CHECKLIST

ASP.NET Core Authorization Policies Checklist

A practical checklist for designing, validating, and preparing ASP.NET Core authorization policies that combine role and claim checks.

ASP.NET Core Authorization Policies: Role and Claim Checks Checklist

Use this checklist to design, validate, and prepare ASP.NET Core authorization policies for production. It is intended for teams that need reusable authorization rules, consistent access control, and safer deployment practices.

---

1) Define the access rule clearly

- [ ] Identify the protected action, endpoint, or resource.
- [ ] State the business purpose of the authorization decision in one sentence.
- [ ] Define the minimum caller attributes required.
- [ ] Separate broad authority from specific entitlement.
- [ ] Confirm whether the rule should apply to all callers or only a subset.
- [ ] Decide whether the policy should enforce authentication explicitly.

2) Decide what belongs in a role and what belongs in a claim

- [ ] Use a role for stable business functions such as administrator, operator, auditor, or support engineer.
- [ ] Use a claim for attributes, entitlements, scopes, tenant membership, assurance level, or other fine-grained conditions.
- [ ] Avoid using roles for conditions that change frequently.
- [ ] Avoid using claims as a substitute for weak or missing authentication.
- [ ] Confirm whether the endpoint needs both a role and a claim.
- [ ] Confirm whether the endpoint can be authorized with either one instead of both.

3) Design the policy name and intent

- [ ] Give the policy a descriptive name that reflects the access boundary.
- [ ] Avoid vague policy names such as CanAccessData.
- [ ] Make the policy name understandable in code review and incident response.
- [ ] Document the intended role and claim requirements next to the policy definition.
- [ ] Keep the policy simple enough to explain to a reviewer in one minute.

4) Verify claim sources and mapping

- [ ] Identify the identity provider or token issuer that supplies the claims.
- [ ] Verify the exact claim type names received by the application.
- [ ] Confirm whether role claims are mapped consistently across environments.
- [ ] Check whether JWT claim mapping changes the claim names seen by ASP.NET Core.
- [ ] Verify that token validation is working before authorization is evaluated.
- [ ] Confirm that trusted claims are signed and come from a reliable source.

5) Attach the policy consistently

- [ ] Apply the policy to every endpoint that requires the same access rule.
- [ ] Use the same policy for controllers, minimal APIs, and resource handlers when appropriate.
- [ ] Avoid duplicating role checks and claim checks in multiple places.
- [ ] Ensure the endpoint does not bypass the policy through an alternate route.
- [ ] Review fallback behavior for endpoints that are not explicitly decorated.

6) Test positive and negative cases

- [ ] Test a valid authenticated caller who meets all requirements.
- [ ] Test a caller with the right role but missing the required claim.
- [ ] Test a caller with the required claim but missing the role.
- [ ] Test an unauthenticated caller.
- [ ] Test a caller with the wrong claim value.
- [ ] Test a caller whose role claim type differs from the expected mapping.
- [ ] Confirm the failure result is a denial, not a silent success.

7) Validate production-like behavior

- [ ] Test the policy with a token that matches the real issuer and environment.
- [ ] Inspect the ClaimsPrincipal seen by the application.
- [ ] Verify the application receives the expected claim names and values.
- [ ] Confirm behavior when a claim is missing, renamed, or empty.
- [ ] Verify behavior when a user is in the correct role but the claim is absent.
- [ ] Verify behavior when the claim is present but the token is otherwise invalid.

8) Check fallback and edge cases

- [ ] Confirm default access for unauthenticated endpoints is intentional.
- [ ] Verify whether fallback policies apply to endpoints without explicit authorization metadata.
- [ ] Check what happens when authentication succeeds but authorization fails.
- [ ] Confirm how multiple requirements are combined: all required versus any one sufficient.
- [ ] Review whether resource-based checks need a custom handler instead of stacked attributes.

9) Confirm auditability and maintainability

- [ ] Make the policy easy to trace from endpoint to requirement.
- [ ] Ensure the access decision can be explained during a security review.
- [ ] Prefer a single named policy over repeated inline checks.
- [ ] Split overly complex policies into smaller, auditable rules.
- [ ] Add comments or documentation for any non-obvious requirement.
- [ ] Review whether the policy still matches the business control after changes to roles or claims.

10) Prepare for production release

- [ ] Confirm the policy matches the intended security model.
- [ ] Verify all claim and role names in the current token format.
- [ ] Check that staging and production use the same or compatible claim mapping.
- [ ] Confirm that logging is sufficient to diagnose denials without exposing sensitive data.
- [ ] Review the impact of identity provider changes on authorization behavior.
- [ ] Retest after any update to authentication, token format, or role assignment logic.

Quick production readiness check

Before release, answer these questions:

- [ ] Does the policy express a real business or security rule?
- [ ] Are the role and claim requirements explicit and necessary?
- [ ] Have we tested both allow and deny outcomes?
- [ ] Do we know the exact claim names the application receives?
- [ ] Have we verified fallback and edge-case behavior?
- [ ] Would a reviewer understand why this policy exists?

Example policy review prompt

Use this prompt during code review or change approval:

> What exact caller attributes does this policy require, where do those attributes come from, and how do we know the application receives them in the expected form?

If the team cannot answer that clearly, the policy is not ready for production.

Final rule of thumb

- Use roles for broad authority.
- Use claims for specific conditions.
- Use policies to combine them into one reusable authorization decision.
- Validate the real token and principal data before production.