Implementing role-based authorization

(Demo prep)

- Remove all tokens from Postman
- Clear Authorization values in Postman

It's time to enable role-based security across our microservices.

Since our catalog of items is something that should be managed only by administrators, let's start by requiring the Admin role for all of our Catalog REST API operations.

In Identity repo

Start the identity microservice

In Catalog repo

1. Update **ItemsController.cs**:

```
[ApiController]
[Route("items")]
[Authorize(Roles = AdminRole)]
public class ItemsController : ControllerBase
{
   private const string AdminRole = "Admin";

   private readonly IRepository<Item> itemsRepository;
   …
}
```

2. Run the Identity and Catalog microservices

3. In Postman, use only these scopes: **openid profile catalog.fullaccess**

4. Get an access token for Player1 (https://localhost:5003/connect/authorize?prompt=login)

5. Try a GET /items (403 Forbidden)

6. Decode the token and notice no Role claim.

In Identity repo

7. Update appsettings.json

```
 "IdentityServerSettings": {
   …
```

```
  "ApiResources": [
   {
    "Name": "Catalog",
    "Scopes": [
     "catalog.fullaccess"
    ],
    "UserClaims": [
     "role"
    ]
   },
   …
  ]
 },
```

8. Start the services

9. Get a new token for player1@play.com

10. Decode the token and notice the new **Role** claim

11. Try a GET /items (403 Forbidden)

12. Get a new token for admin@play.com

13. Try a GET /items (200 OK)

In Inventory repo

14. Update **ItemsController.cs**:

```
[ApiController]
[Route("items")]
public class ItemsController : ControllerBase
{
  private const string AdminRole = "Admin";
  …

  [HttpGet]
  [Authorize]
  public async Task<ActionResult<IEnumerable<InventoryItemDto>>> GetAsync(Guid userId)
  {
    if (userId == Guid.Empty)
    {
      return BadRequest();
    }
```

```
      var currentUserId = User.FindFirstValue(JwtRegisteredClaimNames.Sub);

      if (Guid.Parse(currentUserId) != userId)
      {
         if (!User.IsInRole(AdminRole))
         {
            return Forbid();
         }
      }
      …
   }

   [HttpPost]
   [Authorize(Roles = AdminRole)]
   public async Task<ActionResult> PostAsync(GrantItemsDto grantItemsDto)
   {
    …
   }
}
```

In Identity repo

15. Update appsettings.json

```
  "IdentityServerSettings": {
    …
    "ApiResources": [
     {
      …
     {
       "Name": "Inventory",
       "DisplayName": "Inventory API",
       "Scopes": [
         "inventory.fullaccess"
       ],
       "UserClaims": [
         "role"
       ]
     }
    ]
  },
```

16. Start the Identity service

17. In Postman, use only these scopes: **openid profile inventory.fullaccess**

18. DELETE ALL TOKENS

19. Get a new token for Admin

20. In Inventory POST /items tab, use same Admin token to grant an item to Player1

21. In Inventory GET /items?userId={userId} tab, use same Admin token to get items for Player1

22. Get an access token for Player1 (NAME THE TOKEN)

23. Try the same GET /items?userId={userId} query

24. Register Player2

25. Get a new access token as Player2 (NAME THE TOKEN)

26. Try GET /items?userId={userId} using Player1's user id (401 Unauthorized)

27. Update **UsersController.cs**:

```
[ApiController]
[Route("users")]
[Authorize(Policy = LocalApi.PolicyName, Roles = Roles.Admin)]
public class UsersController : ControllerBase
{
  …
}
```

28. Update appsettings.json

```
"IdentityServerSettings":{
 …
 "ApiResources" : [
  …
  {
   "Name": "Inventory",
   …
  },
  {
   "Name": "Identity",
   "Scopes" : [
    "IdentityServerApi"
   ],
```

```
    "UserClaims":[
      "role"
    ]
  }
 ]
},
```

29. In Postman, use only these scopes: **openid profile IdentityServerApi**

30. DELETE ALL TOKENS

31. Try GET /users using Player1 token (403 Forbidden)

32. Try same GET using Admin token (200 OK)

In the next lesson we go beyond role-based authorization and into claims-based authorization to enable more strict rules on the access to our microservice REST APIs.