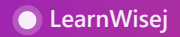


Autorisierung: Rollen, Claims und Berechtigungen

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Rollen- und claimbasierte Prüfungen, Berechtigungs-Services, Durchsetzung von Regeln in Services und Repositories und die Anpassung der UI an das, was der Benutzer tun darf. Was das für Entwickler bedeutet, die eine Wisej.NET-App im Produktivbetrieb absichern – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Rollen- und claimbasierte Prüfungen, Berechtigungs-Services, Durchsetzung von Regeln in Services und Repositories und die Anpassung der UI an das, was der Benutzer tun darf.

1. Das Problem im Produktivbetrieb

AccessOps weiß jetzt, wer angemeldet ist: Modul 2 hat eine verifizierte Identität im `SessionContext` hinterlassen. Zu wissen, wer jemand ist, beantwortet aber nicht, was diese Person mit einem vertraulichen Dokument oder einem sechsstelligen Genehmigungsantrag tun darf. Das typische Versagen in einem internen Portal ist selten ein kaputter Login – es ist der Sachbearbeiter der Lohnbuchhaltung, der die Verträge der Rechtsabteilung lesen kann, weil die Abfrage nie eingeschränkt wurde, oder die Teamleiterin, die ihre eigene Spesenabrechnung genehmigt, weil die einzige Prüfung ein deaktivierter Button war.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Rolle	Eine grobe, benannte Tätigkeitsfunktion wie Employee, Manager oder Auditor. Rollen sind bequem zu verwalten, aber zu grob, um direkt dagegen zu prüfen, denn ein Bildschirm muss über eine einzelne Aktion Bescheid wissen, nicht über eine ganze Stelle.
Claim	Eine Name-Wert-Aussage über die verifizierte Identität – Abteilung 14, Kostenstelle EU-North, Leitung von Team 7. Claims tragen die Daten, die Regeln auf Datensatzebene brauchen, und sie stammen aus dem Identitätsspeicher, nie aus dem Browser.
Berechtigung	Eine benannte Aktion, die die Anwendung ausführen kann, etwa <code>ApproveRequest</code> oder <code>ViewDepartmentDocuments</code> . Code prüft Berechtigungen, Administratoren vergeben Rollen, und die Zuordnung zwischen beiden ist Konfiguration.
Auflösung von Berechtigungen	Der einzelne Schritt beim Login, der Rollen und Claims zu einer flachen, unveränderlichen Berechtigungsmenge im <code>SessionContext</code> expandiert, sodass kein Request die Rollentabelle erneut liest und keine zwei Prüfungen zu unterschiedlichen Ergebnissen kommen.
Autorisierungs-Guard	Ein Aufruf wie <code>Require(Permission.ApproveRequest)</code> am Anfang einer Service-Methode, der eine Exception wirft, bevor irgendeine Arbeit geschieht. Er ist der einzige Ort, an dem tatsächlich durchgesetzt wird.
Regel auf Datensatzebene	Eine Einschränkung, welche Zeilen eine erlaubte Aktion berühren darf – die eigene Abteilung, die eigenen direkt unterstellten Mitarbeitenden –, angewendet innerhalb der Abfrage oder des Befehls statt erst, nachdem die Daten bereits geladen wurden.
UI-Anpassung	Eine Aktion ausblenden, deaktivieren oder erklären, basierend auf derselben Berechtigungsmenge, die der Server durchsetzt. Das ist eine Gefälligkeit gegenüber dem Benutzer und niemals eine Kontrolle.

3. Der rote Faden der Lektion

Rollen und Claims hinein, Berechtigungen heraus

Trennen Sie zunächst die drei Vokabulare. AccessOps-Administratoren denken in Rollen: Ein Mitarbeiter ist Employee, Manager, DepartmentAdmin oder Auditor. Der Identitätsspeicher trägt außerdem Claims – die Abteilung des Mitarbeiters, die Kostenstelle, die Liste der direkt Unterstellten. Anwendungscode sollte jedoch nie `if (role == "Manager")` fragen, denn an dem Tag, an dem Finanzprüfer genehmigen dürfen, bearbeiten Sie vierzig Bildschirme statt einer Tabelle.

Definieren Sie stattdessen eine `Permission`-Enumeration der Aktionen: `ViewDepartmentDocuments`, `ViewAllDocuments`, `UploadDocument`, `SubmitRequest`, `ApproveRequest`, `ViewAuditTrail`, `ManageUsers`. Ein `PermissionService` verwaltet die Zuordnung von Rollen zu Berechtigungen, wendet mitarbeiterbezogene Freigaben und Verbote aus Claims an und erzeugt eine einzige flache Menge. Verbote haben Vorrang vor Freigaben, und das Ergebnis ist sortiert und unveränderlich, damit nichts weiter hinten es verändern kann.

Lösen Sie sie genau einmal auf, unmittelbar nach der Prüfung der Zugangsdaten, und legen Sie sie im `SessionContext` ab, der bereits in `Application.Session` lebt. Der Zeitpunkt ist wichtig: Eine Wisej.NET-Session ist ein langlebiges Serverobjekt, daher würde eine Auflösung pro Klick die Datenbank überlasten, während eine verzögerte Auflösung zwei unterschiedliche Antworten innerhalb eines Arbeitsablaufs provoziert. Der Preis ist Veralterung – eine entzogene Berechtigung wirkt erst bei der nächsten Anmeldung –, daher speichert AccessOps einen Zeitstempel `PermissionsResolvedAt` und gibt der Admin-Konsole eine Aktion „Neu auflösen erzwingen“, die die Menge für eine benannte Session neu aufbaut.

Der Guard am Anfang jeder Methode

Die Durchsetzung lebt an einem vorhersehbaren Ort: der ersten ausführbaren Zeile jeder öffentlichen Service-Methode. `DocumentService` und `ApprovalService` erhalten beide einen `IAuthorize`-Guard und den `SessionContext`; jede Methode beginnt mit `Require`, das eine `NotAuthorizedException` wirft, wenn die Berechtigung fehlt. Weil der Guard eine Exception wirft, statt ein Flag zurückzugeben, kann eine vergessene Ergebnisprüfung keinen Befehl stillschweigend durchlassen.

Setzen Sie den Guard in den Service, nicht in den Button-Handler und nicht nur ins Repository. Der Handler ist nur einer von mehreren Aufrufern – ein mit `Application.StartTask` gestarteter Hintergrundjob, eine geplante Erinnerung, ein eigener HTTP-Handler und der OpenID-Connect-Callback aus dem nächsten Modul erreichen alle dieselben Methoden, und eine in einem `Click`-Handler geschriebene Prüfung schützt genau einen davon. Repositories sind aus dem umgekehrten Grund der falsche Ort: Sie sollen für Aufrufer mit unterschiedlichen Rechten wiederverwendbar sein, und ein Repository, das die Session befragt, ist nicht testbar.

Halten Sie den Guard auch bei der Identität ehrlich. Er liest den Benutzer aus dem `SessionContext`, nie aus einem Parameter, sodass ein Aufrufer nicht die Mitarbeiter-ID eines anderen übergeben und dessen Rechte erben kann. Jede Ablehnung wird mit Benutzer, Berechtigung und Zieldatensatz protokolliert, bevor die Exception weitergereicht wird, denn eine Häufung solcher Ablehnungen ist genau das, wie ein Angriff von der Serverseite aus aussieht.

Regeln, die in der Abfrage leben

Eine Berechtigung beantwortet die Frage „Darf diese Person überhaupt Anträge genehmigen?“; sie sagt nichts darüber, *welche*. AccessOps braucht zusätzlich zwei Regeln auf Datensatzebene: Ein Mitarbeiter sieht nur Dokumente seiner eigenen Abteilung, und ein Manager darf nur Anträge seiner direkt Unterstellten genehmigen. Das sind die Regeln, die ein Feature-Flag nicht ausdrücken kann, und genau die, die am häufigsten erst nachträglich angeflanscht werden, wenn die Daten schon geladen sind.

Wenden Sie sie innerhalb der Abfrage an. Das Dokument-Repository nimmt die Abteilung aus dem Kontext und baut sie in die `Where`-Klausel ein, sodass die gesperrten Zeilen die Datenbank nie verlassen, nie in einer Zählung auftauchen und nie ein Grid erreichen, das durch sie blättern könnte. Die Alternative – alles laden und danach die `DataSource` filtern – hat den Preis bereits bezahlt, die Daten einem Prozess preiszugeben, der sie protokollieren kann, und genau das macht eine spätere Änderung am Paging gern wieder rückgängig.

Befehle werden genauso eingeschränkt. Beim Genehmigen wird der Antrag über eine Repository-Methode geholt, die sowohl die Antrags-ID als auch die Mitarbeiter-ID des Genehmigenden verlangt und null

zurückgibt, wenn der Antrag nicht im Zuständigkeitsbereich dieses Managers liegt; so ist eine ID außerhalb des Bereichs nicht von einer fehlenden zu unterscheiden.

```
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);
}
```

Eine UI, die die Wahrheit sagt, ohne dass man ihr vertraut

Wenn der Server solide ist, kann die Oberfläche endlich hilfreich sein. AccessOps liest beim Aufbau jedes Bildschirms dieselbe Berechtigungsmenge: Der Eintrag der Admin-Konsole wird aus der Navigation entfernt, wenn `ManageUsers` fehlt, der Upload-Button wird ohne `UploadDocument` ausgeblendet, und der Approve-Button bei einem Antrag, den der Benutzer nicht bearbeiten darf, wird mit einer kurzen Erklärung deaktiviert – „nur der Manager des Antragstellers kann genehmigen“ –, statt stillschweigend nichts zu tun.

- Blenden Sie aus, was der Benutzer in dieser Anwendung überhaupt nie tun kann.
- Deaktivieren und erklären Sie, was er bei einem anderen Datensatz oder mit einer anderen Rolle tun könnte.
- Entfernen Sie nie ein Control, statt die serverseitige Prüfung zu schreiben.

Der Unterschied zwischen Ausblenden und Deaktivieren ist eine Frage der Bedienbarkeit, keine der Sicherheit. In einer Wisej.NET-App sind die Controls serverseitige C#-Objekte, ein ausgeblendeter `Button` wird also tatsächlich nicht gerendert – aber die Events, die er ausgelöst hätte, sind nicht der einzige Weg in den Service, und eine Session, die über eine Berechtigungsänderung hinweg offen war, zeigt womöglich ein veraltetes Menü. Deshalb ist das letzte Ergebnis des Labs ein Negativtest, der die UI komplett umgeht: Er baut eine Session für jemanden ohne Managerrolle, ruft `ApprovalService.ApproveAsync` direkt mit einer gefälschten Antrags-ID auf und prüft sowohl, dass `UnauthorizedException` geworfen wurde, als auch, dass sich der Status des Antrags in der Datenbank nie geändert hat.



Sicherheitsgewohnheit

Dem Browser wird nie vertraut. Jede relevante Entscheidung – wer der Benutzer ist, was er tun darf, welchen Datensatz er sehen darf – wird auf dem Server in einem Service getroffen, und die UI spiegelt sie nur wider. Existiert eine Regel nur als versteckter Button, dann existiert sie nicht.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Rollen, Claims und feingranulare Berechtigungen modellieren und sie einmalig in den SessionContext auflösen.
- ✓ Autorisierung in Services und Repositories durchsetzen, sodass jeder Befehl und jede Abfrage den Aufrufer prüft.
- ✓ Regeln auf Datensatzebene wie Abteilungszugehörigkeit und Managerzuständigkeit anwenden, nicht nur Feature-Flags.
- ✓ UI-Aktionen anhand von Berechtigungen ausblenden, deaktivieren oder erklären, ohne sich bei der Durchsetzung auf die UI zu verlassen.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab erstellen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praxis-Lab

Ergänzen Sie AccessOps um Autorisierung: einen `PermissionService`, der beim Login Rollen und Claims in Berechtigungen auflöst, einen `Authorize-Guard`, den jede Methode von `DocumentService` und `ApprovalService` aufruft, eine Regel auf Datensatzebene, nach der Mitarbeitende nur ihre Abteilung und Manager nur ihre direkt Unterstellten sehen, eine UI, die Admin-Aktionen ausblendet und `Approve-Buttons` deaktiviert, die der Benutzer nicht verwenden darf, und einen Test, der belegt, dass der Service einen gefälschten Antrag von jemandem ohne Managerrolle abweist.

Ausgangspunkt

Sie öffnen die AccessOps-Solution so, wie Modul 2 sie hinterlassen hat: eine Login-Seite, die gegen einen gesalzenen Hash prüft, ein `SessionContext`, der den verifizierten Mitarbeiter in `Application.Session` hält, Leerlauf-Timeout und Logout sowie Audit-Einträge für Anmeldungen. Die Dokumentenliste und die Genehmigungswarteschlange existieren als Bildschirme, die derzeit jedem alle Zeilen zeigen, und in der Rollentabelle sind Mitarbeitende den Rollen `Employee`, `Manager`, `DepartmentAdmin` und `Auditor` zugeordnet, ohne dass sie bisher jemand ausliest.

Vorgeschlagenes Vorgehen

- ✓ Definieren Sie eine `Permission`-Enumeration für die Aktionen, die AccessOps tatsächlich ausführt, und eine Zuordnung von Rollen zu Berechtigungen in der Konfiguration statt im Code, sodass eine neue Rolle nur eine Datenänderung ist.
- ✓ Schreiben Sie `PermissionService.ResolveFor(employee)`, das Rollen expandiert, mitarbeiterbezogene Freigaben und Verbote aus Claims anwendet (Verbote haben Vorrang) und eine unveränderliche Menge zurückgibt, die der Login-Ablauf zusammen mit der Abteilung und der Liste der direkt Unterstellten an den `SessionContext` anhängt.
- ✓ Fügen Sie einen `Authorize`-Guard hinzu, der den aktuellen `SessionContext` liest und `NotAuthorizedException` wirft, und rufen Sie `Require` als erste Zeile jeder öffentlichen Methode von `DocumentService` und `ApprovalService` auf.
- ✓ Verlagern Sie die Regeln auf Datensatzebene in die Repositories: Bauen Sie den Abteilungsfilter in die Dokumentabfrage ein und geben Sie dem Genehmigungs-Repository eine Abfragemethode, die die Antrags-ID und die Mitarbeiter-ID des Genehmigenden entgegennimmt und null liefert, wenn der Antrag nicht von einem direkt Unterstellten dieses Managers stammt.
- ✓ Binden Sie die Bildschirme an die Berechtigungsmenge – die Admin-Konsole aus der Navigation entfernen, Upload ausblenden, Approve bei Zeilen, die der Benutzer nicht bearbeiten darf, mit Begründung deaktivieren – und schreiben Sie dann den Negativtest, der den Service direkt mit einer gefälschten Antrags-ID aufruft und prüft, dass sich nichts geändert hat.

Geforderte Ergebnisse

- ✓ PermissionService, der Berechtigungen in den SessionContext auflöst
- ✓ Autorisierungs-Guard in jeder Service-Methode
- ✓ Regeln auf Datensatzebene für Abteilung und Manager
- ✓ Berechtigungsabhängige Anpassung der UI
- ✓ Negativtest mit einem gefälschten Antrag

Wiederholungsfragen

- ✓ Eine Prüferin erhält Genehmigungsrechte, während ihre AccessOps-Session noch offen ist. Erklären Sie genau, wann die neue Berechtigung in Ihrem Design wirksam wird, warum Sie diesen Zeitpunkt gewählt haben und was Sie ändern müssten, damit sie sofort gilt.
- ✓ Ihre Dokumentabfrage filtert im Repository nach Abteilung. Beschreiben Sie, was ein Angreifer gewinnt, wenn dieser Filter stattdessen in die `DataSource` des Grids verlagert wird, obwohl das sichtbare Ergebnis auf dem Bildschirm identisch ist.
- ✓ Sowohl `ApprovalService` als auch sein Repository könnten prüfen, ob der Genehmigende der Vorgesetzte des Antragstellers ist. Begründen Sie, welcher der beiden der richtige Ort ist, und sagen Sie, was kaputtgeht, wenn die Prüfung im anderen oder in beiden liegt.



Sicherheitsgewohnheit

Dem Browser wird nie vertraut. Jede relevante Entscheidung – wer der Benutzer ist, was er tun darf, welchen Datensatz er sehen darf – wird auf dem Server in einem Service getroffen, und die UI spiegelt sie nur wider. Existiert eine Regel nur als versteckter Button, dann existiert sie nicht.