCAG 2.0: Principios rectores y niveles**
- Nivel A, **WCAG 2.0: Principios rectores y niveles**
 - requisitos, **Normas de conformidad de nivel A-Robusto**
- Nivel AA, **WCAG 2.0: Principios y niveles rectores**
- Nivel AAA, **WCAG 2.0: Principios y niveles rectores**

Iniciativa de Accesibilidad a la Web (WAI), **Pruebas de accesibilidad**

Pautas de Accesibilidad del Contenido en la Web (WCAG) (ver WCAG (Pautas de Accesibilidad al Contenido en la Web))

raspado web, **Web scraping**

servicios web, **Pruebas API**

pruebas de interfaz de usuario web

- Navegadores, **Navegadores-Browsers**
- Imán para bichos, imán para bichos-imán para bichos
- **Chrome DevTools**, **Chrome DevTools-Chrome DevTools**

WebPageTest, **WebPageTest-Flujo de trabajo**

Herramienta de prueba de API **WireMock**, **WireMock-WireMock**

X

Inspector de accesibilidad de XCode, **iOS**

XCUITest, **Appium**

XP (Programación Extrema), **Pruebas por turnos**

XSS (secuencias de comandos en sitios cruzados), **Secuencias de comandos en sitios cruzados**, **Secuencias de comandos en sitios cruzados**

Z

Zerocode, **Apache Kafka** y **Zerocode**

- creación de pruebas, **Workflow-Workflow**

Sobre el autor

Gayathri Mohan es una apasionada líder tecnológica con experiencia en múltiples funciones de desarrollo de software y dominios técnicos e industriales. Gayathri ha demostrado su valía gestionando con éxito grandes equipos de control de calidad (QA) para clientes de Thoughtworks, donde ahora es consultora principal. Mientras trabajaba como SME global de QA de la empresa, definió las trayectorias profesionales y la estructura de desarrollo de habilidades deseada para los QA en Thoughtworks. Como directora técnica de oficina, Gayathri cultivó las comunidades técnicas locales, organizó eventos técnicos y desarrolló el liderazgo de pensamiento en temas técnicos.

Gayathri también es coautor de *Perspectives of Agile Software Testing*, publicado por Thoughtworks en el 10º aniversario de Selenium.

Colofón

El animal de la portada de *Full Stack Testing* es un tenrec rayado de las tierras bajas (*Hemicentetes semispinosus*). Estos pequeños mamíferos insectívoros son una de las muchas especies de tenrecs de la isla de Madagascar. Los tenrecs rayados de tierras bajas suelen encontrarse en matorrales, selvas tropicales de tierras bajas, tierras agrícolas e incluso en algunos jardines rurales de la parte oriental de la isla.

Los tenrecs rayados de llanura se identifican fácilmente por su hocico negro, largo y puntiagudo, y su cuerpo pequeño y sin cola, rayado con púas negras y amarillas. Una cresta de espinas amarillas les cubre la nuca. Sus púas son desmontables y pueden utilizarse como mecanismo de defensa; los tenrecs también las utilizan para comunicarse frotándolas entre sí, lo que produce un sonido agudo. Los tenrecs rayados de llanura completamente adultos miden entre 15 y 20 cm y pesan entre 1,5 y 1,5 kg.

Los tenrecs rayados de llanura son sociables y se reúnen en grupos de hasta 20 individuos. Cavan madrigueras conectadas para anidar y buscan lombrices e insectos individualmente o en pequeños grupos. En invierno, entran en torpor, un estado de temperatura corporal reducida y metabolismo disminuido. Las hembras sólo son fértiles durante un año y son reproductivamente activas a los 25 días de vida, lo que las convierte en la única especie de tenrec que puede criar en la misma estación en la que nació. El tenrec rayado de llanura está clasificado como especie menos preocupante por la UICN debido a su amplia distribución, gran abundancia y alta tolerancia a zonas con gran número de humanos. Muchos de los animales de la cubierta de O'Reilly están en peligro; todos ellos son importantes para el mundo.

La ilustración de la portada es de Karen Montgomery, basada en un grabado en blanco y negro de *la English Cyclopedia*. Los tipos de letra de la portada son Gilroy Semibold y Guardian Sans. La fuente del texto es Adobe Minion Pro; la fuente del encabezamiento es

Adobe Myriad Condensed; y la fuente del código es Ubuntu Mono de Dalton Maag.