Securing microservices

(Demo prep)

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

Generalizing how to secure a microservice

Let's see how we can generalize and simplify the authentication configuration for Catalog, Inventory and any future microservices.

In Catalog repo

1. Remove **Microsoft.AspNetCore.Authentication.JwtBearer** PackageReference

2. Bump **Play.Common** package reference to **1.0.3**

3. Update **appsettings.json**:

```
 "ServiceSettings": {
  "ServiceName": "Catalog",
  "Authority": "https://localhost:5003"
 },
```

4. Update **Startup.cs**:

```
public void ConfigureServices(IServiceCollection services)
{
  serviceSettings = Configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();

  services.AddMongo()
        .AddMongoRepository<Item>("items")
        .AddMassTransitWithRabbitMq()
        .AddJwtBearerAuthentication();

  services.AddControllers(options =>
  {
     options.SuppressAsyncSuffixInActionNames = false;
  });
  …
}
```

5. Start Identity and Catalog and try GET /items in Catalog

In Inventory repo (Let student do as practice?)

At this point our Catalog microservice is properly configured for authorization. So it's time to do the same for our Inventory microservice. As a quick exercise you could try pausing this lesson now and try to enable authorization in the Inventory microservice by yourself, following what you have learned so far. You'll see it's a fairly straightforward procedure.

6. Bump **Play.Common** package reference to **1.0.3**

7. Update **appsettings.json**:

```
 "ServiceSettings": {
   "ServiceName": "Catalog",
   "Authority": "https://localhost:5003"
 },
```

8. Update **Startup.cs**:

```
public void ConfigureServices(IServiceCollection services)
{
   serviceSettings = Configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();

   services.AddMongo()
         .AddMongoRepository<Item>("items")
         .AddMassTransitWithRabbitMq()
         .AddJwtBearerAuthentication();

   services.AddControllers(options =>
   {
      options.SuppressAsyncSuffixInActionNames = false;
   });
   …
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
   …
   app.UseRouting();

   app.UseAuthentication();
   app.UseAuthorization();
  …
}
```

9. Update **ItemsController.cs**:

```
[ApiController]
[Route("items")]
[Authorize]
public class ItemsController : ControllerBase
{

}
```

10. Update **appsettings.Development.json**:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "Microsoft": "Warning",
      "Microsoft.AspNetCore.Authorization": "Information",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  }
}
```

11. Start Identity and Inventory and try GET /items/{userId} in Inventory (401 Unauthorized)

In Identity repo (Let student do as practice?)

12. Update **appsettings.json**:

```
"IdentityServerSettings": {
  "ApiScopes": [
    {
      "Name": "catalog.fullaccess"
    },
    {
      "Name": "inventory.fullaccess"
    }
  ],
  "ApiResources": [
    {
      "Name": "Catalog",
      "Scopes": [
        "catalog.fullaccess"
      ]
```

```
    },
    {
      "Name": "Inventory",
      "DisplayName": "Inventory API",
      "Scopes": [
        "inventory.fullaccess"
      ]
    }
  ]
},
```

13. Update **appsettings.Development.json**:

```
  "IdentityServerSettings": {
    "Clients": [
      {
        …
        "RedirectUris": [
          "urn:ietf:wg:oauth:2.0:oob"
        ],
        "AllowedScopes": [
          "openid",
          "profile",
          "catalog.fullaccess",
          "inventory.fullaccess"
        ],
        "AlwaysIncludeUserClaimsInIdToken": true
      }
    ]
  }
```

14. Start Identity and Inventory

15. Request a new token adding **inventory.fullaccess** scope

16. Try GET /items/{userId} in Inventory (200 OK)

In the next lesson we will secure our Users API via a few built in IdentityServer methods.