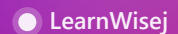


Autorización: roles, claims y permisos

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Comprobaciones basadas en roles y en claims, servicios de permisos, aplicación de las reglas en servicios y repositorios, y adaptación de la interfaz a lo que el usuario puede hacer. Qué significa esto para un desarrollador que protege una aplicación Wisej.NET en producción, y cómo leer este módulo.



Enfoque en producción

Comprobaciones basadas en roles y en claims, servicios de permisos, aplicación de las reglas en servicios y repositorios, y adaptación de la interfaz a lo que el usuario puede hacer.

1. El problema en producción

AccessOps ya sabe quién ha iniciado sesión: el Módulo 2 dejó una identidad verificada en el `SessionContext`. Saber quién es alguien no dice nada sobre lo que puede hacer con un documento confidencial o con una solicitud de aprobación de seis cifras. En un portal interno, el fallo rara vez es un inicio de sesión roto: es un administrativo de nóminas que puede leer los contratos del departamento jurídico porque la consulta nunca se acotó, o un jefe de equipo que aprueba su propia solicitud de gastos porque la única comprobación era un botón desactivado.

2. Conceptos clave

Concepto	Por qué importa
rol	Una función laboral amplia y con nombre, como Employee, Manager o Auditor. Los roles son cómodos de administrar, pero demasiado toscos para comprobarlos directamente, porque una pantalla necesita saber de una acción, no de todo un puesto.
claim	Un dato nombre/valor sobre la identidad verificada: departamento 14, centro de coste EU-North, responsable del equipo 7. Los claims llevan los datos que necesitan las reglas a nivel de registro, y proceden del almacén de identidades, nunca del navegador.
permiso	Una acción con nombre que la aplicación puede realizar, como <code>ApproveRequest</code> o <code>ViewDepartmentDocuments</code> . El código comprueba permisos; los administradores asignan roles; la correspondencia entre ambos es configuración.
resolución de permisos	El paso único, al iniciar sesión, que expande roles y claims en un conjunto de permisos plano e inmutable guardado en el <code>SessionContext</code> , para que ninguna solicitud vuelva a leer la tabla de roles y dos comprobaciones no puedan discrepar.
protección de autorización	Una llamada como <code>Require(Permission.ApproveRequest)</code> al principio de un método de servicio que lanza una excepción antes de que se haga ningún trabajo. Es el único lugar donde realmente se hace cumplir la regla.
regla a nivel de registro	Una restricción sobre qué filas puede tocar una acción permitida —tu propio departamento, tus propios subordinados— aplicada dentro de la consulta o del comando y no después de haber cargado ya los datos.
adaptación de la interfaz	Ocultar, desactivar o explicar una acción según el mismo conjunto de permisos que aplica el servidor. Es una cortesía hacia el usuario y nunca un control.

3. Desarrollo de la lección

Entran roles y claims, salen permisos

Empieza separando los tres vocabularios. Los administradores de AccessOps piensan en roles: un empleado es Employee, Manager, DepartmentAdmin o Auditor. El almacén de identidades lleva además claims: el departamento del empleado, el centro de coste, la lista de subordinados directos. El código de la aplicación, sin embargo, nunca debería preguntar `if (role == "Manager")`, porque el día en que se permita aprobar a los auditores financieros estarás editando cuarenta pantallas en lugar de una tabla.

Define en su lugar una enumeración `Permission` de acciones: `ViewDepartmentDocuments`, `ViewAllDocuments`, `UploadDocument`, `SubmitRequest`, `ApproveRequest`, `ViewAuditTrail`, `ManageUsers`. Un `PermissionService` se encarga de la correspondencia entre roles y permisos, aplica las concesiones y denegaciones individuales procedentes de los claims y produce un único conjunto plano. Las denegaciones prevalecen sobre las concesiones, y el resultado está ordenado y es inmutable, para que nada posterior pueda modificarlo.

Resuélvelo exactamente una vez, justo después de verificar las credenciales, y guárdalo en el `SessionContext` que ya vive en `Application.Session`. Ese momento importa: una sesión de Wisej.NET

es un objeto de servidor de larga duración, así que resolver en cada clic sobrecargaría la base de datos, mientras que resolver de forma perezosa invita a obtener dos respuestas distintas dentro de un mismo flujo de trabajo. El precio es la obsolescencia —un permiso revocado no tiene efecto hasta el siguiente inicio de sesión—, así que AccessOps guarda una marca de tiempo `PermissionsResolvedAt` y da a la consola de administración una acción «forzar nueva resolución» que reconstruye el conjunto para una sesión concreta.

La protección al principio de cada método

La aplicación de las reglas vive en un lugar predecible: la primera línea ejecutable de cada método público de servicio. `DocumentService` y `ApprovalService` reciben ambos una protección `IAuthorize` y el `SessionContext`; cada método empieza con `Require`, que lanza una `NotAuthorizedException` cuando falta el permiso. Como lanza una excepción en lugar de devolver un indicador, una comprobación del resultado olvidada no puede dejar pasar un comando en silencio.

Pon la protección en el servicio, no en el controlador del botón, y no solo en el repositorio. El controlador es uno de varios llamantes: un trabajo en segundo plano iniciado con `Application.StartTask`, un recordatorio programado, un handler HTTP personalizado y el callback de OpenID Connect del próximo módulo llegan todos a los mismos métodos, y una comprobación escrita en un controlador `Click` protege exactamente a uno de ellos. Los repositorios son el sitio equivocado por la razón opuesta: deben poder reutilizarse entre llamantes con derechos distintos, y un repositorio que consulta la sesión es imposible de probar.

Mantén también la protección honesta respecto a la identidad. Lee el usuario del `SessionContext`, nunca de un parámetro, para que un llamante no pueda pasar el ID de empleado de otra persona y heredar sus derechos. Cada rechazo se registra en el log con el usuario, el permiso y el registro de destino antes de que la excepción se propague, porque una ráfaga de rechazos es el aspecto que tiene un ataque visto desde el servidor.

Reglas que viven dentro de la consulta

Un permiso responde a «¿puede esta persona aprobar solicitudes en absoluto?»; no dice nada sobre *cuáles*. AccessOps necesita además dos reglas a nivel de registro: un empleado solo ve los documentos de su propio departamento, y un responsable solo puede aprobar las solicitudes creadas por sus subordinados directos. Son reglas que un feature flag no puede expresar, y las que con más frecuencia se añaden a posteriori, cuando los datos ya se han cargado.

Aplicálas dentro de la consulta. El repositorio de documentos toma el departamento del contexto y lo incorpora a la cláusula `Where`, para que las filas restringidas nunca salgan de la base de datos, nunca aparezcan en un recuento y nunca lleguen a una cuadrícula que pueda paginarlas. La alternativa —cargarlo todo y filtrar el `DataSource` después— ya ha pagado el coste de exponer los datos a un proceso que puede registrarlos en el log, y es justo lo que un cambio posterior en la paginación suele deshacer.

Los comandos se acotan del mismo modo. Aprobar obtiene la solicitud mediante un método del repositorio que exige tanto el ID de la solicitud como el ID de empleado del aprobador y devuelve null cuando la solicitud no está en el ámbito de ese responsable, de modo que un identificador fuera de ámbito no se distingue de uno inexistente.

```
public async Task ApproveAsync(int requestId, string note)
{
    _authorize.Require(Permission.ApproveRequest);

    var request = await _requests.GetForApproverAsync(
        requestId, _session.EmployeeId);
    if (request == null)
        throw new UnauthorizedException("Request is outside your scope.");

    request.Approve(_session.EmployeeId, note, DateTime.UtcNow);
    await _requests.SaveAsync(request);
}
```

Una interfaz que dice la verdad sin que se confíe en ella

Con el servidor sólido, la interfaz por fin puede ser útil. AccessOps lee el mismo conjunto de permisos al construir cada pantalla: la entrada de la consola de administración se quita de la navegación cuando falta `ManageUsers`, el botón de subida se oculta sin `UploadDocument`, y el botón Approve de una solicitud sobre la que el usuario no puede actuar se desactiva con una breve explicación —«solo el responsable del solicitante puede aprobar»— en lugar de no hacer nada en silencio.

- Oculta lo que el usuario nunca podrá hacer en esta aplicación.
- Desactiva y explica lo que podría hacer en otro registro o con otro rol.
- Nunca quites un control en lugar de escribir la comprobación del lado del servidor.

La diferencia entre ocultar y desactivar es una decisión de usabilidad, no de seguridad. En una aplicación Wisej.NET los controles son objetos C# del lado del servidor, así que un `Button` oculto realmente no se renderiza; pero los eventos que habría lanzado no son la única vía de entrada al servicio, y una sesión que ha seguido abierta durante un cambio de permisos puede estar mostrando un menú obsoleto. Por eso el último entregable del laboratorio es una prueba negativa que se salta la interfaz por completo: construye una sesión para alguien que no es responsable, llama directamente a `ApprovalService.ApproveAsync` con un ID de solicitud falsificado y comprueba tanto que se lanzó `UnauthorizedException` como que el estado de la solicitud en la base de datos no cambió.



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:

- ✓ Modelar roles, claims y permisos detallados y resolverlos una sola vez en el `SessionContext`.
- ✓ Aplicar la autorización dentro de servicios y repositorios para que cada comando y cada consulta comprueben al llamante.

- ✓ Aplicar reglas a nivel de registro, como la pertenencia al departamento y el ámbito del responsable, y no solo feature flags.
- ✓ Ocultar, desactivar o explicar acciones de la interfaz según los permisos sin depender de la interfaz para hacerlos cumplir.

Guía del laboratorio / examen

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



Laboratorio práctico

Añade autorización a AccessOps: un `PermissionService` que convierte roles y claims en permisos al iniciar sesión, una protección `Authorize` a la que llaman todos los métodos de `DocumentService` y `ApprovalService`, una regla a nivel de registro por la que los empleados solo ven su departamento y los responsables solo a sus subordinados, una interfaz que oculta las acciones de administración y desactiva los botones de aprobar que el usuario no puede usar, y una prueba que demuestra que el servicio rechaza una solicitud falsificada de alguien que no es responsable.

Punto de partida

Abres la solución AccessOps tal como la dejó el Módulo 2: una página de inicio de sesión que verifica un hash con sal, un `SessionContext` que guarda al empleado verificado en `Application.Session`, expiración por inactividad y cierre de sesión, y registros de auditoría de los inicios de sesión. La lista de documentos y la cola de aprobaciones existen como pantallas que ahora muestran todas las filas a todo el mundo, y la tabla de roles tiene empleados asignados a Employee, Manager, DepartmentAdmin y Auditor sin que nada la lea todavía.

Enfoque sugerido

- ✓ Define una enumeración `Permission` para las acciones que AccessOps realiza de verdad, y una correspondencia entre roles y permisos en la configuración y no en el código, para que un rol nuevo sea un cambio de datos.
- ✓ Escribe `PermissionService.ResolveFor(employee)` para que expanda los roles, aplique las concesiones y denegaciones individuales de los claims (la denegación prevalece) y devuelva un conjunto inmutable que el flujo de inicio de sesión asocia al `SessionContext` junto con el departamento y la lista de subordinados directos.
- ✓ Añade una protección `Authorize` que lea el `SessionContext` actual y lance `NotAuthorizedException`, y llama después a `Require` como primera línea de cada método público de `DocumentService` y `ApprovalService`.
- ✓ Lleva las reglas a nivel de registro a los repositorios: incorpora el filtro de departamento a la consulta de documentos y da al repositorio de aprobaciones una búsqueda que reciba el ID de la solicitud y el ID de empleado del aprobador y devuelva null cuando la solicitud quede fuera de los subordinados de ese responsable.
- ✓ Vincula las pantallas al conjunto de permisos —quita la consola de administración de la navegación, oculta la subida, desactiva Approve indicando el motivo en las filas sobre las que el usuario no puede actuar— y escribe después la prueba negativa que llama directamente al servicio con un ID de solicitud falsificado y comprueba que nada ha cambiado.

Entregables requeridos

- ✓ `PermissionService` que resuelve los permisos en el `SessionContext`

- ✓ Protección de autorización en cada método de servicio
- ✓ Reglas a nivel de registro por departamento y por responsable
- ✓ Adaptación de la interfaz según los permisos
- ✓ Prueba negativa con una solicitud falsificada

Preguntas de repaso

- ✓ A un auditor se le conceden derechos de aprobación mientras su sesión de AccessOps sigue abierta. Explica exactamente cuándo entra en vigor el nuevo permiso con tu diseño, por qué elegiste ese momento y qué tendrías que cambiar para que fuera inmediato.
- ✓ Tu consulta de documentos filtra por departamento en el repositorio. Describe qué gana un atacante si ese filtro se traslada al `DataSource` de la cuadrícula, aunque el resultado visible en pantalla sea idéntico.
- ✓ Tanto `ApprovalService` como su repositorio podrían comprobar que el aprobador es responsable del solicitante. Defiende uno de ellos como el lugar correcto y di qué se rompe si la comprobación está en el otro, o en ambos.



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.