# Testing Security: Authorization & Policies

**Benjamin Day**
TRAINER | COACH | DEVELOPER

@benday  www.benday.com

# Overview

**Security Overview**

- Authentication vs. Authorization
- Role-based security
- Claims-based security

**Security in ASP.NET Core**

- [Authorize]
- Role-based
- Policy-based

**Lots of demos**

# First up:
# Security Overview

# Security Overview

# Two Big Pieces

Authentication

Authorization

# Two Big Pieces

**Authentication**

- Who are you?

**Authorization**

- What can you do?
- Permissions

# Authentication

**Username & passwords in your application**

**Social logins**

- Google, Facebook, Twitter, Microsoft Accounts (MSA)

**Azure Active Directory (AAD)**

**Windows Active Directory**

**Lots of other options**

**Bad news:**
**Authentication is more complex**

**Good news:**
**It's usually external to your app**

# Authorization

**Core part of your application logic**

**What is the user allowed to do?**

**User permissions**

- Where are permissions stored?
- How do you check user permissions in your application logic?

# Permissions in ASP.NET

**Role-based Security**

**Claims-based Security**

# Role-based Security

**Sample roles:**

- Administrators
- Users
- Power Users
- Sales
- Marketing

**User is a member of a role**

**Application allows roles to do things in the application**

# Beware:
# Role-based security has limitations

# Role-based Security Concerns

**Maintenance concerns**

- (Not application security concerns)

**Fine for simple apps with simple security**

**“Is User X a member of Role Y?”**

- Broad permissions

**“Is User X a member of Role Y for Item Z?”**

- Permissions in the context of an item
- Impossible with role-based security

# Claims-based Security

**Goes beyond role-based security**

**Authorization based on a list of Claims**

**What does the user claim to be?**

**What does the user claim to be able to do?**

**Claims aren't just permissions**

**Claims can be things like**

- Age
- Name
- Email address
- Roles

**Claims have context**

# Claims Have Context

**Role-based security is just a role**

- Is the user a member of a role

**Claims are key/value pairs**

- Claim Type
- Claim Value

**“Is User X a member of Role Y for Item Z?”**

**Role-based security can’t do this**

**Claims-based security can**

# ASP.NET Core security is primarily about Claims

# Next up:
# Pieces of security in ASP.NET Core

# How is security implemented in ASP.NET Core?

# Security in ASP.NET Core

IIdentity

IPrincipal

ClaimsIdentity

ClaimsPrincipal

# Things to Think About

**Single Responsibility Principle**

**Code against interfaces**

**Keep logic isolated**

**Dependency Injection**

**Code against**

- IIdentity / IPrincipal
- ClaimsIdentity / ClaimsPrincipal

Assumption

You're focused on testing Authorization

# Two Types of Code Related to Authorization

**Code that checks if a user is authorized**

- Security decisions

**Code that you're trying to authorize**

- Actions you're trying to protect

**Keep these separated!!!**

- Single Responsibility Principle

It’s always going to be easier
to unit test the code that makes
the security decisions

If you're trying to test
the decision code and
the protected code
at the same time...

...it's probably an integration test
and not a unit test

# Two Ways to Implement Authorization in ASP.NET Core

**Using the [Authorize] attribute**

- Part of ASP.NET Core
- Apply to Controllers or Controller methods

**Custom logic**

- Checks against IPrincipal yourself

# The [Authorize] Attribute

**Apply it to a**

- Controller class
- Controller method

**User must be authenticated**

- [Authorize()]

**Role-based authorization**

- [Authorize(Roles = "Administrator")]

**Policy-based authorization**

- [Authorize(Policy = “AdminOnlyPolicy")]

# The Bad News About the [Authorize] Attribute

**Nearly impossible to unit test**

**It's really hard to integration test**

**My recommendation:**

- Don't try to test that it's working
- Use reflection to check that the attribute is there with the right value(s)

# Checks Against IPrincipal

**Get an instance of IPrincipal**

**Write checks against IPrincipal**

**Succeed or fail based on the checks**

**Recommendation:**

- Group the checks into methods that make the authorization decisions
- Unit test the logic that makes the decisions

# ASP.NET Core Security Policies

**Encapsulates the authorization decision logic**

**[Authorize(Policy = “AdminOnlyPolicy")]**

**Define policies in Startup.cs**

**Policy has two parts:**

- Requirement
- Handler

**IAuthorizationRequirement**

**AuthorizationHandler<T>**

# IAuthorization Requirement

IAuthorizationRequirement

Configuration information related to a Policy

Create a class

Implement the interface

(Optional) Provide properties for config values

# Authorization Handler<T>

**AuthorizationHandler<T>**

- T = Class the implements IAuthorizationRequirement

**Implement HandleRequirementAsync( context, requirement)**

**AuthorizationHandlerContext**

- Current authorization check info
- Identity, Principal
- MVC Context

**Make the decision**

- Succeed()
- Fail()

# Testing the Policies

Unit test the policy handler logic in isolation

Focus testing on AuthorizationHandler<T>

Integration testing policy handlers with Controllers is HARD

# Next up:
# Demos

# Demo

Unit testing the [Authorize()] attribute

"Didn't you say that was impossible?"

Testing the *existence* of [Authorize()] using Reflection

Avoids integration tests

Trusts that the decision implementation works

Technique is by David Pine

# Credit for This Idea

David Pine

Microsoft MVP

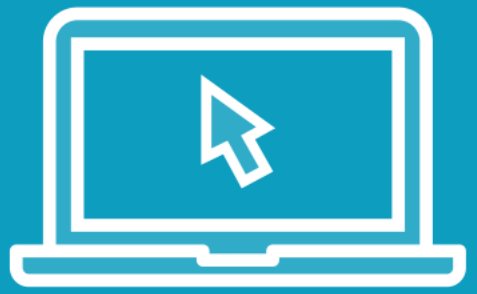
Google Developer Expert

Twitter: @davidpine7

https://davidpine.net/blog/asp-net-core-security-unit-testing/

# Next up:
# Implementing
# AuthorizationHandler<T>

# Demo



**Multi-part demo**

**ASP.NET Core AuthorizationHandler and Policy-based Authorization**

**Part 1: The overall code structure**

**Part 2: Implement the unit tests**

**Part 3: Implement AuthorizationHandler<T>**

**Part 4: Create the policy**

- Hook it in to ASP.NET Core

Next up:
Use the Strategy Pattern
to make authorization decisions

# Summary

**Security Overview**

- Authentication vs. Authorization
- Role-based security
- Claims-based security

**Security in ASP.NET Core**

- [Authorize]
- Role-based
- Policy-based

# Next up:
# Testing Custom Security Logic and Middleware