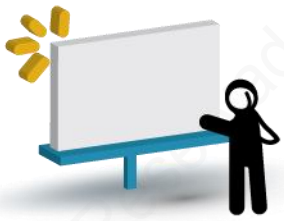
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Professional Testing Master

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Unidad 4: Testing de aplicaciones Web



Presentación:

En esta Cuarta Unidad del curso, nos adentraremos en las particularidades de la estrategia de testing para el caso de aplicaciones web.

Objetivos:

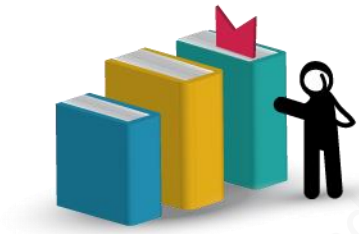
Al terminar la Unidad los participantes:

Habrán comprendido las dificultades propias del testing de aplicaciones web.

Se habrán familiarizado con el proceso general de testing en aplicaciones web.

Estarán en condiciones de aplicar técnicas de testing aprendidas anteriormente como las pruebas de caja negra y caja blanca adaptadas al testing web.

Conocerán aspectos nuevos a tener en cuenta siempre que se realice el testing de una aplicación web.



Bloques temáticos:

1. Fundamentos de testing web
2. Prueba de contenido
3. Prueba de interfaz
4. Prueba a nivel componente
5. Prueba de configuración
6. Prueba de seguridad
7. Prueba de performance

Contenido

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning colaborativo de nuestro Centro.***

Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1 Introducción

Existe una urgencia que siempre impregna un proyecto web. Los participantes (intranquilos por la competencia de otras aplicaciones web (*webapps*), presionados por las demandas del cliente y preocupados porque perderán la ventana de mercado) fuerzan para poner la *webapp* en línea. Como consecuencia, en ocasiones prestan poca atención a las actividades técnicas que ocurren últimas en el proceso, como las pruebas de la aplicación web. Esto puede ser un error catastrófico. Para evitarlo, los miembros del equipo deben asegurarse de que cada producto resultante muestre alta calidad.

Las pruebas no deben esperar hasta que el proyecto finalice. Comience a probar antes de escribir una línea de código. Pruebe constante y efectivamente, y desarrollará un sitio web mucho más duradero.

Los modelos de requerimientos y de diseño no pueden probarse en el sentido clásico: por ello, el equipo debe realizar revisiones técnicas [Mye12] y pruebas ejecutables. La intención es descubrir y corregir errores antes de que la *webapp* esté disponible para sus usuarios finales.

2 Fundamentos de testing en aplicaciones web

Probar es el proceso de ejecución del software con la intención de encontrar (y finalmente corregir) errores. Esta filosofía fundamental, que se expuso por primera vez en la Unidad 1, no cambia para las *webapps*. De hecho, puesto que los sistemas y las aplicaciones basadas en web residen en una red e interactúan con muchos sistemas operativos, navegadores (residentes en varios dispositivos), plataformas de hardware, protocolos de comunicaciones y aplicaciones backend diferentes, la búsqueda de errores representa un reto significativo.

Para entender los objetivos de las pruebas dentro de un contexto de ingeniería web, debe considerar las muchas dimensiones de calidad de la *webapp*. En el contexto de esta discusión, se consideran las dimensiones de calidad que son particularmente relevantes en cualquier análisis de las pruebas de la *webapp*. También se considera la naturaleza de

los errores que se encuentran como consecuencia de las pruebas y la estrategia de prueba que se aplica para descubrir dichos errores.

2.1 Dimensiones de calidad

La calidad se incorpora a una aplicación web como consecuencia de un buen diseño. Se evalúa aplicando una serie de revisiones técnicas que evalúan varios elementos del modelo de diseño y un proceso de prueba que se estudia a lo largo de este capítulo. Tanto las revisiones como las pruebas examinan una o más de las siguientes dimensiones de calidad:

- El *contenido* se evalúa tanto en el nivel sintáctico como en el semántico. En el primero, se verifica vocabulario, puntuación y gramática para documentos basados en texto. En el segundo, se evalúa la corrección (de la información presentada), la consistencia (a través de todo el objeto de contenido y de los objetos relacionados) y la ausencia de ambigüedad.
- La *función* se prueba para descubrir errores que indican falta de conformidad con los requerimientos del cliente. Cada función de la *webapp* se evalúa en su corrección, inestabilidad y conformidad general con estándares de implementación adecuados (por ejemplo, estándares de lenguaje Java o AJAX).
- La *estructura* se evalúa para garantizar que entrega adecuadamente el contenido y la función de la aplicación, que es extensible y que puede soportarse según se agregue nuevo contenido o funcionalidad.
- La *usabilidad* se prueba para asegurar que la interfaz soporta a cada categoría de usuario y que puede aprender y aplicar toda la sintaxis y semántica de navegación requerida.
- La *navegabilidad* se prueba para asegurar que toda la sintaxis y la semántica de navegación se ejecutan para descubrir cualquier error de navegación (por ejemplo, links rotos, inadecuados y erróneos).
- La *performance* se prueba bajo condiciones operativas, configuraciones y cargas diferentes a fin de asegurar que el sistema responde a la interacción con el usuario y que maneja la carga extrema sin degradación operativa inaceptable.
- La *compatibilidad* se prueba al ejecutar la *webapp* en varias configuraciones de host, tanto en el cliente como en el servidor. La intención es encontrar errores que sean específicos de una configuración de host particular.

- La *interoperabilidad* se prueba para garantizar que la *webapp* tiene interfaz adecuada con otras aplicaciones y/o bases de datos.
- La *seguridad* se prueba al evaluar las vulnerabilidades potenciales e intentar explotar cada una. Cualquier intento de penetración exitoso se estima como un fallo de seguridad.

La estrategia y las tácticas para probar las *webapps* se desarrollaron a fin de verificar cada una de estas dimensiones de calidad y se estudian más adelante en esta Unidad.

2.2 Errores dentro de un entorno web

Los errores que se encuentran como consecuencia de una prueba exitosa de una *webapp* tienen algunas características únicas:

1. Puesto que muchos tipos de pruebas de *webapps* descubren problemas que se evidencian primero en el lado del cliente (es decir, mediante una interfaz implementada en un navegador específico o en un dispositivo de comunicación personal), con frecuencia se ve un síntoma del error, no el error en sí.
2. Puesto que una *webapp* se implanta en algunas configuraciones distintas y dentro de diferentes entornos, puede ser difícil o imposible reproducir un error fuera del entorno en el que originalmente se encontró.
3. Dado que las *webapps* residen dentro de una arquitectura cliente-servidor, los errores pueden ser difíciles de rastrear a través de tres capas arquitectónicas: el cliente, el servidor o la red en sí.
4. Algunos errores se deben al *entorno operativo estático* (es decir, a la configuración específica donde se realiza la prueba), mientras que otros son atribuibles al entorno operativo dinámico (es decir, a la carga de recurso instantánea o a errores relacionados con el tiempo).

Estos cuatro atributos de error sugieren que el entorno juega un importante papel en el diagnóstico de todos los errores descubiertos durante la prueba de *webapps*. En algunas situaciones (por ejemplo, la prueba de contenido), la ubicación del error es obvia, pero en muchos otros tipos de prueba de *webapps* (por ejemplo, prueba de rendimiento, prueba

de seguridad), la causa subyacente del error puede ser considerablemente más difícil de determinar.

2.3 Estrategia de prueba

La estrategia para probar *webapps* adopta los principios básicos de todas las pruebas de software (Unidad 1) y aplica una estrategia y las tácticas que se recomendaron para los sistemas orientados a objetos (Unidad 3). Los siguientes pasos resumen el enfoque:

1. El modelo de contenido para la *webapp* a se revisa a fin de descubrir errores.
2. El modelo de interfaz se examina para garantizar que se satisfacen todos los casos de uso.
3. El modelo de diseño para la *webapp* se revisa para descubrir errores de navegación.
4. La interfaz de usuario se prueba para descubrir errores en el mecanismo de presentación y/o navegación.
5. Los componentes funcionales se someten a prueba unitaria.
6. Se prueba la navegación a lo largo de toda la arquitectura.
7. La *webapp* se implementa en varias configuraciones de entorno diferentes y se prueba para asegurar la compatibilidad con cada configuración.
8. Las pruebas de seguridad se realizan con la intención de explotar las vulnerabilidades en la *webapp* o dentro de su entorno.
9. Se realizan pruebas de performance (rendimiento).
10. La *webapp* se prueba con una población controlada y monitoreada de usuarios finales; los resultados de su interacción con el sistema se evalúan para detectar errores de contenido y de navegación, cuestiones de usabilidad y compatibilidad, y seguridad, confiabilidad y rendimiento de la *webapp*.

Puesto que muchas *webapps* evolucionan continuamente, el proceso de prueba es una actividad siempre en marcha que realiza el personal de soporte web, quien usa pruebas de regresión derivadas de las pruebas desarrolladas cuando comenzó la ingeniería de las *webapps*.

2.4 Planificación de pruebas

Excepto por el más simple de los sitios web, rápidamente resulta claro que es necesaria alguna especie de planificación de pruebas. Con demasiada frecuencia, el número inicial de errores encontrados a partir de una prueba *ad hoc* es suficientemente grande como para que no todos se corrijan la primera vez que se detectan. Esto impone una carga adicional sobre el personal que prueba sitios y *webapps*. No sólo deben idear nuevas pruebas imaginativas, sino que también deben recordar cómo se ejecutaron las pruebas anteriores con la finalidad de volver a probar de manera confiable el sitio / *webapp*, y garantizar que se removieron los errores conocidos y que no se introdujeron nuevos.

Las preguntas que deben plantearse son: ¿cómo se "idean nuevas pruebas imaginativas" y sobre qué deben enfocarse dichas pruebas? Las respuestas a estas preguntas se integran en un plan de prueba que identifica: 1) el conjunto de tareas que se van a aplicar cuando comiencen las pruebas, 2) los productos de trabajo que se van a producir conforme se ejecuta cada tarea de prueba y 3) la forma en la que se evalúan, registran y reutilizan los resultados de la prueba cuando se realizan pruebas de regresión. En algunos casos, el plan de prueba se integra con el plan del proyecto. En otros, es un documento separado.

3 Un panorama del proceso de prueba

El proceso de prueba de *webapps* comienza con pruebas que verifican la funcionalidad del contenido y la interfaz que son inmediatamente visibles para el usuario final. Conforme avanza la prueba, se verifican aspectos de la arquitectura del diseño y de la navegación. Finalmente, la atención se centra en las pruebas que examinan las capacidades tecnológicas que no siempre son evidentes para los usuarios finales: los temas de infraestructura e instalación/implantación de la *webapp*.

La Figura 3-1 asocia el proceso de prueba de la *webapp* con la pirámide de diseño para este tipo de aplicaciones. Observe que, conforme el flujo de la prueba avanza de izquierda a derecha y de arriba abajo, los elementos visibles para el usuario del diseño de

la *webapp* (elementos superiores de la pirámide) se prueban primero, seguidos por los elementos de diseño de infraestructura.

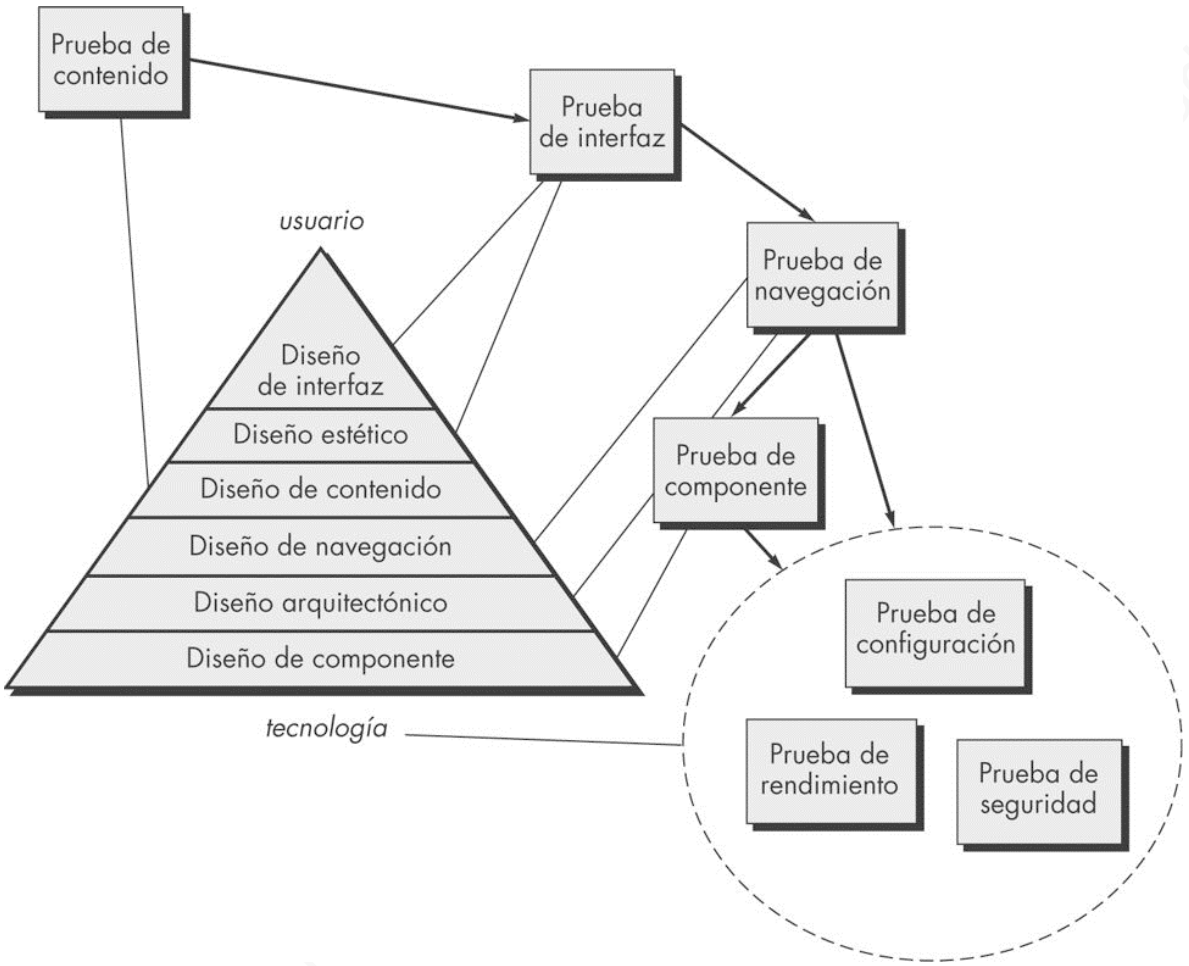


Figura 3-1. El proceso de prueba

4 Prueba de contenido

Los errores en el contenido de la *webapp* pueden ser tan triviales como errores tipográficos menores o tan significativos como información incorrecta, organización inadecuada o violación de leyes de la propiedad intelectual. La *prueba de contenido* intenta descubrir éstos y muchos otros problemas antes de que el usuario los encuentre.

La prueba de contenido combina tanto revisiones como generación de casos de prueba ejecutables.

Las revisiones se aplican para descubrir errores semánticos en el contenido (sección 4.1). Debe realizarse la revisión del contenido para garantizar que éste tiene calidad.

Por otro lado, las pruebas ejecutables (sección 4.2) se usan para descubrir errores de contenido generado dinámicamente a partir de datos adquiridos de una o más bases de datos. Esto se conoce como Prueba de base de datos.

4.1 Revisión de contenido

La revisión de contenido tiene tres objetivos importantes: 1) descubrir errores sintácticos (por ejemplo, errores tipográficos o gramaticales) en documentos de texto, representaciones gráficas y otros medios; 2) descubrir errores semánticos (es decir, errores en la precisión o completitud de la información) en cualquier objeto de contenido que se presente conforme ocurre la navegación y 3) encontrar errores en la organización o estructura del contenido que se presenta al usuario final.

Para lograr el primer objetivo, pueden usarse correctores automáticos de vocabulario y gramática. Sin embargo, muchos errores sintácticos evaden la detección de tales herramientas y los debe descubrir un revisor humano (examinador). De hecho, un sitio web grande debe considerar los servicios de un editor profesional para descubrir errores tipográficos, errores gramaticales, errores en la consistencia del contenido, errores en las representaciones gráficas y en referencias cruzadas.

La prueba semántica se enfoca en la información presentada dentro de cada objeto de contenido. El revisor (tester) debe responder las siguientes preguntas:

- ¿La información realmente es precisa?
- ¿La información es concisa y puntual?
- ¿El diseño del objeto de contenido es fácil de comprender para el usuario?
- ¿La información incrustada dentro de un objeto de contenido puede encontrarse con facilidad?

- ¿Se proporcionaron referencias adecuadas para toda la información derivada de otras fuentes?
- ¿La información presentada es consistente internamente y con la información presentada en otros objetos de contenido?
- ¿El contenido es ofensivo, confuso o abre la puerta a demandas?
- ¿El contenido infringe derechos de autor o nombres comerciales existentes?
- ¿El contenido incluye links internos que complementan el contenido existente? ¿Los links son correctos?
- ¿El estilo estético del contenido entra en conflicto con el estilo estético de la interfaz?

4.2 Prueba de base de datos

Las *webapps* modernas hacen mucho más que presentar objetos de contenido estáticos. Hoy en día, las *webapp* tienen conexión con sofisticados sistemas de gestión de bases de datos y construyen objetos de contenido dinámico que se crean en tiempo real, usando los datos adquiridos desde dicha base de datos.

Por ejemplo, una *webapp* de servicios financieros puede producir información compleja basada en texto, tablas y gráficos sobre un activo específico (por ejemplo, una acción o fondo de inversión). El objeto de contenido compuesto que presenta esta información se crea de manera dinámica después de que el usuario hace una solicitud de información sobre el activo específico. Para lograrlo, se requieren los siguientes pasos: 1) consulta a una gran base de datos de activos, 2) extracción de datos relevantes de la base de datos, 3) organización de los datos extraídos como un objeto de contenido y 4) transmisión de este objeto de contenido al entorno del cliente para su visualización. Los errores pueden ocurrir, y ocurren, como consecuencia de cada uno de estos pasos. El objetivo de la prueba de la base de datos es descubrir dichos errores, pero esta prueba es complicada por varios factores:

1. *El lado cliente solicita información que rara vez se presenta en una forma –ej., lenguaje de consulta estructurado (SQL)- que pueda ingresarse a un sistema de gestión de base de datos (DBMS)*. Por tanto, las pruebas deben diseñarse para descubrir errores cometidos al traducir la solicitud del usuario a la forma que puede procesar el DBMS.

2. *La base de datos puede ser remota en relación con el servidor que alberga la webapp.* En consecuencia, deben desarrollarse pruebas que descubran errores en la comunicación entre la *webapp* y la base de datos remota.
3. *Los datos brutos adquiridos de la base de datos deben transmitirse al servidor de la webapp y formatearse de manera adecuada para su posterior transmisión al cliente.* Por lo tanto, deben desarrollarse pruebas que demuestren la validez de los datos brutos recibidos por el servidor de la *webapp* y también deben crearse pruebas adicionales que demuestren la validez de las transformaciones aplicadas a los datos brutos para crear objetos de contenido válidos.
4. *El objeto de contenido dinámico debe transmitirse al cliente de forma que pueda desplegarse al usuario final.* Por ende, debe diseñarse una serie de pruebas para 1) descubrir errores en el formato del objeto de contenido y 2) probar la compatibilidad con diferentes configuraciones del entorno del cliente.

Al considerar estos cuatro factores, los métodos de diseño de casos de prueba deben aplicarse a cada una de las “capas de interacción” que se mencionan en la Figura 4-1. Las pruebas deben garantizar que 1) la información que pasa entre el cliente y el servidor desde la capa interfaz es válida, 2) la *webapp* procesa los guiones (scripts) de manera correcta y extrae o formatea adecuadamente los datos del usuario, 3) los datos del usuario pasan correctamente a una función de transformación de datos del lado servidor que formatea consultas adecuadas (por ejemplo, SQL) y 4) las consultas pasan a una capa de gestión de datos (ADO.NET, ODBC o JDBC, etc.) que se comunica con las rutinas de acceso a la base de datos (potencialmente ubicadas en otra máquina).

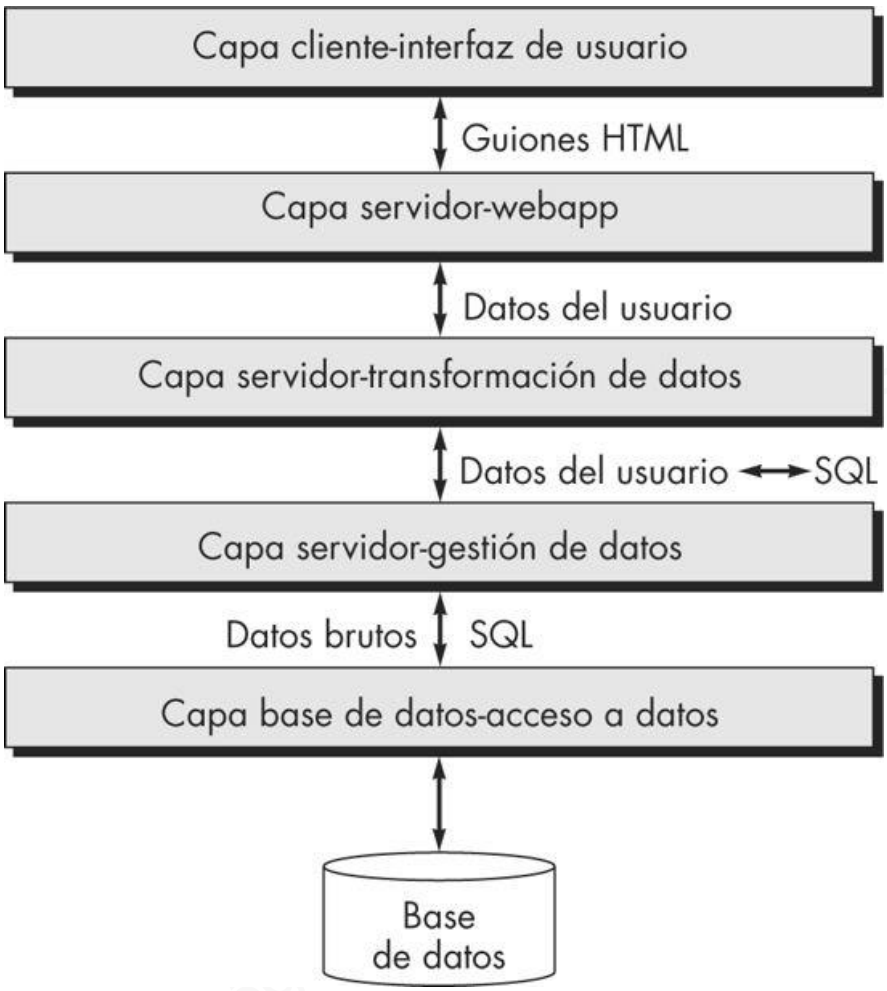


Figura 4-1. Capas de interacción

Las capas de transformación de datos, de gestión de datos y de acceso a base de datos que se muestran en la Figura 4-1, con frecuencia se construyen con componentes reutilizables que se validaron por separado y como paquete. Si éste es el caso, la prueba de *webapps* se enfoca en el diseño de casos de prueba para verificar las interacciones entre la capa cliente y las primeras dos capas servidor (*webapp* y transformación de datos) que se muestran en la figura.

La capa de interfaz de usuario se prueba para garantizar que los scripts se construyeron de manera adecuada para cada consulta de usuario y que transmiten adecuadamente al lado servidor. La capa *webapp* en el lado servidor se prueba para asegurar que los datos

de usuario se extraen de manera adecuada de los scripts y que se transmite adecuadamente a la capa de transformación de datos en el lado servidor. Las funciones de transformación de datos se prueban para asegurar que se creó el SQL correcto y que pasó a componentes de gestión de datos adecuados.

5 Prueba de interfaz de usuario

La verificación y validación de una interfaz de usuario de *webapp* ocurre en tres puntos distintos. Durante el análisis de requerimientos, el modelo de interfaz se revisa para garantizar que se da conformidad a los requerimientos de los participantes y a otros elementos del modelo de requerimientos. Durante el diseño, se revisa el modelo de diseño de interfaz para garantizar que se logran los criterios de calidad genéricos establecidos para todas las interfaces de usuario y que los temas de diseño de interfaz específicos de la aplicación se abordaron de manera adecuada. Durante la prueba, la atención se centra en la ejecución de aspectos específicos de la interacción con el usuario, conforme se manifiesten por la sintaxis y la semántica de la interfaz. Además, la prueba proporciona una evaluación final de la usabilidad.

Con excepción de especificaciones orientadas a *webapp*, la estrategia de prueba de interfaz que se comenta a continuación es aplicable a todo tipo de software cliente-servidor.

5.1 Estrategia de prueba de interfaz

La *prueba de interfaz* verifica los mecanismos de interacción y valida los aspectos estéticos de la interfaz de usuario. La estrategia global para la prueba de interfaz es 1) descubrir errores relacionados con mecanismos de interfaz específicos (por ejemplo, en la ejecución adecuada de un link de menú o en la forma como entran los datos en un formulario) y 2) descubrir errores en la forma como la interfaz implementa la semántica de navegación, la funcionalidad de la *webapp* o la visualización de contenido. Para lograr esta estrategia, se inician algunos pasos tácticos:

- *Las características de la interfaz se prueban para garantizar que las reglas del diseño, estética y contenido visual relacionado estén disponibles sin error para el usuario.* Las características incluyen tipo de fuente, uso de color, frames, imágenes, bordes, tablas y características de interfaz relacionadas que se generan conforme avanza la ejecución de la *webapp*.
- *Los mecanismos de interfaz individuales se prueban en forma análoga a la prueba unitaria.* Por ejemplo, las pruebas se diseñan para verificar todos los formularios, scripts del lado cliente, HTML dinámicos, scripts de servidor, contenido de *streaming* (transmisión continua) y mecanismos de interfaz específicos de la aplicación (por ejemplo, un carrito de compras para una aplicación de comercio electrónico). En muchos casos, la prueba puede enfocarse exclusivamente en uno de estos mecanismos (la "unidad") y excluir otras características y funciones de interfaz.
- *Cada mecanismo de interfaz se prueba dentro del contexto de un caso de uso para una categoría de usuario específica.* Este enfoque de pruebas es análogo a la prueba de integración porque las pruebas se realizan conforme los mecanismos de interfaz se integran para permitir la ejecución de un caso de uso.
- *La interfaz completa se prueba contra los casos de uso seleccionados a fin de descubrir errores en la semántica de la interfaz.* Este enfoque de prueba es análogo a la prueba de validación porque el propósito es demostrar conformidad con la semántica de casos de uso específicos. En esta etapa se lleva a cabo una serie de pruebas de usabilidad.
- *La interfaz se prueba dentro de varios entornos (por ejemplo, navegadores) para garantizar que será compatible.* En realidad, esta serie de pruebas también puede considerarse como parte de las pruebas de configuración.

5.2 Prueba de mecanismos de interfaz

Cuando un usuario interactúa con una *webapp*, la interacción ocurre a través de uno o más mecanismos de interfaz. En los párrafos que siguen se presenta un breve panorama de las consideraciones de prueba para cada mecanismo de interfaz.

5.2.1 Links

Cada link se prueba para garantizar que se alcanza el objetivo de contenido o función apropiados. Se construye una lista de todos los links asociados con el diseño de interfaz (por ejemplo, barras de menú e índices) y luego se ejecuta cada uno individualmente.

Además, deben verificarse los links dentro de cada objeto de contenido para descubrir URL o links defectuosos, con objetos de contenido o funciones inadecuadas. Finalmente, los links con *webapps* externas deben probarse en su precisión y también evaluarse para determinar el riesgo de que se vuelvan inválidos con el tiempo. La prueba de vínculos externos debe ocurrir durante la vida de la *webapp*. Parte de una estrategia de soporte debe ser la calendarización regular de pruebas de vínculos.

5.2.2 Formularios

En un nivel macroscópico, las pruebas se realizan para asegurarse de que 1) las etiquetas identifican correctamente los campos dentro del formulario y los campos obligatorios se identifican visualmente, 2) el servidor recibe toda la información contenida dentro del formulario y ningún dato se pierde en la transmisión entre cliente y servidor, 3) se usan valores por defecto adecuados cuando el usuario no selecciona de un menú desplegable o conjunto de botones, 4) las funciones del navegador (por ejemplo, la flecha "retroceso") no corrompen la entrada de datos en un formulario y 5) los scripts que realizan la comprobación de errores en los datos ingresados funcionan de manera adecuada y proporcionan mensajes de error explicativos.

En un nivel más específico, las pruebas deben garantizar que 1) los campos del formulario tienen ancho y tipos de datos adecuados, 2) el formulario establece protecciones adecuadas que prohíben que el usuario ingrese cadenas de texto más largas que cierto máximo predefinido, 3) todas las opciones adecuadas para menús desplegables se especifican y ordenan en forma clara para el usuario final, 4) las características de "autocompletado" (auto-fill) del navegador no conducen a errores en la entrada de datos y 5) la tecla de tabulación (o alguna otra) inicia el movimiento adecuado entre los campos del formulario.

5.2.3 Scripts en el lado cliente

Se realizan pruebas de caja negra para descubrir cualquier error en el procesamiento conforme se ejecuta el script. Estas pruebas con frecuencia se acoplan con pruebas de formularios porque la entrada del script con frecuencia se deriva de los datos proporcionados como parte del procesamiento de formulario. Debe realizarse una prueba de compatibilidad para garantizar que tanto el lenguaje de scripting elegido (por ejemplo JavaScript) como sus bibliotecas (ejemplo jQuery) funcionarán adecuadamente en las configuraciones de entorno que soporten la *webapp*. Además de probar el script en sí,

debe asegurarse de que los estándares de la compañía enuncien el lenguaje y versión preferidos a usar para la escritura de scripts en el lado cliente y en el lado servidor.

Las pruebas de script en el lado cliente y las pruebas asociadas con HTML dinámico podrían repetirse cuando se libera una nueva versión de un navegador popular.

5.2.4 HTML dinámico

Cada página web que contenga HTML dinámico (casi todas en la actualidad) se ejecuta para asegurar que la visualización dinámica es correcta. Además, debe llevarse a cabo una prueba de compatibilidad para asegurarse que el HTML dinámico funciona adecuadamente en las configuraciones de entorno que soportan la *webapp*.

5.2.5 Ventanas pop-up

Una serie de pruebas garantiza que 1) el pop-up tiene el tamaño y posición adecuadas, 2) no cubre la ventana de la *webapp* original, 3) el diseño estético es consistente con el diseño estético de la interfaz y 4) las barras de desplazamiento y otros mecanismos de control anexados a la ventana pop-up se ubican y funcionan de manera adecuada, como se requiere.

5.2.6 Scripts del lado del servidor

Las pruebas de caja negra se realizan con énfasis sobre la integridad de los datos (conforme los datos pasan al script del lado del servidor) y del procesamiento del script (una vez recibidos los datos validados). Además, la prueba de rendimiento puede realizarse para garantizar que la configuración del lado servidor puede satisfacer las demandas de procesamiento de múltiples invocaciones de los scripts. Esto se aplica a todas las tecnologías de procesamiento del lado del servidor tales como CGI, PHP, JSP, ASP.NET, Node.js, etc.

5.2.7 Contenido de streaming

Las pruebas deben demostrar que los datos de *streaming* están actualizados, que se visualizan de manera adecuada y que pueden suspenderse sin error y reanudarse sin dificultad.

5.2.8 Cookies

Se requieren pruebas tanto del lado servidor como del lado cliente. En el primero, las pruebas deben garantizar que una *cookie* se construyó adecuadamente (que contiene datos correctos) y que se transmitió de manera adecuada al lado cliente cuando se solicitó contenido o funcionalidad específico. Además, la persistencia adecuada de la *cookie* se prueba para asegurar que su fecha de expiración es correcta. En el lado cliente, las pruebas determinan si la *webapp* relaciona adecuadamente las *cookies* existentes a una solicitud específica (enviada al servidor).

5.2.9 Mecanismos de interfaz específicos de aplicación

Las pruebas se siguen de acuerdo a una lista de comprobación de funcionalidad y características que se definen mediante el mecanismo de interfaz. Por ejemplo, Splaine y Jaskiel [Spl01] sugieren la siguiente lista de comprobación para la funcionalidad carro de compras definida para una aplicación de comercio electrónico:

- Prueba de valor límite (Unidad 2) del número mínimo y máximo de artículos que pueden colocarse en el carro de compras.
- Prueba de una solicitud de “salida” para un carro de compras vacío.
- Prueba de borrado de un artículo del carro de compras.
- Prueba para determinar si una compra vacía el contenido del carro.
- Prueba para determinar la persistencia del contenido del carro de compras (esto debe especificarse como parte de los requerimientos del cliente).
- Prueba para determinar si la *webapp* puede recordar el contenido del carro de compras en alguna fecha futura (suponiendo que no se realizó compra alguna).

5.3 Prueba de usabilidad

La prueba de usabilidad evalúa el grado en el cual los usuarios pueden interactuar efectivamente con la *webapp* y el grado en el que la *webapp* guía las acciones del usuario, proporciona retroalimentación significativa y refuerza un enfoque de interacción

consistente. En lugar de enfocarse atentamente en la semántica de algún objetivo interactivo, las revisiones y pruebas de usabilidad se diseñan para determinar el grado en el cual la interfaz de la *webapp* facilita la vida del usuario.

Invariablemente, el ingeniero de software o tester contribuirá con el diseño de las pruebas de usabilidad, pero las pruebas en sí las realizan los usuarios finales. La siguiente secuencia de pasos es aplicable para tal fin [Spl01]:

1. Definir un conjunto de categorías de prueba de usabilidad e identificar las metas de cada una.
2. Diseñar pruebas que permitirán la evaluación de cada meta.
3. Seleccionar a los participantes que realicen las pruebas.
4. Instrumentar la interacción de los participantes con la *webapp* mientras se lleva a cabo la prueba.
5. Desarrollar un mecanismo para evaluar la usabilidad de la *webapp*.

La prueba de usabilidad puede ocurrir en varios niveles diferentes de abstracción: 1) puede evaluarse la usabilidad de un mecanismo de interfaz específico (por ejemplo, un formulario), 2) puede evaluarse la usabilidad de una página web completa (que abarque mecanismos de interfaz, objetos de datos y funciones relacionadas) y 3) puede considerarse la usabilidad de la *webapp* completa.

El primer paso en la prueba de usabilidad es identificar un conjunto de categorías de usabilidad y establecer los objetivos de la prueba para cada categoría. Las siguientes categorías y objetivos de prueba (escritos en forma de pregunta) ilustran este enfoque:

- *Interactividad*: ¿Los mecanismos de interacción (por ejemplo, menús desplegables, botones, punteros) son fáciles de entender y usar?
- *Diseño:* ¿Los mecanismos de navegación, contenido y funciones se colocan de forma que el usuario pueda encontrarlos rápidamente?
- *Legibilidad:* ¿El texto está bien escrito y es comprensible? ¿Las representaciones gráficas se entienden con facilidad?
- *Estética:* ¿El diseño, color, fuente y características relacionadas facilitan el uso? ¿Los usuarios se sienten cómodos con el “look and feel” de la *webapp*?

- *Características de visualización:* ¿La *webapp* usa de manera óptima el tamaño y la resolución de la pantalla?
- *Sensibilidad temporal:* ¿Las características, funciones y contenido importantes pueden usarse o adquirirse en forma oportuna?
- *Personalización:* ¿La *webapp* se adapta a las necesidades específicas de diferentes categorías de usuario o de usuarios individuales?
- *Accesibilidad:* ¿La *webapp* es accesible a personas que tienen discapacidades?

Dentro de cada una de estas categorías se diseña una serie de pruebas. En algunos casos, la "prueba" puede ser una revisión visual de una página web. En otros, pueden ejecutarse de nuevo pruebas semánticas de la interfaz, pero en esta instancia las preocupaciones por la usabilidad son primordiales.

Como ejemplo, considere la evaluación de usabilidad para los mecanismos de interacción e interfaz. Debe revisarse la siguiente lista de características de interfaz y probar la usabilidad: animación, botones, color, control, diálogo, campos, formularios, marcos, gráficos, etiquetas, vínculos, menús, mensajes, navegación, páginas, selectores, texto y barras de herramientas. Conforme se evalúa cada característica, es calificada por los usuarios que realizan la prueba sobre una escala cualitativa.

5.4 Prueba de compatibilidad

Diferentes computadoras, dispositivos de visualización, sistemas operativos, navegadores y velocidades de conexión de red pueden tener influencia significativa sobre la operación de una *webapp*. Cada configuración puede dar como resultado diferencias en velocidades de procesamiento en el lado cliente, en la resolución de visualización y en las velocidades de conexión. Los caprichos de los sistemas operativos en ocasiones pueden producir conflictos de procesamiento en la *webapp*. En ocasiones, diferentes navegadores producen resultados ligeramente distintos, sin importar el grado de estandarización HTML dentro de la *webapp*.

En algunos casos, pequeños conflictos de compatibilidad no representan problemas significativos, pero en otros pueden encontrarse serios errores. Por ejemplo, las velocidades de descarga pueden volverse inaceptables, carecer de un plug-in requerido puede hacer que el contenido no esté disponible, las diferencias de navegador pueden

cambiar dramáticamente el diseño de la página, los estilos de fuente pueden alterarse y volverse ilegibles o los formularios pueden organizarse de manera inadecuada. La *prueba de compatibilidad* busca descubrir dichos problemas antes de que la *webapp* esté en línea.

El primer paso en la prueba de compatibilidad es definir un conjunto de configuraciones de cómputo, y sus variantes, que "se encuentran comúnmente" en el lado cliente. En esencia, se crea una estructura de árbol, identificación de cada plataforma de cómputo, dispositivos de visualización frecuentes, sistemas operativos aceptados en la plataforma, navegadores disponibles, probables velocidades de conexión a internet e información similar. A continuación se deriva una serie de pruebas de validación de compatibilidad, con frecuencia adaptadas de pruebas de interfaz existentes, de rendimiento y de seguridad. La intención de estas pruebas es descubrir errores o problemas de ejecución que pueden rastrearse para identificar diferencias de configuración.

6 Prueba a nivel componente

La *prueba a nivel componente*, también llamada *prueba de función*, se enfoca en un conjunto de pruebas que intentan descubrir errores en funciones de las *webapps*. Cada función de una *webapp* es un componente de software (implementado en uno de varios lenguajes de programación o lenguajes de scripts) y puede probarse usando técnicas de caja negra (y en algunos casos de caja blanca), como se estudió en la Unidad 2.

Los casos de prueba en el nivel de componente con frecuencia se derivan de la entrada a formularios. Una vez definidos los datos de los formularios, el usuario selecciona un botón u otro mecanismo de control para iniciar la ejecución. Son usuales los siguientes métodos de diseño de diseño de caso de prueba (Unidad 2):

- *Partición de equivalencia*
- *Análisis de valor límite*
- *Prueba de rutas*

Además de estos métodos de diseño de casos de prueba, se usa una técnica llamada *prueba de error forzado* para derivar casos de prueba que a propósito conducen al componente web a una condición de error. El propósito es descubrir los errores que ocurren durante la manipulación del error (por ejemplo, mensajes de error incorrectos o inexistentes, falla de la *webapp* como consecuencia del error, salida errónea activada por entrada errónea, efectos colaterales que se relacionan con el procesamiento de componentes).

Cada caso de prueba en el nivel componente especifica todos los valores de entrada y salida que se espera que proporcione el componente. La salida real producida como consecuencia de la prueba se registra para futuras referencias durante el soporte y el mantenimiento.

En muchas situaciones, la ejecución correcta de la función de una *webapp* se relaciona a la interfaz adecuada con una base de datos que puede ser externa a la *webapp*. Por tanto, la prueba de base de datos se convierte en parte integral del régimen de prueba de componente.

7 Prueba de configuración

La variabilidad y la inestabilidad de la configuración son factores importantes que hacen de la prueba de *webapps* un desafío. El hardware, los sistemas operativos, navegadores, capacidad de almacenamiento, velocidades de comunicación de red y varios otros factores en el lado cliente son difíciles de predecir para cada usuario. Además, la configuración para un usuario dado puede cambiar de manera regular (por ejemplo, actualizaciones del sistema operativo, nuevos ISP y velocidades de conexión). El resultado puede ser un entorno del lado cliente que es proclive a errores sutiles y significativos. La impresión que un usuario tiene de la *webapp* y la forma en la que interactúa con ella puede diferir significativamente de la experiencia de otro usuario si ambos usuarios no trabajan dentro de la misma configuración en el lado cliente.

La labor de la prueba de configuración no es verificar toda configuración posible en el lado cliente. En vez de ello, es probar un conjunto de probables configuraciones en los lados cliente y servidor para garantizar que la experiencia del usuario será la misma en todos ellos y que aislará los errores que puedan ser específicos de una configuración particular.

7.1 Servidor

En el lado servidor, los casos de prueba de configuración se diseñan para verificar que la configuración servidor proyectada [es decir, servidor *webapp*, servidor de base de datos, sistemas operativos, software de firewall, aplicaciones concurrentes] pueden soportar la *webapp* sin error.

Para diseñar las pruebas, debe considerarse cada componente de la configuración del servidor. Entre las preguntas que deben plantearse y responderse durante la prueba se encuentran:

- ¿La *webapp* es completamente compatible con el sistema operativo (SO) servidor?
- ¿Los archivos de sistema, directorios y datos de sistema relacionados se crean correctamente cuando la *webapp* está operativa?
- ¿Las medidas de seguridad del sistema (por ejemplo, firewalls o encriptado) permiten a la *webapp* ejecutarse y atender a los usuarios sin interferencia o degradación del rendimiento?
- ¿La *webapp* se probó con la configuración de servidor distribuido (si existe alguno) que se eligió?
- ¿La *webapp* se integró adecuadamente con el software de base de datos? ¿La *webapp* es sensible a diferentes versiones del software de base de datos?
- ¿Los scripts de la *webapp* en el lado servidor se ejecutan adecuadamente?
- ¿Los errores del administrador del sistema se examinaron en sus efectos sobre las operaciones de la *webapp*?

7.2 Cliente

En el lado cliente, las pruebas de configuración se enfocan con más peso en la compatibilidad de la *webapp* con las configuraciones que contienen una o más combinaciones de los siguientes componentes:

- *Hardware:* CPU, memoria, almacenamiento, pantallas táctiles y dispositivos de impresión
- *Sistemas operativos:* Linux, Mac OS, Windows, Android, iOS
- *Software navegador:* Firefox, Safari, Internet Explorer, Edge, Chrome y otros
- *Componentes de interfaz de usuario:* HTML 5, JS, React y otros
- *Conectividad:* cable-módem, ADSL, Fibra óptica, WiFi, 3G, 4G LTE, 5G, etc.

Además de estos componentes, otras variables incluyen software de red, firewalls, caprichos del ISP[1] y aplicaciones que corren de manera concurrente.

Para diseñar pruebas, debe reducir el número de variables de configuración a un número manejable. Para lograr esto, cada categoría de usuario se evalúa para determinar las probables configuraciones que pueden encontrarse dentro de la categoría. Además, pueden usarse datos de participación de mercado para predecir las combinaciones de componentes más probables. Entonces la *webapp* se prueba dentro de estos entornos.

8 Prueba de seguridad

La seguridad de la *webapp* es un tema complejo que debe comprenderse por completo antes de que pueda lograrse una prueba de seguridad efectiva. Las *webapps* y los entornos en los lados cliente y servidor donde se alojan representan un blanco atractivo para *hackers* externos, empleados descontentos, competidores deshonestos y para quien quiera robar información sensible, modificar contenido maliciosamente, degradar el rendimiento, deshabilitar la funcionalidad o avergonzar a una persona, organización o negocio.

Las pruebas de seguridad se diseñan para sondear las vulnerabilidades del entorno lado cliente, las comunicaciones de red que ocurren conforme los datos pasan de cliente a servidor y viceversa, y el entorno del lado servidor. Cada uno de estos dominios puede atacarse, y es tarea del tester de seguridad descubrir las debilidades que puedan explotar quienes tengan intención de hacerlo.

[1] ISP: Internet Service Provider (proveedor de servicio de Internet)

En el lado cliente, las vulnerabilidades con frecuencia pueden rastrearse en errores preexistentes en navegadores o sistemas operativos.

Uno de los errores comúnmente mencionados es el desbordamiento de buffer (buffer overflow), que permite que código malicioso se ejecute en la máquina cliente. Por ejemplo, ingresar una URL en un navegador que es mucho más larga que el tamaño de buffer asignado para la URL provocará un error de sobreescritura de memoria (desbordamiento de buffer) si el navegador no tiene código de detección de error para validar la longitud de la URL ingresada. Un hacker experimentado puede explotar astutamente este error al escribir una URL larga con código que se va a ejecutar y que puede hacer que el navegador altere las configuraciones de seguridad (de alto a bajo) o, peor aún, corromper datos del usuario.

Otra vulnerabilidad potencial en el lado cliente es el acceso no autorizado a las c*ookies* colocadas dentro del navegador. Los sitios web creados con intenciones maliciosas pueden adquirir información contenida dentro de *cookies* legítimas y usar esta información en formas que ponen en riesgo la privacidad del usuario o, peor aún, que montan el escenario para el robo de identidad.

Los datos comunicados entre el cliente y el servidor son vulnerables al *spoofing* (engaño). El *spoofing* ocurre cuando un extremo de la ruta de comunicación se trastorna por una entidad con intenciones maliciosas. Por ejemplo, un usuario puede ser engañado por un sitio malicioso que actúa como si fuese el servidor de la *webapp* legítimo. La intención es robar contraseñas, información personal o datos de tarjetas de crédito.

En el lado servidor, la principal vulnerabilidad es el ataque de *inyección SQL*, que consiste en escribir código SQL como parte de los datos de una consulta que al ejecutarse revela información indebida (robo de datos o contraseñas). Otras vulnerabilidades incluyen ataques de negación de servicio (DoS, denial of service) y scripts maliciosos que pueden pasar hacia el lado cliente o usarse para deshabilitar operaciones del servidor o bien acceder a las bases de datos en el lado servidor.

Para proteger contra éstas (y muchas otras) vulnerabilidades, se implementa uno o más de los siguientes elementos de seguridad:

- *Sanitization:* saneamiento o filtrado de los datos (texto etc.) provenientes de formularios web y otras fuentes eliminando o deshabilitando cualquier código SQL. Se realiza en el código del servidor antes de procesarlos.

- *Firewall:* mecanismo de filtrado, que es una combinación de hardware y software que examina cada paquete de información entrante para asegurarse de que proviene de una fuente legítima y que bloquea cualquier dato sospechoso.
- *Autenticación:* mecanismo de verificación que valida la identidad de todos los clientes y servidores, y permite que la comunicación ocurra solamente cuando ambos lados se verifican.
- *Encriptado:* mecanismo de codificación que protege los datos sensibles al modificarlos de forma que hace muy difícil leerlos por quienes tienen intenciones maliciosas. El encriptado se fortalece usando *certificados digitales* que permiten al cliente verificar el destino al que se transmiten los datos.
- *Autorización:* mecanismo de filtrado que permite el acceso al entorno cliente o servidor sólo a aquellos individuos con códigos de autorización apropiados (por ejemplo, ID de usuario y contraseña).

Las pruebas de seguridad deben diseñarse para explorar cada una de estas tecnologías de seguridad con la intención de descubrir huecos en la seguridad.

El diseño de las pruebas de seguridad requiere conocimiento profundo del funcionamiento interno de cada elemento de seguridad y amplia comprensión de una gran gama de tecnologías de redes. En muchos casos, la prueba de seguridad se subcontrata con firmas que se especializan en dichas tecnologías.

9 Prueba de performance

Nada es más frustrante que una *webapp* que tarda minutos en cargar contenido cuando sitios de la competencia descargan contenido similar en segundos. Nada es más exasperante que intentar ingresar a una *webapp* y recibir un mensaje de “servidor ocupado”, con la sugerencia de que se intente de nuevo más tarde. Nada es más desconcertante que una *webapp* que responde instantáneamente en algunas situaciones y luego en otras parece caer en un estado de espera infinita. Estos eventos suceden en la web todos los días y todos ellos se relacionan con el rendimiento (performance).

Las *pruebas de performance* se usan para descubrir problemas de rendimiento que pueden ser resultado de: falta de recursos en el lado servidor, red con ancho de banda inadecuada, capacidades de base de datos inadecuadas, capacidades de sistema

operativo deficientes o débiles, funcionalidad de *webapp* pobremente diseñada y otros conflictos de hardware o software que pueden conducir a rendimiento cliente-servidor degradado. La intención es doble: 1) comprender cómo responde el sistema cuando aumenta la *carga* (es decir, número de usuarios, número de transacciones o volumen de datos global) y 2) recopilar mediciones que conducirán a modificaciones de diseño para mejorar el rendimiento.

9.1 Objetivos

Las pruebas de performance se diseñan para simular situaciones de carga del mundo real. A medida que aumenta el número de usuarios simultáneos de la *webapp* o el número de transacciones online o la cantidad de datos (descargados o subidos), las pruebas de performance ayudarán a responder las siguientes preguntas:

- ¿El tiempo de respuesta del servidor se degrada hasta un punto donde es apreciable e inaceptable?
- ¿En qué punto (en términos de usuarios, transacciones o carga de datos) el rendimiento se vuelve inaceptable?
- ¿Qué componentes del sistema son responsables de la degradación del rendimiento?
- ¿Cuál es el tiempo de respuesta promedio para los usuarios bajo diversas condiciones de carga?
- ¿La degradación del rendimiento tiene impacto sobre la seguridad del sistema?
- ¿La confiabilidad o precisión de la *webapp* resulta afectada a medida que crece la carga sobre el sistema?
- ¿Qué sucede cuando se aplican cargas que son mayores que la capacidad máxima del servidor?
- ¿La degradación del rendimiento tiene impacto sobre los ingresos de la compañía?

Para desarrollar respuestas a estas preguntas, se realizan dos tipos diferentes de pruebas de performance: 1) la *prueba de carga* examina la carga del mundo real para varios niveles de carga y en varias combinaciones, y 2) la *prueba de esfuerzo* fuerza a aumentar la carga hasta el punto de rompimiento para determinar cuánta capacidad puede manejar

el entorno de la *webapp*. Cada una de estas estrategias de prueba se considera en las secciones siguientes.

9.2 Prueba de carga

La intención de la prueba de carga (load test) es determinar cómo responderán las *webapps* y su entorno del lado servidor a varias condiciones de carga. A medida que avanzan las pruebas, las combinaciones de las siguientes variables definen un conjunto de condiciones de prueba:

N, número de usuarios concurrentes

T, número de transacciones online por unidad de tiempo

D, carga de datos procesados por el servidor en cada transacción

En cada caso, dichas variables se definen dentro de límites operativos normales del sistema. Cuando se aplica cada condición de prueba, se recopila una o más de las siguientes medidas: respuesta de usuario promedio, tiempo promedio para descargar una unidad estandarizada de datos y tiempo promedio para procesar una transacción. Estas medidas deben examinarse para determinar si una disminución abrupta en el rendimiento puede rastrearse en una combinación específica de *N*, *T* y *D*.

La prueba de carga también puede usarse para evaluar las velocidades de conexión recomendadas para los usuarios de la *webapp*. El rendimiento global, *P*, se calcula de la forma siguiente:

$$P = N \times T \times D$$

Tome como ejemplo un sitio de noticias deportivas. En un momento dado, 20000 usuarios concurrentes (es decir simultáneos) envían una solicitud (una transacción, *T*) en promedio una vez cada 2 minutos (120 segundos). Cada transacción requiere que la *webapp* descargue un nuevo artículo que en promedio “pesa” 3 KB (kilobytes) de tamaño. Por lo tanto, el rendimiento global puede calcularse como:

$$P = \frac{20000 * 1 * 3KB}{120s} = 500\frac{KB}{s}$$

Si ahora se tiene en cuenta que 1 byte = 8 bits, se tiene que:

$$P = 500\frac{KB}{s} = 500 * 8\frac{Kb}{s} = 4000\frac{Kb}{s}$$

Notar que *KB* es kilobyte y *Kb* es kilobit. A su vez 1000*Kb* = 1*Mb* (megabit). Entonces:

$$\boldsymbol{P = 4Mbps}$$

Por ende, la conexión de la red para el servidor tendría que soportar una velocidad (real) de ***4 megabits por segundo***, y debería ponerse a prueba para asegurarse de que lo hace.

9.3 Prueba de stress

La *prueba de stress (esfuerzo)* es una continuación de la prueba de carga, pero en esta instancia las variables *N*, *T* y *D* se establecen en valores normales y luego se superan los límites operativos. La intención de estas pruebas es responder a cada una de las siguientes preguntas:

- ¿El sistema se degrada “suavemente” o el servidor se apaga cuando la capacidad se supera?
- ¿El software servidor genera mensajes “servidor no disponible”? De manera más general, ¿los usuarios están conscientes de que no pueden llegar al servidor?
- ¿El servidor pone en cola los recursos solicitados y vacía la cola una vez que disminuye la demanda de capacidad?
- ¿Las transacciones se pierden conforme la capacidad se excede?
- ¿La integridad de los datos resulta afectada conforme la capacidad se excede?
- ¿Qué valores de *N*, *T* y *D* fuerzan el fallo del entorno servidor? ¿Cómo se manifiesta la falla? ¿Se envían notificaciones automáticas al personal de soporte técnico del servidor?
- Si el sistema falla, ¿cuánto tiempo tardará en regresar online?
- ¿Ciertas funciones de la *webapp* (por ejemplo, funcionalidad de cálculo intenso, capacidades de streaming de datos) quedan interrumpidas conforme la capacidad alcanza el nivel de 80 o 90 por ciento?

A una variación de las pruebas de stress en ocasiones se le conoce como *prueba pico/rebote* (*spike/bounce*). En este régimen de pruebas, la carga alcanza un pico de capacidad, luego se baja rápidamente a condiciones operativas normales y después alcanza de nuevo un pico. Al rebotar la carga del sistema, es posible determinar cuán bien el servidor puede ordenar los recursos para satisfacer una demanda muy alta y entonces liberarlos cuando reaparecen condiciones normales (de modo que esté listo para el siguiente pico).

Bibliografía utilizada y sugerida

Libros y otros manuscritos

- [Eve07] Everett, Gerald & McLeod, Raymond. Software Testing - Testing Across The Entire Software Development LifeCycle. 2007.
- [Far08] Farrell-Vinay, Peter. Manage Software Testing. 2008
- [Mye12] Myers, Glenford, Badgett, T. y Sandler, C. The Art of Software Testing. Third Edition. 2012.
- [Pre19] Pressman, Roger y Maxim, B. Software Engineering: A Practitioner's Approach. Ninth Edition. 2019.
- [Spl01] Splaine, S. y Jaskiel, S. The Web Testing Handbook. 2001.

Lo que vimos:

En esta Unidad nos adentramos en el estudio de las dificultades, las particularidades y las estrategias de testing para el caso particular de las aplicaciones web.

Lo que viene:

En la próxima Unidad completaremos el conocimiento de testing con conceptos, métodos y herramientas complementarias, incluyendo otros tipos de test y una introducción a las certificaciones más importantes.