Identity

(Demo prep)

- Delete Identity database
- Ensure Catalog DB has only these:

```
[
  {
    "name": "Potion",
    "description": "Restores some HP",
    "price": 5
  },
  {
    "name": "Antidote",
    "description": "Cures poison",
    "price": 7
  }
]
```

- Ensure catalogitems collection in Inventory DB is in sync with Catalog
- Disable “Save and fill personal info” in browser
- Hide favorites bar
- Disable extensions in browser
- Delete packages from %userprofile%\.nuget
- Remove Play.Identity.Service Postman collection

Configuring ASP.NET Core Identity with MongoDB

By default ASP.NET Core Identity uses Entity Framework Core and SQL Server as it’s backing store. However, since our existing microservices are already using MongoDB, it’s not a bad idea to use the same in our Identity microservice. Let’s see how to enable this.

1. Add more NuGet packages:

```
dotnet add package AspNetCore.Identity.MongoDbCore
dotnet add package Play.Common
```

2. Add **Entities** directory

3. Add ApplicationUser entity:

```
namespace Play.Identity.Service.Entities
{
  [CollectionName("Users")]
  public class ApplicationUser : MongoIdentityUser<Guid>
  {
```

```
        public decimal Gil { get; set; }
    }
}
```

4. Add ApplicationRole entity:

```
namespace Play.Identity.Service.Entities
{
    [CollectionName("Roles")]
    public class ApplicationRole : MongoIdentityRole<Guid>
    {
    }
}
```

5. Replace IdentityUser with **ApplicationUser** in **RegisterModel.cshtml.cs**

6. Replace IdentityUser with **ApplicationUser** in **_LoginPartial.cshtml**

7. Set the **StartingGil** in Register.cshtml.cs:

```
public class RegisterModel : PageModel
{
    private const decimal StartingGil = 100;
    …

    public async Task<IActionResult> OnPostAsync(string returnUrl = null)
    {
        returnUrl ??= Url.Content("~/");
        ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
        if (ModelState.IsValid)
        {
            var user = new ApplicationUser { UserName = Input.Email, Email = Input.Email, Gil = StartingGil };
            …
        }
        …
    }
}
```

8. Update appsettings.json:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
```

```
    "Microsoft": "Warning",
    "Microsoft.Hosting.Lifetime": "Information"
   }
  },
  "ServiceSettings": {
   "ServiceName": "Identity"
  },
  "MongoDbSettings": {
   "Host": "localhost",
   "Port": 27017
  },
  "AllowedHosts": "*"
}
```

9. Update Startup:

```
public void ConfigureServices(IServiceCollection services)
{
   BsonSerializer.RegisterSerializer(new GuidSerializer(BsonType.String));
   var serviceSettings = Configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();
   var mongoDbSettings = Configuration.GetSection(nameof(MongoDbSettings)).Get<MongoDbSettings>();

   services.AddDefaultIdentity<ApplicationUser>()
      .AddRoles<ApplicationRole>()
      .AddMongoDbStores<ApplicationUser, ApplicationRole, Guid>
      (
         mongoDbSettings.ConnectionString,
         serviceSettings.ServiceName
      );

   services.AddControllers();

   services.AddSwaggerGen(c =>
   {
      c.SwaggerDoc("v1", new OpenApiInfo { Title = "Play.Identity.Service", Version = "v1" });
   });
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
   if (env.IsDevelopment())
   {
      app.UseDeveloperExceptionPage();
      app.UseSwagger();
      app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "Play.Identity.Service v1"));
   }
```

```
    app.UseHttpsRedirection();

    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
        endpoints.MapRazorPages();
    });
}
```

10. On Play.Infra dir: docker-compose up -d

11. dotnet run

12. Browse to https://localhost:5003/identity/account/register

13. Try invalid emails and weak password

14. Register a new user

15. Explore the created user in DB

16. Browse to https://localhost:5003/Identity/Account/Login

17. Login with created user

In the next lesson we will add a proper REST API to manage users