Goals

- Creating the project structure
- Implementing the logic for the PhotoGallery
- Implementing the logic for the Upload
- Implementing the logic for the Details

In this lab, we're going to create the initial structure of the PhotoSharing application.
Since we want all our parts to be testable and losely coupled, we're going to use the CLEAN architecture, making sure that each page can be dependent on abstractions and that each concern is neatly separated.

We're going to: - Create three projects for - **Core** (with Entities, Interfaces and Services) - **Infrastructure** (with repository implementations) - **Web** (with Razor Pages) - In the **Core Project**: - Define the Photo Entity in the Core project - Define interfaces for a Service and a Repository to - Get all Photos - Add a Photo - Get One Photo By Id - Implement the Service for the Photos in the Core project - Let the Service depend on an abstraction of a Repository - In the **Infrastructure** project: - Implement the Repository to work with a static List of Photos (we will switch to a db on a later lab) - In the **Web** project: - Register the interfaces and classes in the Service Container

Create the initial application structure

- Open Visual Studio 2022 (or later)
- Create a New Project of type `ASP.NET Core Web App` (not the one that specifies `MVC` but the one that specifies `Razor Pages`)
    - Name the solution `PhotoSharingApplication`
    - Name the Project `PhotoSharingApplication.Web`
- Add to the solution a new project of type `Class Library`
    - Name the project `PhotoSharingApplication.Core`
- Add to the solution a new project of type `Class Library`
    - Name the project `PhotoSharingApplication.Infrastructure`
- In the `Infrastructure` project, add a project reference to the `Core` project
- In the `Web` project, add a project reference to the `Core` and to the `Infrastructure` projects

Define the Photo Entity

- In the `Core` project, add a new folder `Entities`
- In the `Entities` folder, add a new file `Photo.cs`
- In the `Photo.cs` file, add the following code:

```
namespace PhotoSharingApplication.Core.Entities;

public class Photo {
    public int Id { get; set; }
    public string Title { get; set; } = string.Empty;
    public string Description { get; set; } = string.Empty;
}
```

Define the Interfaces

- In the `Core` project, add a new `Interfaces` folder
- In the `Interfaces` folder, add a new `IPhotosService` interface
- In the `IPhotosService.cs` file, add the following code:

```
using PhotoSharingApplication.Core.Entities;

namespace PhotoSharingApplication.Core.Interfaces;

public interface IPhotosService {
    Task<IEnumerable<Photo>> GetAllPhotosAsync();
    Task<Photo?> GetPhotoByIdAsync(int id);
    Task AddPhotoAsync(Photo photo);
}
```

- In the `Interfaces` folder, add a new `IPhotosRepository` interface
- In the `IPhotosRepository.cs` file, add the following code:

```
using PhotoSharingApplication.Core.Entities;

namespace PhotoSharingApplication.Core.Interfaces;

public interface IPhotosRepository {
    Task<IEnumerable<Photo>> GetAllPhotosAsync();
    Task<Photo?> GetPhotoByIdAsync(int id);
    Task AddPhotoAsync(Photo photo);
}
```

Implement the Service

- In the `Core` project, add a new `Services` folder
- In the `Services` folder, add a new `PhotosService` class
- Let the class be dependent on an `IPhotosRepository` by explicitly stating it in the constructor
- Use the dependency when implementing the methods of the `IPhotosService` interface
- The `PhotosService.cs` file becomes:

```
using PhotoSharingApplication.Core.Entities;
using PhotoSharingApplication.Core.Interfaces;

namespace PhotoSharingApplication.Core.Services;

public class PhotosService : IPhotosService {
    private readonly IPhotosRepository photosRepository;

    public PhotosService(IPhotosRepository photosRepository) => this.photosRepository = photosRepository;
    public Task AddPhotoAsync(Photo photo) => photosRepository.AddPhotoAsync(photo);

    public Task<IEnumerable<Photo>> GetAllPhotosAsync() => photosRepository.GetAllPhotosAsync();

    public Task<Photo?> GetPhotoByIdAsync(int id) => photosRepository.GetPhotoByIdAsync(id);
}
```

Implement the Repository

- In the `Infrastructure` project, add a `Repositories` folder
- In the `Repositories` folder, add a `PhotosRepositoryList` class
- Let the class implement the `IPhotosRepository` interface
- Make use of a private static `List<Photo>` when implementing the methods of the `IPhotosRepository` interface
- The code becomes:

```
using PhotoSharingApplication.Core.Entities;
using PhotoSharingApplication.Core.Interfaces;

namespace PhotoSharingApplication.Infrastructure.Repositories;

public class PhotosRepositoryList : IPhotosRepository {
    private static List<Photo> photos;
    public PhotosRepositoryList() => photos = new() {
        new() { Id = 1, Title = "One Photo", Description = "The first photo" },
        new() { Id = 2, Title = "Another Photo", Description = "The second photo" }
    };
    public Task AddPhotoAsync(Photo photo) {
        photo.Id = photos.Max(p => p.Id) + 1;
        photos.Add(photo);
        return Task.CompletedTask;
    }

    public Task<IEnumerable<Photo>> GetAllPhotosAsync() => Task.FromResult((IEnumerable<Photo>)photos);

    public Task<Photo?> GetPhotoByIdAsync(int id) => Task.FromResult(photos.FirstOrDefault(p => p.Id == id));
}
```

Register the services

Instead of registering every service directly on the `Program.cs` file, we will group the registration of multiple services into an extension method and invoke that in the `Program.cs` file.

- In the `Web` project, add a new class named `ServiceCollectionExtensions`
- Mark the `ServiceCollectionExtensions` class as `static`
- Add a new static method named `AddPhotoSharingServices` to the `ServiceCollectionExtensions` class
- Make this method an extension method for the `IServiceCollection` interface
- Register the `PhotosRepositoryList` as a `Singleton` implementation for the `IPhotosRepository` type
- Register the `PhotosService` as a `Scoped` implementation for the `IPhotosService` type
- The code becomes:

```
 using PhotoSharingApplication.Core.Interfaces;
using PhotoSharingApplication.Core.Services;
using PhotoSharingApplication.Infrastructure.Repositories;

namespace PhotoSharingApplication.Web;

public static class ServiceCollectionExtensions {
    public static IServiceCollection AddPhotoSharingServices(this IServiceCollection services) {
        services.AddSingleton<IPhotosRepository, PhotosRepositoryList>();
        services.AddScoped<IPhotosService, PhotosService>();
        return services;
    }
}
```

Results

At the end of this lab, you will have the initial structure for the Photo Sharing Application, including the implementation of the logic of the PhotoGallery, Details and Upload features.
In the next lab we're going to implement the user interface for those features.

Resources

- https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
- https://github.com/ardalis/CleanArchitecture
- https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0
- https://andrewlock.net/exploring-dotnet-6-part-2-comparing-webapplicationbuilder-to-the-generic-host/