SpbDotNet N°45

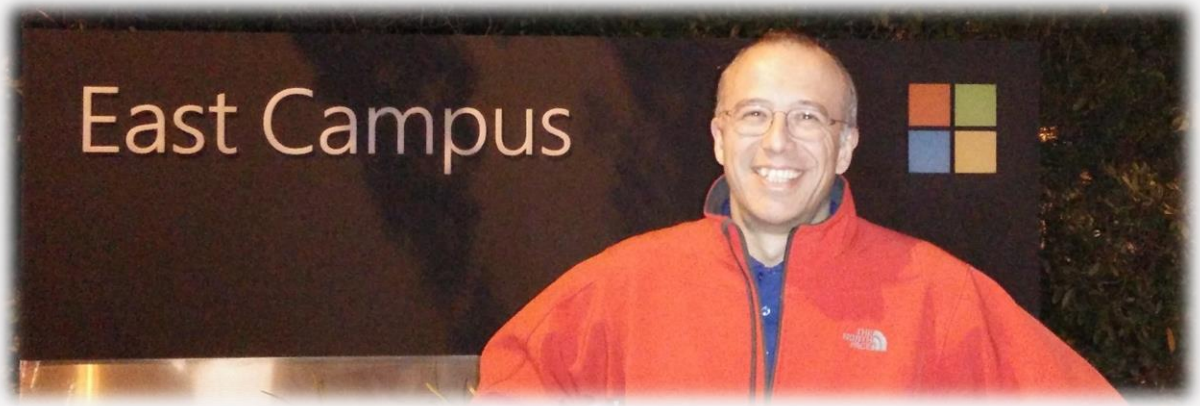
Deep dive into the ASP.NET Core authorization framework

Raffaele Rialdi
Senior Software Architect
Microsoft MVP

@raffaeler
https://github.com/raffaeler
http://iamraf.net

Who am I?

- Raffaele Rialdi, Senior Software Architect in Vevy Europe – Italy
  - @raffaeler also known as "Raf"
- Consultant in many industries
  - Manufacturing, racing, healthcare, financial, …
- Speaker and Trainer around the globe (development and security)
  - Italy, Romania, Bulgaria, Russia (Moscow, St Petersburg and Novosibirsk), USA, …
- And proud member of the great Microsoft MVP family since 2003



Why ASP.NET Core?

1. Generic Host for all non-visual applications
2. Outstanding performance on the whole request/response
3. Entirely re-written from scratch to avoid any past glitch
4. Cross-platform and container-ready
5. *Built-in support for policy-based authorization*
6. Extremely easy to extend (dependency injection)
7. Built-in support for background services
8. A single project addresses self-hosting, service or reverse-proxy scenarios
9. Simpler configuration system
10. Rich middleware ecosystem (Websockets, SignalR, Swashbuckle, …)

Performance!

Saturating 10GbE with 7+ million requests

https://www.techempower.com/benchmarks/#section=test&runid=8ca46892-e46c-4088-9443-05722ad6f7fb&hw=ph&test=plaintext

Best plaintext responses per second, Test environment (331 tests)

Rnk	Framework	Best performance (higher is better)		Errors	Cls (Classification)	Lng (Language)	Plt (Platform)	FE (Web Server)
1	actix-raw	7,003,320	100.0%	0	Plt	Rus	Non	act
2	wizzardo-http	7,002,116	100.0%	0	Mcr	Jav	Non	Non
3	aspcore	7,000,118	100.0%	0	Plt	C#	.NE	kes
4	ulib	6,998,365	99.9%	1	Plt	C++	Non	ULi
5	ulib-plaintext_fit	6,998,068	99.9%	0	Plt	C++	Non	ULi
6	hyper	6,996,874	99.9%	0	Mcr	Rus	Rus	Hyp
7	libreactor	6,996,726	99.9%	0	Mcr	C	Non	Non
8	tokio-minihttp	6,995,981	99.9%	0	Mcr	Rus	Rus	tok
9	baseio-http-lite	6,983,540	99.7%	0	Plt	Jav	bas	Non
10	aspcore-rhtx	6,975,733	99.6%	0	Plt	C#	.NE	kes
11	rapidoid-http-fast	6,951,751	99.3%	0	Plt	Jav	Rap	Non
12	rapidoid	6,926,431	98.9%	0	Plt	Jav	Rap	Non
13	actix	6,840,894	97.7%	0	Mcr	Rus	Non	act
14	mofuw	6,762,181	96.6%	0	Plt	Nim	Non	Non
15	thruster	6,498,029	92.8%	0	Mcr	Rus	Rus	Non
16	httpbeast	6,464,835	92.3%	0	Plt	Nim	Non	Non

ASPNET Core: Security and GDPR

- HTTPS everywhere
  - Self-signed certificates generation
  - Tip: use "Let's Encrypt" free provider for production https://letsencrypt.org/
  - app.UseHttpsRedirection() ➔ server-side redirection to HTTPS
  - app.UseHsts() ➔ client-side redirection to HTTPS

- New templates provide basic GDPR requirements
  - "cookie banner"
  - Privacy data download
  - Profile deletion

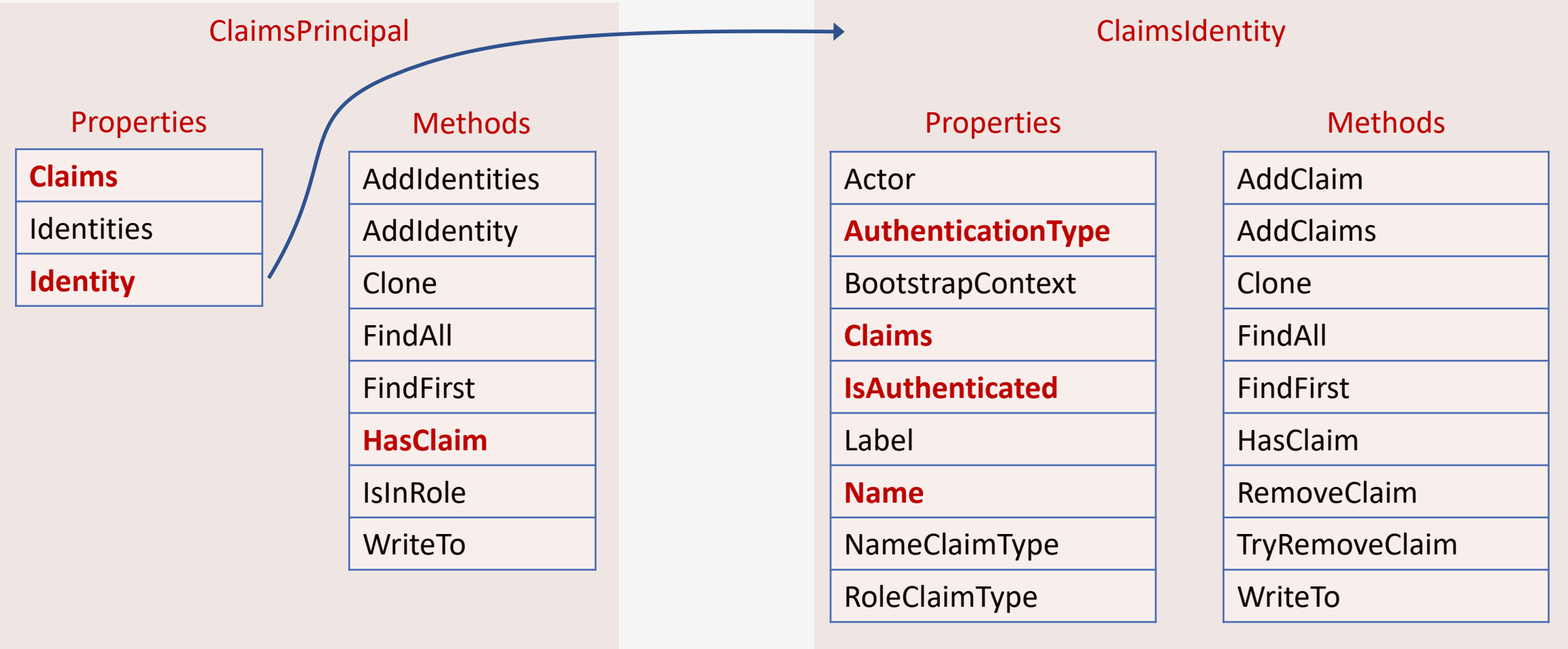
Authentication is simpler

- Authentication is plugged in using Dependency Injection
  - Many provider available in the box
  - New providers can be easily created

- New in ASP.NET Core 3.0 SDK Preview 3: Identity Server is integrated
  - Provide authentication for SPA using Web API
  - More sophisticated authentication scenarios
  - New ASP.NET Core templates for SPAs and classic web apps

Let's focus on the Authorization

Houston, we got a token … (and now what?)

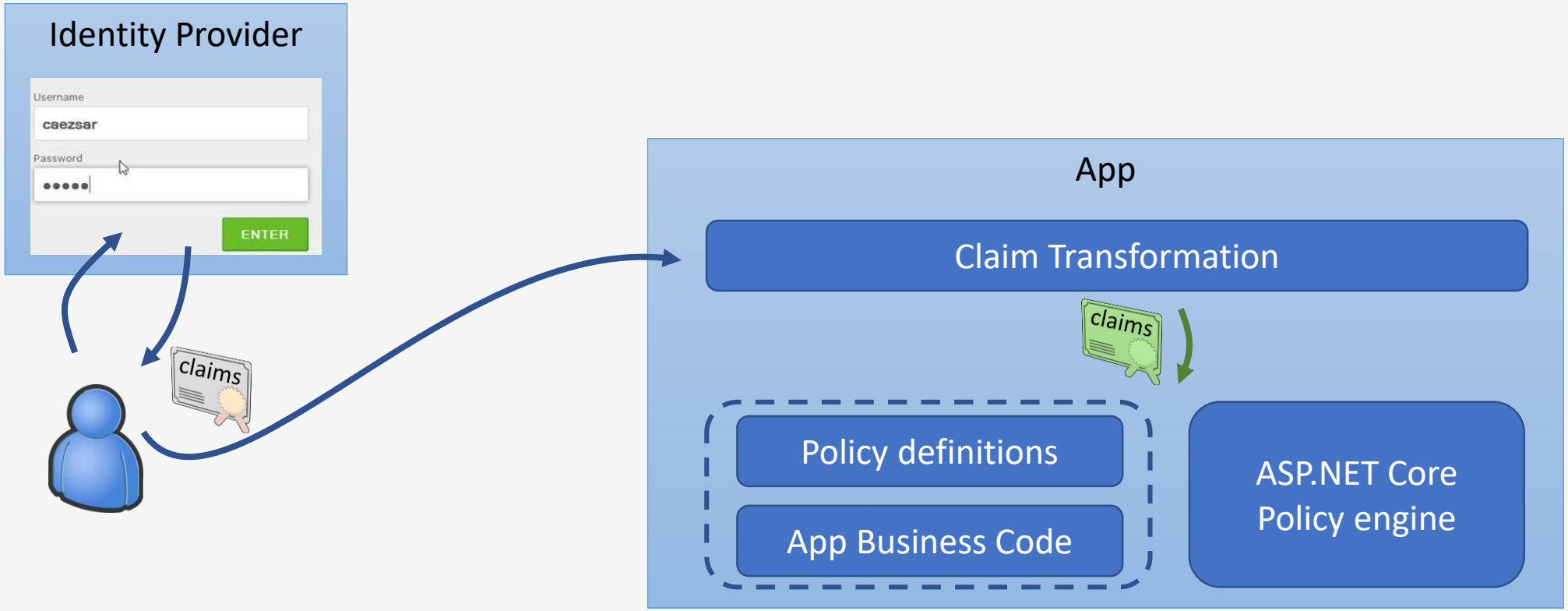


User vs Application claims

- Identity Providers provide only general user-related Claims
  - FirstName, LastName, age and company-related data
  - Two-Factor authentication (2FA) or Fido tokens (U2F)
  - <u>Should NOT contain permissions</u>

- Application Claims are provided by the App or an external service
  - Generated claims are application specific
  - <u>May contain permissions</u>
  - As they are computed every time the user lands in the App, they are 'volatile'

Transforming claims



Policies, Requirements and Handlers

- A Policy is just a container for one or more requirements
  - Access is granted if … **all** the requirements are satisfied (**AND** condition)

- A Requirement is associated to one or more Authorization Handlers
  - Access is granted if … **at least one** handler is satisfied (**OR** condition)
  - Requirements are either empty or contain few properties used by the Handlers
  - OperationAuthorizationRequirement just defines a Name property

```
public class AgeRequirement : IAuthorizationRequirement  {  }
```

Authorization Handlers: the core logic

- Contain the authorization logic
- Derives from AuthorizationHandler<TRequirement>
- Gets as input: User, Claims and Requirement to satisfy

```
public class SpeakerAuthorizationHandler : AuthorizationHandler<SpeakerRequirement>
{
    protected override Task HandleRequirementAsync(
        AuthorizationHandlerContext context, SpeakerRequirement requirement)
    {
        var identity = context.User.Identity as ClaimsIdentity;
        if (context.User.HasClaim(identity.RoleClaimType, "Speaker"))
            context.Succeed(requirement);
        return Task.CompletedTask;
    }
}
```

Resource Authorization Handlers

- Derive from AuthorizationHandler<TRequirement, TResource>
- Handlers get User, Claims, Requirement and the resource to access
- Passing a null resource to the Authorization Manager will reject

```
public class OwnerHandler : AuthorizationHandler<LcrudRequirement, Article>
{
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext context,
      LcrudRequirement requirement, Article resource)
    {
        if (resource.Owner == context.User.Identity.Name) context.Succeed(requirement);
        return Task.CompletedTask;
    }
}
```

Authorize or reject the access?

- Do you want to authorize the request?
  - At least one of the handlers for every requirement must call "Succeed"
- Do you want to reject the request?
  - Just exit the method
- Is something really bad happened? (exception or db failure)
  - Call **context.Fail**, but be sure to call it **only** in case of fatal errors

Triggering
the Authorization checks

Declarative authorization

- It may be used when the authorization process does not rely on the resource being accessed

```
[Authorize("ReadArticles")]
```

- The attribute can contain other parameters:

```
[Authorize(
    AuthenticationSchemes = "...",
    Policy = "...",
    Roles = "...")]
```

Imperative authorization

- IAuthorizationService can be injected in any Page/Controller

```
var authResult = await _auth.AuthorizeAsync(
                 User, Article, ArticlePolicies.ReadArticles);
if (!authResult.Succeeded)  return new ChallengeResult();
```

Access denied

- And in the Razor views as well

```
@using Microsoft.AspNetCore.Authorization
@inject IAuthorizationService AuthorizationService
@{
  var result = await AuthorizationService.AuthorizeAsync(
               User, article, Operations.Create);
  if (result.Succeeded)   {
      <a asp-action="Create">Edit</a>
  }
}
```

Policy Providers

- What can we do to calculate the policies dynamically?

  Use cases:

  1. Loading the policies from a database
  2. Dynamically generate the code for the authorization logic

- Policy Providers just implements IAuthorizationPolicyProvider
  - GetPolicyAsync method returns the generated policy
  - GetDefaultPolicyAsync returns the default policy

Filtering lists based on authorization

- Strategy 1
  - Load unfiltered query results from the database
  - Filter the data in-memory
  - Paginate the results in-memory (hard part)
  - Result: bad performance! A lot of read data will be discarded

- Strategy 2
  - Modify all the queries to load only the authorized data
  - The authorization logic is now in both the Handlers and the DB queries
  - Result: hard to maintain! Very error-prone

Filtering lists based on authorization

- Strategy 3
  - Leverage the Entity Framework Core **Global Filters**
  - Inject the authorization filters on all the queries touching the affected tables
  - Result: poor performance in complex queries

- Strategy 4
  - Use an Expression Visitor to generate the authorization code
  - The logic written in C# but translated in SQL
  - Result: hard to write the first time, easy to maintain

Demo!

Takeaways

- Authorization should be designed as an Application-specific process
- Policies should be modeled appropriately and subject to unit-testing
- The test outcome should be part of the GDPR report

Questions?

```
unsafe
{
    byte* blob = stackalloc byte[_size];
    Span<byte> span = new Span<byte>(blob, _size);
    span.Fill(1);
    return span;

}
```

Thank you!