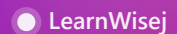


Autenticación e identidad de sesión

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Flujos de inicio de sesión, tratamiento de contraseñas, un SessionContext con una identidad verificada, tiempos de expiración, cierre de sesión, «recordarme» y fijación de sesión. Qué significa esto para un desarrollador que protege una aplicación Wisej.NET en producción, y cómo leer este módulo.



Enfoque en producción

Flujos de inicio de sesión, tratamiento de contraseñas, un SessionContext con una identidad verificada, tiempos de expiración, cierre de sesión, «recordarme» y fijación de sesión.

1. El problema en producción

AccessOps contiene fichas de empleados, documentos confidenciales y solicitudes de aprobación, así que la primera pantalla a la que llega cualquiera es también el primer control que importa. Todo lo que el resto del curso hace cumplir —permisos, reglas a nivel de registro, registros de auditoría— descansa sobre una única suposición: que el servidor sabe quién está al otro lado de esta sesión y puede demostrarlo. El Módulo 1 dibujó los límites de confianza; este módulo coloca una identidad verificada dentro del límite de sesión e impide que se adivine, se tome prestada o se prolongue más allá de su vida.

2. Conceptos clave

Concepto	Por qué importa
hash con sal y lento	Una derivación unidireccional de la contraseña con una sal aleatoria por usuario y un número de iteraciones deliberadamente costoso, para que una tabla robada no pueda revertirse y cada intento cueste trabajo real.
comparación en tiempo constante	Comparar dos hashes con un número fijo de operaciones mediante <code>CryptographicOperations.FixedTimeEquals</code> , para que el tiempo que tarda el servidor en responder nunca revele cuántos bytes de un intento eran correctos.
SessionContext	El objeto que se crea solo después de verificar una contraseña, con el id del empleado, el departamento, los roles y los datos del inicio de sesión, y que se guarda en <code>Application.Session</code> como el único lugar al que cualquier código pregunta quién es el usuario.
expiración de la sesión	El periodo de inactividad tras el cual Wisej.NET libera la sesión del lado del servidor y todo lo que contiene, configurado para la aplicación en <code>Default.json</code> en lugar de estimarse pantalla por pantalla.
fijación de sesión	Un ataque en el que el identificador que llevaba la página de inicio de sesión anónima lleva después también la sesión autenticada, de modo que un identificador plantado por el atacante se convierte en uno con sesión iniciada.
limitación de intentos de inicio de sesión	Contar los fallos por cuenta y por dirección del cliente y rechazar nuevos intentos durante un periodo de bloqueo, que es lo que convierte una máquina ilimitada de adivinar en unos pocos intentos por hora.
registro de auditoría del inicio de sesión	Una fila por intento con el nombre de usuario probado, la hora, la dirección del cliente y el resultado real: el lugar donde se escribe el motivo concreto del fallo, ya que la pantalla solo muestra uno genérico.

3. Desarrollo de la lección

Verificar una contraseña que nunca guardas

AccessOps no almacena contraseñas. Para cada empleado almacena tres valores: el algoritmo y el número de iteraciones usados, una sal aleatoria procedente de un generador criptográfico y los bytes derivados. Iniciar sesión ejecuta la misma derivación sobre la contraseña recién escrita, con la sal y el número de iteraciones de ese empleado, y compara el resultado con los bytes almacenados. No se descifra nada, porque nada se cifró: un hash va en un solo sentido, y la sal hace que dos empleados que eligieron la misma contraseña produzcan valores almacenados completamente distintos.

Aquí hay dos implementaciones adecuadas. PBKDF2 mediante `Rfc2898DeriveBytes` da un control explícito: SHA-256, una sal de 16 bytes, una salida de 32 bytes y un número de iteraciones lo bastante alto como para que una verificación tarde una fracción apreciable de segundo. El `PasswordHasher` de ASP.NET Core Identity, en cambio, empaqueta el marcador del algoritmo, el número de iteraciones, la sal y el hash en una sola cadena y devuelve un `PasswordVerificationResult` cuyo valor `SuccessRehashNeeded` te

indica que vuelvas a derivar con los parámetros actuales. En ambos casos el coste es lo que se busca: cada intento del atacante debe pagar una derivación completa.

La contraseña en sí vive lo que dura una llamada a un método. Se lee del `TextBox` de inicio de sesión, se pasa a `PasswordService.Verify` y se olvida: nunca es un campo de la ventana, nunca un valor de la fila de auditoría, nunca parte del mensaje de una excepción y nunca una línea de log, ni siquiera truncada.

```
public bool Verify(string password, StoredCredential stored)
{
    var candidate = Rfc2898DeriveBytes.Pbkdf2(
        Encoding.UTF8.GetBytes(password),
        stored.Salt,
        stored.Iterations,
        HashAlgorithmName.SHA256,
        stored.Hash.Length);

    return CryptographicOperations.FixedTimeEquals(candidate, stored.Hash);
}
```

Una identidad verificada, y solo una

Cuando `Verify` devuelve true, el servicio de inicio de sesión construye un `SessionContext`: el id del empleado, el nombre visible, el id del departamento, si esa persona tiene a alguien a su cargo, los roles resueltos desde la base de datos en ese momento, la hora de inicio de sesión y la dirección del cliente. Se guarda una sola vez en `Application.Session`, que es por sesión de navegador, así que nada pasa entre la responsable de operaciones y el administrativo de nóminas. Un campo estático lo compartirían todas las sesiones del proceso.

A partir de ahí es la única fuente de verdad. Las pantallas no guardan copia del id de usuario y los servicios no reciben el id del llamante como parámetro, porque un parámetro es algo que elige el código que llama: `DocumentService` lee el contexto por sí mismo. Cuando la cuadrícula de documentos muestra un departamento, ese departamento sale del contexto y no de un combo que se le podría haber pedido cambiar al navegador. Este es el hábito del curso llevado al código: una regla que solo vive en lo que ofrecía la interfaz no es una regla.

El contexto se crea después de la verificación y nunca antes. No hay un contexto anónimo con un id vacío esperando a rellenarse, porque un objeto que existe en ambos estados acabará siendo de confianza en el equivocado. Todas las pantallas de AccessOps salvo la de inicio de sesión exigen el contexto y devuelven el navegador cuando falta, y un accesor tipado que lanza una excepción cuando la sesión no contiene nada lo convierte en un hábito en lugar de una nota de revisión.

Expiración, cierre por inactividad y un Logout que cierra de verdad

Una sesión de Wisej.NET es estado del servidor, y su duración es configuración, no código. La expiración de la sesión se fija en el `Default.json` de la aplicación, y cuando vence el servidor libera la sesión junto con las ventanas, los controles y el `SessionContext` que contiene. Elige un valor que el negocio pueda defender —veinte minutos encajan en un portal que muestra bandas salariales y aprobaciones firmadas— y mantenlo por debajo de cualquier tiempo de espera del proxy situado delante de la aplicación, para que el comportamiento sea el que configuraste.

El cierre por inactividad es la mitad visible de la misma regla. Un `Wisej.Web.Timer` que se reinicia con la actividad del usuario avisa un minuto antes del límite y después ejecuta la misma limpieza que el botón Logout: borrar el contexto, descartar todo lo que haya en caché sobre la persona, escribir el registro de auditoría y llamar a `Application.Exit()`. Ambas vías llaman a un único método, porque dos rutinas de limpieza acaban divergiendo y la que nadie pulsa es la que olvida algo.

- La sesión termina en el servidor: cerrar la pestaña del navegador no es cerrar sesión.
- Todo estado de pantalla que nombraba a una persona desaparece antes de liberar la sesión.
- El navegador llega a la página de inicio de sesión, y retroceder en el historial no muestra datos.

Fuerza bruta, relleno de credenciales y fijación

Una página de inicio de sesión sin limitación es una máquina ilimitada de adivinar. `LoginThrottle` mantiene contadores por cuenta y por dirección del cliente; tras cinco fallos en un intervalo corto la cuenta se rechaza durante un periodo de bloqueo, y los contadores se reinician con un acierto. Da además a cada respuesta fallida la misma forma, no solo el mismo texto: cuando el nombre de usuario no existe, verifica igualmente contra una credencial ficticia fija, para que una cuenta inexistente y una contraseña incorrecta cuesten el mismo tiempo.

El mensaje en pantalla es una única frase genérica para todos los fallos: usuario desconocido, contraseña incorrecta, cuenta bloqueada o desactivada. El motivo real va en la fila de auditoría, donde un administrador puede ver treinta intentos de usuario desconocido desde una misma dirección en diez minutos. Ese patrón es el relleno de credenciales (credential stuffing): pares obtenidos de la filtración de otro sitio y probados en masa, que se detienen con el contador por dirección y no con el de cuenta, ya que cada cuenta quizá solo se pruebe una vez.

La fijación de sesión es la más sutil. El identificador que llevaba la página de inicio de sesión anónima no debe llevar la identidad verificada, así que la sesión previa termina tras la verificación y el navegador continúa en una nueva, sin copiar nada. «Recordarme» sigue la misma disciplina: nunca una contraseña guardada ni un valor adivinable, sino un token aleatorio largo por dispositivo, almacenado como hash, ligado a un empleado, con caducidad, revocable desde la consola de administración y comprobado de nuevo contra la cuenta activa antes de reconstruir el `SessionContext`.



Regla de seguridad

Nunca se confía en el navegador. Toda decisión importante —quién es el usuario, qué puede hacer, qué registro puede ver— se toma en el servidor, dentro de un servicio, y la interfaz solo la refleja. Si una regla solo existe como un botón oculto, no existe.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Implementar una página de inicio de sesión que verifique las credenciales contra un hash con sal y nunca almacene la contraseña.
- ✓ Crear un `SessionContext` que guarde una identidad verificada en `Application.Session` y sea la única fuente de verdad sobre quién es el usuario.

- ✓ Configurar la expiración de la sesión, el cierre por inactividad y el cierre de sesión explícito para que el estado se libere y no pueda reutilizarse.
- ✓ Proteger el flujo de inicio de sesión frente a la fuerza bruta, el relleno de credenciales y la fijación de sesión.

Guía del laboratorio / examen

Qué construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Implementa la autenticación de AccessOps: una página de inicio de sesión con bloqueo de cuenta tras cinco fallos, verificación de contraseñas con un hash moderno con sal, un SessionContext creado solo tras la verificación y guardado en la sesión, una expiración por inactividad que devuelve al usuario al inicio de sesión y borra el estado, un Logout explícito que termina la sesión y un registro de auditoría por cada intento de inicio de sesión.

Punto de partida

Abres la solución AccessOps que creaste en el Módulo 1, que sigue siendo un esqueleto: una ventana de inicio, una página de inicio de sesión provisional cuyo botón simplemente abre la consola principal y el documento del modelo de amenazas que señaló la autenticación como el primer control que había que construir. La tabla de empleados existe con nombres y departamentos, pero sin credenciales, y nada en la solución sabe todavía quién ha iniciado sesión.

Enfoque sugerido

- ✓ Escribe primero `PasswordService`, con un método Hash que genere una sal aleatoria nueva y un método Verify que vuelva a derivar con la sal y el número de iteraciones almacenados y compare con una comprobación de igualdad en tiempo constante; añade a la tabla de empleados las columnas de sal, número de iteraciones y hash, y crea dos cuentas de prueba a través del servicio.
- ✓ Define `SessionContext` con el id del empleado, el nombre visible, el id del departamento, el indicador de responsable, los roles y la hora de inicio de sesión, además de un accesor tipado sobre `Application.Session` que devuelva el contexto actual y lance una excepción cuando no haya nadie con la sesión iniciada.
- ✓ Construye `LoginService.SignIn` como la única vía que verifica una credencial: pregunta a `LoginThrottle` si la cuenta está bloqueada, verifica la contraseña, escribe el registro de auditoría para cada resultado y, si tiene éxito, reinicia los contadores, termina la sesión previa al inicio de sesión y crea el SessionContext para la nueva.
- ✓ Reconstruye la página de inicio de sesión para que el controlador de su botón solo llame a `LoginService` y reaccione al resultado —una frase genérica para cualquier fallo, la consola si tiene éxito— y haz que todas las demás pantallas exijan el contexto y redirijan al inicio de sesión cuando falte.
- ✓ Fija la expiración de la sesión en `Default.json`, añade el temporizador de inactividad con su aviso un minuto antes y escribe una única rutina Logout que borre el contexto, registre el cierre de sesión y llame a `Application.Exit()`, invocada tanto por el botón como por la vía de inactividad.

Entregables requeridos

- ✓ Página de inicio de sesión con bloqueo de cuenta y mensajes de error genéricos
- ✓ Servicio de hash y verificación de contraseñas
- ✓ SessionContext con identidad verificada
- ✓ Expiración por inactividad y cierre de sesión explícito
- ✓ Registros de auditoría de los inicios de sesión

Preguntas de repaso

- ✓ Tu página de inicio de sesión responde a una contraseña incorrecta en 480 ms y a un nombre de usuario desconocido en 6 ms. Explica qué averigua un atacante solo con esa diferencia y describe los dos cambios que eliminan la señal sin cambiar el mensaje.
- ✓ Un compañero propone guardar al empleado con la sesión iniciada en una propiedad estática porque «es más sencillo que ir a buscarlo a la sesión cada vez». Describe exactamente lo que ve el segundo usuario concurrente y en qué momento interfieren por primera vez las dos sesiones.
- ✓ Tu botón Logout borra el SessionContext pero no termina la sesión. Nombra dos cosas que siguen siendo recuperables después y di cuál consideras más grave en un portal que muestra documentos confidenciales.



Regla de seguridad

Nunca se confía en el navegador. Toda decisión importante —quién es el usuario, qué puede hacer, qué registro puede ver— se toma en el servidor, dentro de un servicio, y la interfaz solo la refleja. Si una regla solo existe como un botón oculto, no existe.