NT KONF

25. – 27.
SEPTEMBER
2023
PORTOROŽ

NT KONF
NT KONFERENCA

NT KONF
NT KONFERENCA

25. – 27.
SEPTEMBER
2023
PORTOROŽ

A practical guide to authorization in ASP.NET Core

M. Eng. Raffaele Rialdi

@raffaeler - raffaeler@vevy.com

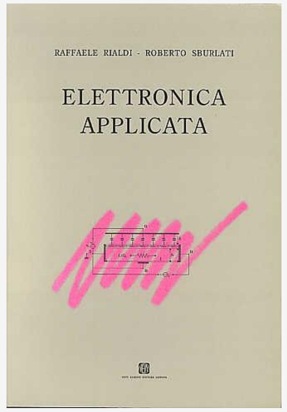
MVP Microsoft Most Valuable Professional

The code for this talk is here:
https://github.com/raffaeler/authorization

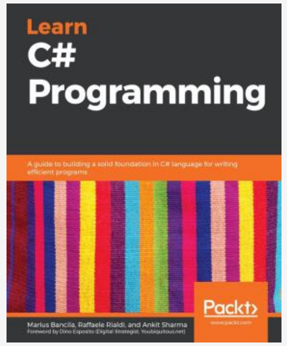
MVP Microsoft Most Valuable Professional

Who am I?

- Raffaele Rialdi: @raffaeler also known as "Raf"
  - Master degree in Electronic Engineering, University of Genoa (Italy)
  - Teacher at the Informatics Engineering University of Genoa
- Consultant in many industries
  - Manufacturing, racing, healthcare, financial, …
- Speaker and Trainer around the globe (development and security)
  - Italy, Romania, Slovenia, Bulgaria, Russia, USA, …
- Proud member of the great Microsoft MVP family since 2003


RAFFAELE RIALDI · ROBERTO SBURLATI

ELETTRONICA APPLICATA

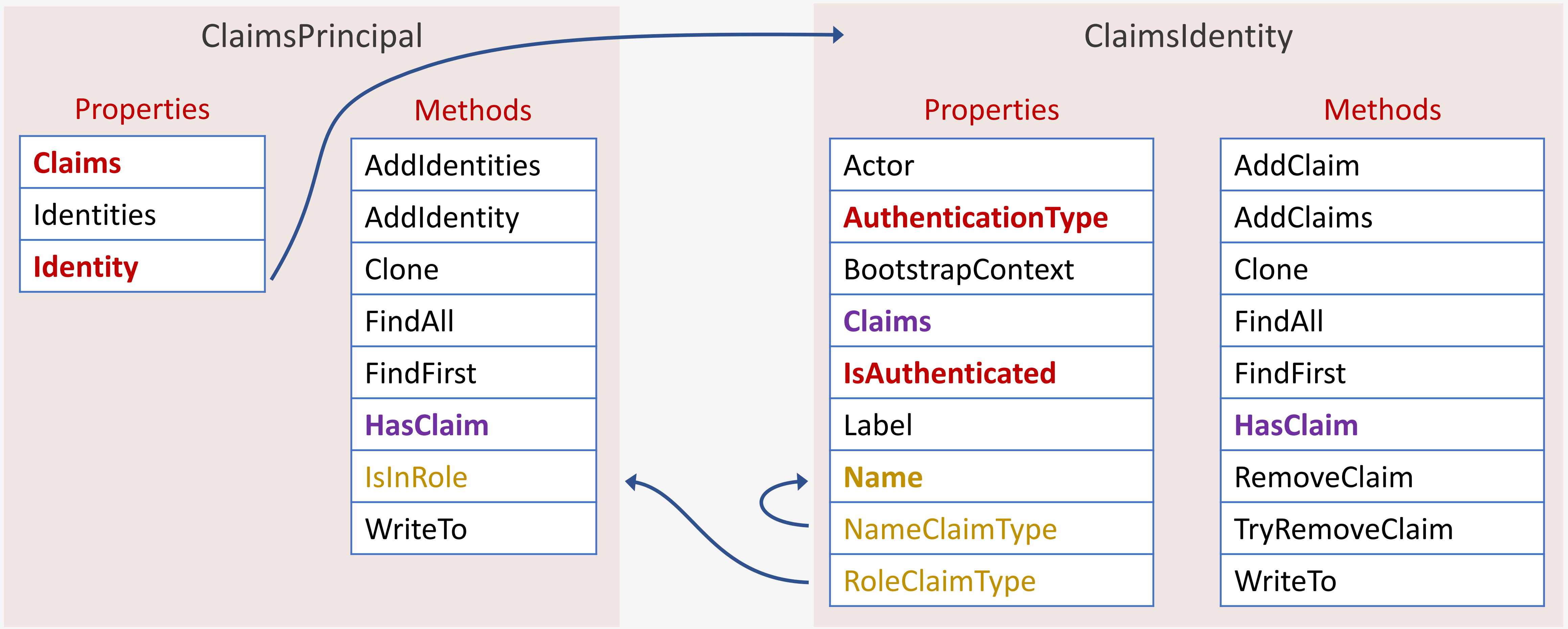

Learn C# Programming

Packt

Authentication and Authorization in .NET

- Authentication is the process of identifying a user or service
  - This produces a token that is transformed in a ClaimsPrincipal
- ASP.NET 8 provides a new <u>custom</u> authentication flow
  - It only targets SPAs to a its own backend
- Authorization determines whether the user can access a resource
- .NET provides three different strategies:
  1. Role-based: the identity has been given or not a given role (Boolean)
  2. Claim-based: the claim(s) values satisfy the requested condition
     - Values can be Boolean, integer, string, other, including a complex JSON
  3. Policy-based: a number of rules governing the access

ClaimsPrincipal and ClaimsIdentity



ClaimsPrincipal

Properties
Claims
Identities
Identity

Methods
AddIdentities
AddIdentity
Clone
FindAll
FindFirst
HasClaim
IsInRole
WriteTo

ClaimsIdentity

Properties
Actor
AuthenticationType
BootstrapContext
Claims
IsAuthenticated
Label
Name
NameClaimType
RoleClaimType

Methods
AddClaim
AddClaims
Clone
FindAll
FindFirst
HasClaim
RemoveClaim
TryRemoveClaim
WriteTo

🟨 Role-based authorization

🟪 Claims-based authorization

Tip #1: using IClaimsTransformation

- Claims are written in the token by the Identity Provider
  - A few of them have standard names like *name* and *email*
  - When using OIDC, there are functional claims like issuer, audience and scope
- .NET provides IClaimsTransformation to handcraft the principal

  ```
  Task<ClaimsPrincipal> TransformAsync(ClaimsPrincipal principal);
  ```

  - This method allows to add, remove or modify any Identity and its Claims
- The transformation service must be added in the DI:

  ```
  services.AddSingleton<IClaimsTransformation, DemoClaimInjector>();
  ```

AuthorizeAttribute

```
[Authorize]
public class Asset1Model : PageModel
{
 [AllowAnonymous] public void OnGet() {}
}
```

Requires authenticated access

Exception for the Get action

```
[Authorize(Roles="Administrators, Super")]
public class Asset2Model : PageModel {}
```

Role-based authentication

```
[Authorize(Policy = "SeniorTechStaff")]
public class Asset4Model : PageModel {}
```

Policy-based authentication

```
[Authorize(AuthenticationSchemes =
  JwtBearerDefaults.AuthenticationScheme)]
```

Needed in Web APIs to prevent redirection to the login page

Authorization Requirements

- It is a simple way to express the need for authorization checks
- The interface `IAuthorizationRequirement` is just an empty interface
- We have to implement the interface and optionally add properties:

```
public class TechStaffRequirement : IAuthorizationRequirement { }
```

- We then build a policy which may ask one or more requirements
  - All the requirements required by a policy must succeed to provide access
  - This translates in the **AND** conditions of all the requirements

```
authorizationOptions.AddPolicy(MyPolicies.TechStaff, builder =>
    { builder.Requirements.Add(new TechStaffRequirement()); });
```

Requirement handlers

- Requirements are verified by requirement handlers
- Each requirement may be satisfied by one or more handlers
  - This translates in the OR condition of the handlers evaluating a requirement

```
public class DeveloperRequirementHandler : AuthorizationHandler<TechStaffRequirement>
{
    protected override Task HandleRequirementAsync(
        AuthorizationHandlerContext context, TechStaffRequirement requirement)
    { ... }
}
```

```
public class ItproRequirementHandler     : AuthorizationHandler<TechStaffRequirement>
{
    protected override Task HandleRequirementAsync(
        AuthorizationHandlerContext context, TechStaffRequirement requirement)
    { ... }
}
```

Policies

```
options.AddPolicy(MyPolicies.SeniorTechStaff, builder =>
    {
        builder.Requirements.Add(new TechStaffRequirement());
        builder.Requirements.Add(new SeniorRequirement(10));
    });
```

- Authorization policies are enforced with the [Authorization] attribute

```
[Authorize(Policy = MyPolicies.SeniorTechStaff)]
```

- or through an imperative demand

```
var check = await _authorizationService.AuthorizeAsync(
    user:User,
    resource: null,
    requirement: new SportRequirement());
if(check.Succeeded) { ... }
```

IAuthorizationRequirementData in .NET 8

- Drastically reduce the code needed to authorize
  - No need to create the policy or the explicit requirement class
  - Just create the requirement handler and a custom attribute
- The custom attribute is the following:

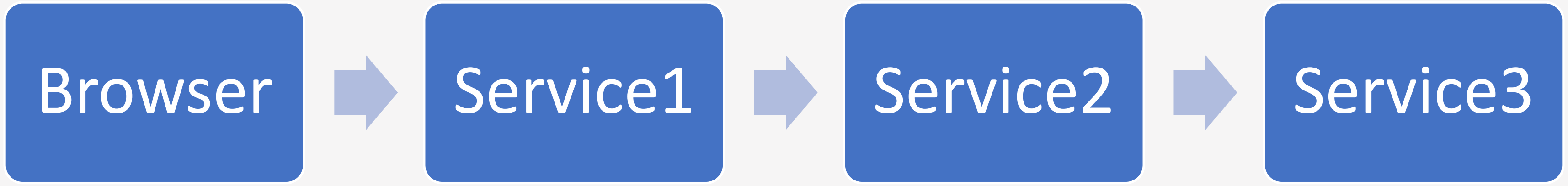
```
public class AuthorizeJuniorsAttribute : AuthorizeAttribute,
    IAuthorizationRequirement, IAuthorizationRequirementData
{
    public AuthorizeJuniorsAttribute(int years) => Years = years;
    public int Years { get; }
    public IEnumerable<IAuthorizationRequirement> GetRequirements()
    {
        yield return this;
    }
}
```

A first lap into the Authorization features in .NET

Tips (slides or demos)

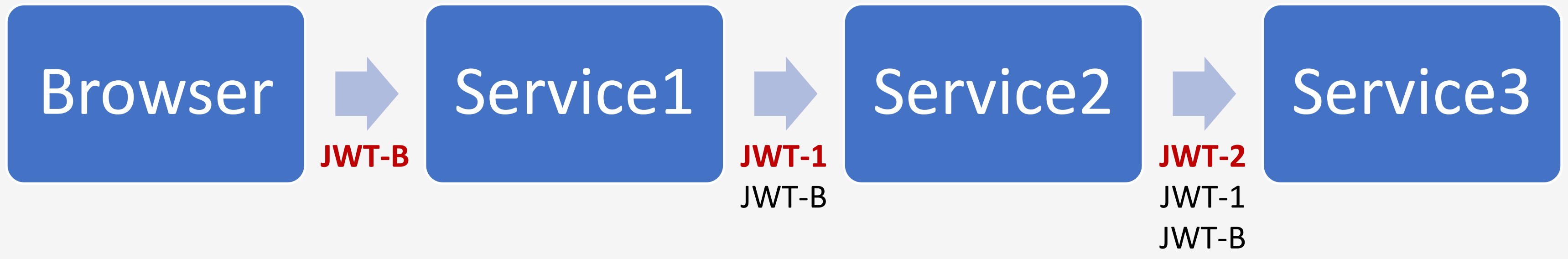
1. Microservices scenario
2. Scope in OIDC

Tip #2: microservices scenario 1/2



- Each block is independently authenticated to an OIDC IP provider
  - Browser is authenticated as the interactive user
  - Service1, 2, 3 have independent identities used to access local resources
- What if you need to provide audit/authorization in Service3 knowing the exact full chain of all identities causing the call?
  - Each service usually calls the next hop using a JWT
  - You can manually add a `X-WhateverYouWant` attribute to the HTTP request with the JWT that the service received from the caller

Tip #2: microservices scenario 2/2



1. Add the `Services.AddHttpContextAccessor()` service in ASP.NET
2. Implement the IClaimsTransformation interface:
   - Register the class in the DI as Scoped lifetime
   - Request the IHttpContextAccessor from the DI in the constructor
3. In `TransformAsync`, read the JWTs from the HTTP headers
4. For each JWT, add a new ClaimsIdentity to the ClaimsPrincipal

Tip #3: Using the scope in OIDC

- Scopes in OIDC determine which set of claims will appear in the token
- It is a convenient way to request *on-demand* a set of claims

- Scopes are a good way to avoid JWT being too powerful
- You can use different scopes in the same application to reduce the impacts of a stolen JWT being used for administrative purposes

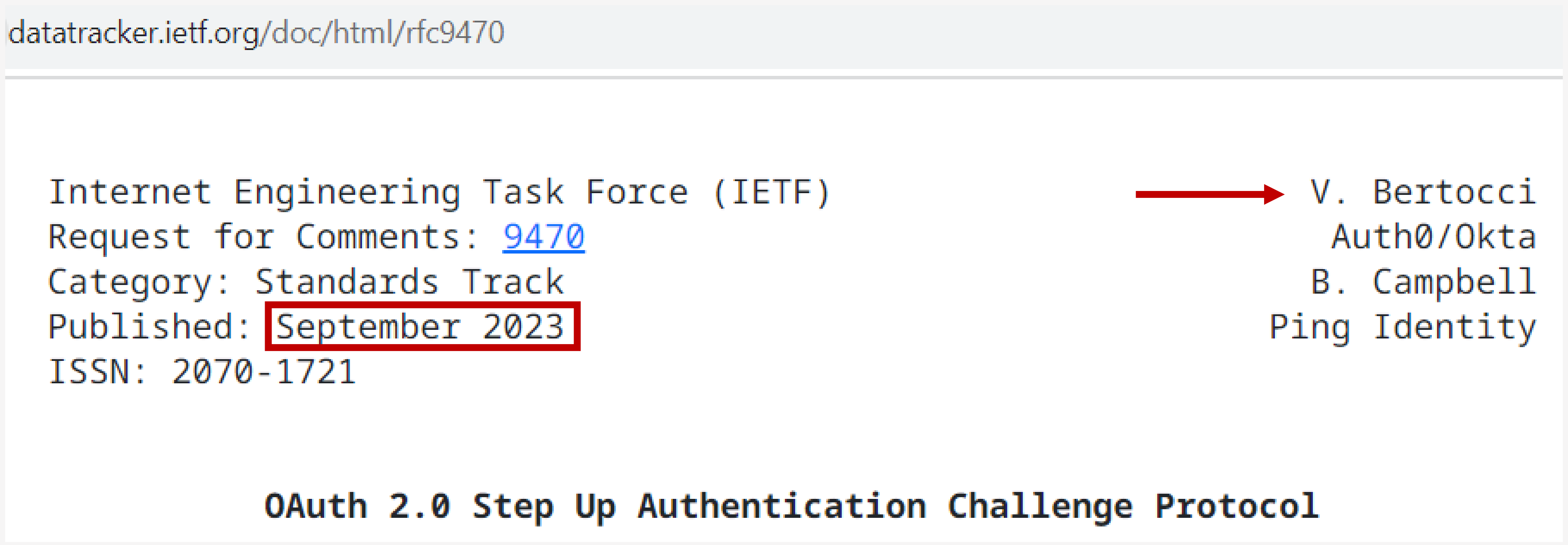
The Document management project

Scenario

- A document Web API serving documents in CRUD style
- A React front-end performing the List and CRUD operations
  - The UI does NOT forbid actions to allow testing the backend authorizations
- The rules are quite simple:
  - List permission allow to see the document properties only
  - Read permission includes the document content
  - Update, Delete permissions are always granted to the document owner
  - Users can share CRUD access to other users for specific documents
  - Admins always have the full CRUD access

Implementing the scenario

- A single requirement: `OperationAuthorizationRequirement`
  - Takes a string "action" specifying the action this requirement related to
  - We have 5 possible actions: List, Read, Create, Update, Delete
- Three requirement handlers:
  - `DocumentOperationAuthorizationHandler` is meant to control general access for users (typically admin) who can always have one ore more LCRUD access
  - `AuthorAuthorizationHandler` verifies that a specific document is owned by the current user
  - `InvitedAuthorizationHandler` verifies that the children share objects of a specific document allow the current user to access the document for a specific action (RUD)


datatracker.ietf.org/doc/html/rfc9470

```
Internet Engineering Task Force (IETF)                      V. Bertocci
Request for Comments: 9470                                   Auth0/Okta
Category: Standards Track                                   B. Campbell
Published: September 2023                                 Ping Identity
ISSN: 2070-1721
```

OAuth 2.0 Step Up Authentication Challenge Protocol

Step-up Authentication

Tip #3 Step-up in ASP.NET OIDC provider

- The IP Provider expect the parameter acr_values="mfa"
- In ASP.NET use the event OpenIdConnectEvents. OnRedirectToIdentityProvider

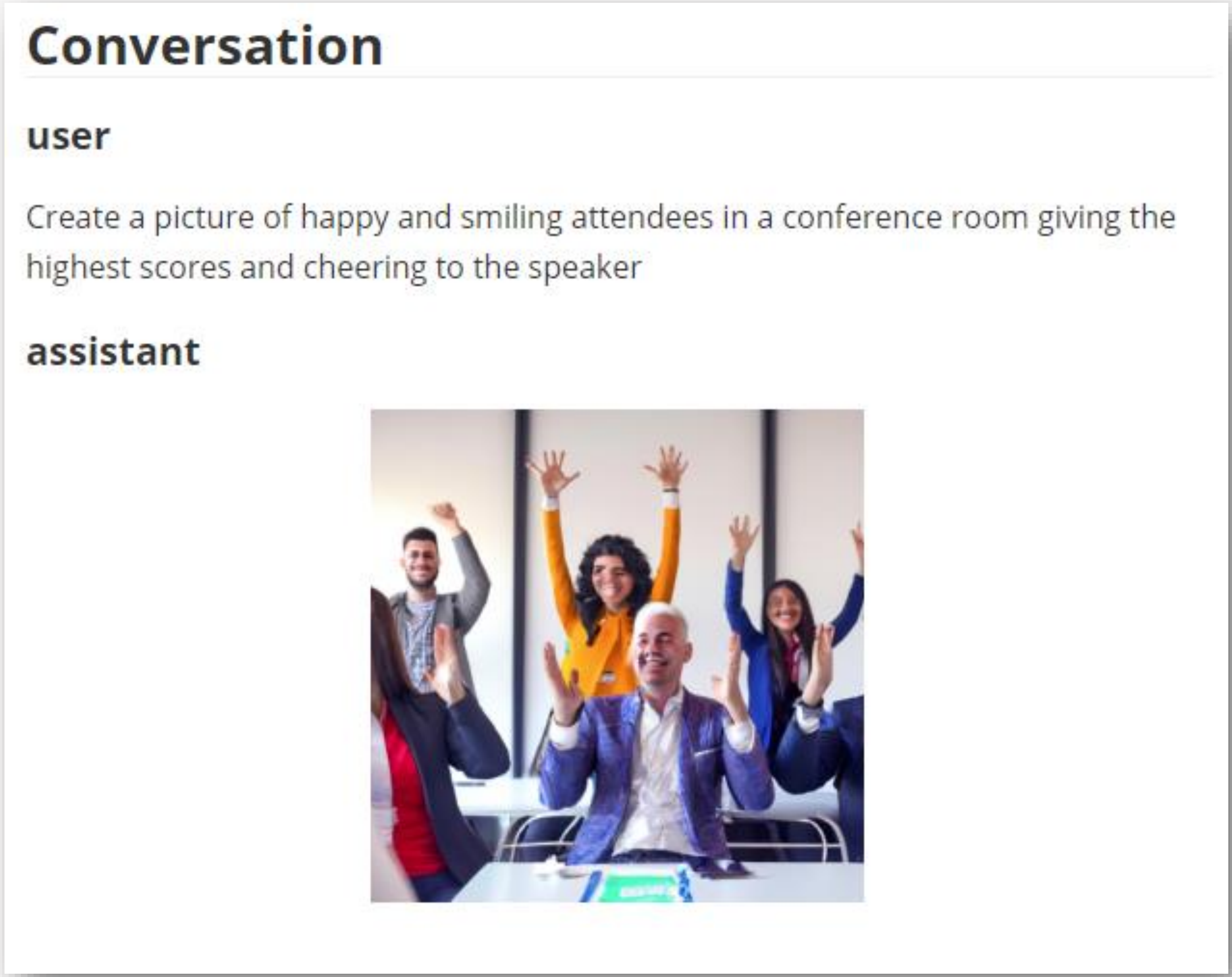
```
OnRedirectToIdentityProvider = ctx =>
{
    if (ctx.HttpContext.Items.TryGetValue("acr", out object value))
    {
        var acr = value as string ?? string.Empty;  // "mfa"
        ctx.ProtocolMessage.SetParameter("acr_values", acr);
    }
    return Task.CompletedTask;
},
```

- Any code setting HttpContext.Items["acr"] and calling Challenge will redirect the user to Keycloak asking the OTP code.

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

This is not school, but we love to get grades.
Please fill out our questioneers and leave us your feedback.
You may even win some cool rewards.



Conversation

user

Create a picture of happy and smiling attendees in a conference room giving the highest scores and cheering to the speaker

assistant