Securing microservices

(Demo prep)

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

Seeding users and roles

Before enabling role-based authentication across our microservices, let's see how we can create or seed a couple of roles, Admin and Player, and also an admin user the first time our Identity microservice boots.

In Identity repo

1. Add IdentitySettings to appsettings.json:

```
 "IdentitySettings": {
  "AdminUserEmail": "admin@play.com"
 },
```

2. Run in **Terminal** in **Play.Identity.Service** dir:

```
dotnet user-secrets init
dotnet user-secrets set "IdentitySettings:AdminUserPassword" "Pass@word1"
```

3. Add **IdentitySettings.cs** to Settings dir:

```
namespace Play.Identity.Service.Settings
{
    public class IdentitySettings
    {
        public string AdminUserEmail { get; init; }
        public string AdminUserPassword { get; init; }
    }
}
```

4. Add **Roles.cs**:

```
namespace Play.Identity.Service
{
    public static class Roles
    {
        public const string Admin = "Admin";
        public const string Player = "Player";
    }
```

}

Probably the best way to do this initialization is via a hosted service. A hosted service is just a class that you can register on Startup and that can execute asynchronous background task logic when the service host is starting.

5. Add the **HostedServices** directory.

6. Add **IdentitySeedHostedService.cs**:

```
namespace Play.Identity.Service.HostedServices
{
   public class IdentitySeedHostedService : IHostedService
   {
      private readonly IServiceScopeFactory serviceScopeFactory;
      private readonly IdentitySettings settings;

      public IdentitySeedHostedService(IServiceScopeFactory serviceScopeFactory, IOptions<IdentitySettings>
identityOptions)
      {
         this.serviceScopeFactory = serviceScopeFactory;
         settings = identityOptions.Value;
      }

      public async Task StartAsync(CancellationToken cancellationToken)
      {
         // Create a new scope to retrieve scoped services
         using var scope = serviceScopeFactory.CreateScope();

         var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<ApplicationRole>>();
         var userManager = scope.ServiceProvider.GetRequiredService<UserManager<ApplicationUser>>();

         await CreateRoleIfNotExistsAsync(Roles.Admin, roleManager);
         await CreateRoleIfNotExistsAsync(Roles.Player, roleManager);

         var adminUser = await userManager.FindByEmailAsync(settings.AdminUserEmail);
         if (adminUser == null)
         {
            adminUser = new ApplicationUser
            {
               UserName = settings.AdminUserEmail,
               Email = settings.AdminUserEmail,
            };

            IdentityResult userResult = await userManager.CreateAsync(adminUser, settings.AdminUserPassword);

            if (!userResult.Succeeded)
            {
```

```
            throw new Exception(string.Join(Environment.NewLine, userResult.Errors.Select(e => e.Description)));
          }

          var roleToUserResult = await userManager.AddToRoleAsync(adminUser, Roles.Admin);

          if (!roleToUserResult.Succeeded)
          {
            throw new Exception(string.Join(Environment.NewLine, roleToUserResult.Errors.Select(e =>
e.Description)));
          }
        }
      }

      private static async Task CreateRoleIfNotExistsAsync(string role, RoleManager<ApplicationRole> roleManager)
      {
        var roleExists = await roleManager.RoleExistsAsync(role);
        if (!roleExists)
        {
          var roleResult = await roleManager.CreateAsync(new ApplicationRole { Name = role });
          if (!roleResult.Succeeded)
          {
            throw new Exception(string.Join(Environment.NewLine, roleResult.Errors.Select(e => e.Description)));
          }
        }
      }

      public Task StopAsync(CancellationToken cancellationToken) => Task.CompletedTask;
    }
}
```

7. Update **Startup.cs**:

```
public void ConfigureServices(IServiceCollection services)
{
  …
  var identityServerSettings =
Configuration.GetSection(nameof(IdentityServerSettings)).Get<IdentityServerSettings>();

  services.Configure<IdentitySettings>(Configuration.GetSection(nameof(IdentitySettings)))
      .AddDefaultIdentity<ApplicationUser>()
      .AddRoles<ApplicationRole>()
      .AddMongoDbStores<ApplicationUser, ApplicationRole, Guid>
      (
        mongoDbSettings.ConnectionString,
        serviceSettings.ServiceName
      );
  …
```

```
    services.AddControllers();
    services.AddHostedService<IdentitySeedHostedService>();
    …
    });
}
```

8. Start service and verify there is an admin and two roles available now.

In the next lesson we will modify our user Registration logic so that we assign every new user to our new Player role.