Entity Framework

Goals

Until now, our repository uses a List in memory to store the photos. It's time to switch to a real SQL Server database. We're going to use `Entity Framework`.

We're going to: - Create a DbContext - Create a Repository that uses DbContext - Register the DbContext in the IoC container - Register the Repository in the IoC container - Create the database

Create a DbContext

- In the `PhotoSharingApplication.Infrastructure` project, add the `Microsoft.EntityFrameworkCore.SqlServer` NuGet Package
- Add a new `Data` Folder.
- In the `Data` folder, create a new class called `PhotoSharingContext` that inherits from `DbContext`.
- In the `PhotoSharingContext` class:
    - Add a constructor that accepts a `DbContextOptions<PhotoSharingDbContext>` and passes it to the base constructor.
    - Add a `DbSet<Photo>` property.
    - Add a `OnModelCreating` method that accepts a `ModelBuilder`.
    - Use the `ModelBuilder` to configure the `Photo` Entity so that its properties follow the same rules that we defined in the validator. The code becomes:

```
using Microsoft.EntityFrameworkCore;
using PhotoSharingApplication.Core.Entities;

namespace PhotoSharingApplication.Infrastructure.Data;

public class PhotoSharingDbContext : DbContext {
    public PhotoSharingDbContext(DbContextOptions<PhotoSharingDbContext> options) : base(options) {

    }
    protected override void OnModelCreating(ModelBuilder modelBuilder) {
        modelBuilder.Entity<Photo>()
            .Property(b => b.Title)
            .HasMaxLength(100);
        modelBuilder.Entity<Photo>()
            .Property(b => b.Description)
            .HasMaxLength(250);
        modelBuilder.Entity<Photo>()
            .Property(b => b.ContentType)
            .HasMaxLength(30);
    }
    public DbSet<Photo> Photos { get; set; }
}
```

Create a Repository that uses DbContext

- In the `PhotoSharingApplication.Infrastructure` project, under the `Repositories` folder, add a new class `PhotosRepositoryEF` that implements the `IPhotosRepository` interface.
- Add a constructor that accepts a `PhotoSharingDbContext` and saves it in a private readonly field.
- Implement the methods so that they use the dbContext field.

The code becomes:

```
using Microsoft.EntityFrameworkCore;
using PhotoSharingApplication.Core.Entities;
using PhotoSharingApplication.Core.Interfaces;
using PhotoSharingApplication.Infrastructure.Data;

namespace PhotoSharingApplication.Infrastructure.Repositories;

public class PhotosRepositoryEF : IPhotosRepository {
    private readonly PhotoSharingDbContext dbContext;

    public PhotosRepositoryEF(PhotoSharingDbContext dbContext) {
        this.dbContext = dbContext;
    }
    public async Task AddPhotoAsync(Photo photo) {
        dbContext.Photos.Add(photo);
        await dbContext.SaveChangesAsync();
    }

    async Task<IEnumerable<Photo>> IPhotosRepository.GetAllPhotosAsync() => await dbContext.Photos.ToListAsync();

    public async Task<Photo?> GetPhotoByIdAsync(int id) => await dbContext.Photos.FirstOrDefaultAsync(p => p.Id == id);
}
```

Register the DbContext and the Repository in the IoC container

In the `PhotoSharingApplication.Infrastructure` project, add a new `ServiceCollectionExtensions` class. - Let the class be public and static. - Add a method called `AddPhotoSharingDb` that accepts a `this IServiceCollection` and a `string connectionString` and returns an `IServiceCollection` . - Register the DbContext in the IoC container. - Register the Repository in the IoC container. The code becomes:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using PhotoSharingApplication.Core.Interfaces;
using PhotoSharingApplication.Infrastructure.Data;
using PhotoSharingApplication.Infrastructure.Repositories;

namespace PhotoSharingApplication.Infrastructure;

public static class ServiceCollectionExtensions {
    public static IServiceCollection AddPhotoSharingDb(this IServiceCollection services, string connectionString) {
        services.AddDbContext<PhotoSharingDbContext>(options => options.UseSqlServer(connectionString));
        services.AddScoped<IPhotosRepository, PhotosRepositoryEF>();
        return services;
    }
}
```

- In the `Program.cs` file, add a call to `AddPhotoSharingDb` with the connection string.

```
builder.Services
    .AddPhotoSharingServices()
    .AddPhotoSharingDb(builder.Configuration.GetConnectionString("Default"));
```

- Add the connection string to the `appsettings.json` file

```
"ConnectionStrings": {
    "Default": "Server=(localdb)\\mssqllocaldb;Database=PhotoSharing;Trusted_Connection=True;MultipleActiveResultSets=true"
}
```

- In the `ServiceCollectionExtensions.cs` of the `PhotoSharingApplication.Web` project, remove the registration of the `PhotosRepositoryList` . The code becomes:

```
using FluentValidation.AspNetCore;
using PhotoSharingApplication.Core.Interfaces;
using PhotoSharingApplication.Core.Services;
using PhotoSharingApplication.Core.Validators;

namespace PhotoSharingApplication.Web;

public static class ServiceCollectionExtensions {
    public static IServiceCollection AddPhotoSharingServices(this IServiceCollection services) {
        //services for Validation
        services.AddFluentValidation(fv => fv.RegisterValidatorsFromAssemblyContaining<PhotoValidator>());

        services.AddScoped<IPhotosService, PhotosService>();
        return services;
    }
}
```

Create the database

- In the `PhotoSharingApplication.Web` project, install the `Microsoft.EntityFrameworkCore.Tools` NuGet Package.
- In the Package Manager Console, run the following commands:

`Add Migration InitialCreate` followed by `Update-Database`

At this point, navigating to the `/Photos` URL will display an empty page.
The Upload should still work, but this time the data should end up in the db.

References

- https://docs.microsoft.com/en-us/ef/core/get-started/overview/install
- https://docs.microsoft.com/en-us/ef/core/dbcontext-configuration/
- https://docs.microsoft.com/en-us/ef/core/modeling/entity-properties?tabs=fluent-api%2Cwithout-nrt
- https://docs.microsoft.com/en-us/ef/core/querying/
- https://docs.microsoft.com/en-us/ef/core/saving/basic
- https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=vs
- https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/sql?view=aspnetcore-6.0&tabs=visual-studio