Blazor Integration

Goal

Build the client side of the Comments section integrating Blazor Components into the Photo/Details Razor Page. - One Blazor component - CommentsForPhotoComponent
Note: we could create different components and let them communicate with each other, but it's out of scope for this course.
For now we use a fake service to read / write the comments. We'll build a real service later.

At first, we just try to add the Counter to the Details Razor Page, to see if the integration works. Then, we build the `CommentsForPhotoComponent`.
We concentrate on reading the existing comments first, writing a new comment at a later step.

Steps

- Add a new project of type **Blazor WebAssembly**. Name the App `PhotoSharingApplication.Blazor`.
- In the Additional information dialog, select the **ASP.NET Core hosted** checkbox when creating the Blazor WebAssembly app.

We have 3 projects added to the solution: - `PhotoSharingApplication.Blazor.Client` - `PhotoSharingApplication.Blazor.Server` - `PhotoSharingApplication.Blazor.Shared`.

We will remove the `Server` and the `Shared` in the future, but we can use them to steal some code from them and configure our Web app.

Blazor.Client

- Delete the `wwwroot/index.html` and `wwwroot.favicon.ico` files.
- Delete the following lines in `Program.cs`:

```
- builder.RootComponents.Add<App>("#app");
- builder.RootComponents.Add<HeadOutlet>("head::after");
```

PhotoSharingApplication.Web

- Add the Package `"Microsoft.AspNetCore.Components.WebAssembly.Server`
- Add a Project Reference to th `PhotoSharingApplication.Blazor.Client` project.
- In `program.cs`, between `app.UseHttpsRedirection();` and `app.UseStaticFiles();` add the following:

```
app.UseBlazorFrameworkFiles();
```

In the `Pages/Shared/_Layout.cshtml` file, add the following:

```
<base href="~/" />
<link rel="stylesheet" href="~/css/app.css" />
<link rel="stylesheet" href="~/PhotoSharingApplication.Blazor.Client.styles.css" asp-append-version="true" />
```

In the `Pages/Photos/Details.cshtml` file, add the following:

```
@using PhotoSharingApplication.Blazor.Client.Pages
```

```
<component type="typeof(Counter)" render-mode="WebAssemblyPrerendered"/>

@section Scripts {
    <script src="_framework/blazor.webassembly.js"></script>
}
```

Run the application, navigate to the details of a photo and verify that the counter is rendered and that it is interactive.

We successfully integrated the Blazor Client into the `Photo/Details` Razor Page. Now it's time to build the actual Blazor Component.
We want the component to be agnostic of technologies and implementation, so we'll follow the same pattern of service and repository as we did for the server.
Because the entities and validators will be shared between client side and server side, we will move them to a new project.
Since the component will be prerendered, we will have to implement services and repositories both on the client and on the server. On the client, our repository will talk to the webapi using an `HttpClient`, while on the server the repository willl talk to the Api Controller by invoking it directly (no need for an Http call, since we're already in process on the server). For now, since we don't have a webapi yet, we'll just use a fake `List<Comment>` to simulate the repository.

PhotoSharingApplication.Shared

Let's start by creating a shared project where we can move models and validators - Add a new project of type Class Library. Name it `PhotoSharingApplication.Shared`.

- Move the `Entities` folder from the `PhotoSharingApplication.Core` project to the `PhotoSharingApplication.Shared` project.
- Move the `Validators` folder from the `PhotoSharingApplication.Core` project to the `PhotoSharingApplication.Shared` project.
- In the `Entities` folder, add a new file called `Comment.cs` with the `Comment` class containing properties for `Id`, `Title` and `Body`. - In the `Validators` folder, add a new file called `CommentValidator.cs` with the `CommentValidator` class. Add rules to ensure that the `Title` and `Body` are not empty and that they are not longer than 100 and 250 characters respectively.

PhotoSharingApplication.Blazor.Core

Now let's focus on the core of our client side, with interfaces and the service.

- Add a new project of type Class Library. Name it `PhotoSharingApplication.Blazor.Core`.
- Add a new folder. Name it `Interfaces`.
- In the `Interfaces` folder, add a new file called `ICommentService.cs` with the `ICommentService` interface. Add methods to get comments for a specific photo given the photoId, add a comment and get one comment given its id.

```
using PhotoSharingApplication.Shared.Entities;

namespace PhotoSharingApplication.Blazor.Core.Interfaces;

public interface ICommentsService {
    Task<IEnumerable<Comment>> GetCommentsForPhotoAsync(int photoId);
    Task<Comment?> GetCommentByIdAsync(int id);
    Task<Comment> AddCommentAsync(Comment comment);
}
```

- In the `Interfaces` folder, add a new file called `ICommentRepository.cs` with the `ICommentRepository` interface. Add the same methods as the service.

```
using PhotoSharingApplication.Shared.Entities;

namespace PhotoSharingApplication.Blazor.Core.Interfaces;

public interface ICommentsRepository {
    Task<IEnumerable<Comment>> GetCommentsForPhotoAsync(int photoId);
    Task<Comment?> GetCommentByIdAsync(int id);
    Task<Comment> AddCommentAsync(Comment comment);
}
```

- Add a new folder. Name it `Services`.
- In the `Services` folder, add a new file called `CommentService.cs` with the `CommentService` class. The `CommentsService` class has a dependency on the `ICommentRepository` interface. Implement the methods defined in the interface.

```
using FluentValidation;
using PhotoSharingApplication.Blazor.Core.Interfaces;
using PhotoSharingApplication.Shared.Entities;
using PhotoSharingApplication.Shared.Validators;

namespace PhotoSharingApplication.Blazor.Core.Services;

public class CommentsService : ICommentsService {
    private readonly ICommentsRepository repository;
    private readonly CommentValidator validator;

    public CommentsService(ICommentsRepository repository, CommentValidator validator) {
        this.repository = repository;
        this.validator = validator;
    }
    public Task<Comment> AddCommentAsync(Comment comment) {
        validator.ValidateAndThrow(comment);
        return repository.AddCommentAsync(comment);
    }

    public Task<Comment?> GetCommentByIdAsync(int id) => repository.GetCommentByIdAsync(id);

    public Task<IEnumerable<Comment>> GetCommentsForPhotoAsync(int photoId) => repository.GetCommentsForPhotoAsync(photoId);
}
```

PhotoSharingApplication.Blazor.Infrastructure

For the client infrastructure, we'll add a Repository that uses a fake list. We will replace it with two real repositories later: - One for the prerendering phase happening on the server, where we will talk to the Api Controller - One for the client side initialization, where we will invoke our Api through an `HttpClient`.

These are the steps we need to take to implement the infrastructure: - Add a new project of type Class Library. Name it `PhotoSharingApplication.Blazor.Infrastructure`. - Add a new folder. Name it `Repositories`. - In the `Repositories` folder, add a new file called `CommentRepositoryList.cs` with the `CommentRepositoryList` class. Implement the methods defined in the `ICommentsRepository` interface by using a private static `List<Comment>`.

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

namespace PhotoSharingApplication.Blazor.Infrastructure.Repositories;

public class CommentsRepositoryList : ICommentsRepository {
    private readonly List<Comment> comments;
    public CommentsRepositoryList() {
        comments = new List<Comment>() {
                new() { Id = 1, PhotoId = 1, Title = "One Comment for Photo 1", Body = "Lorem ipsum" },
                new() { Id = 2, PhotoId = 1, Title = "Anoter Comment for Photo 1", Body = "Bacon ipsum" },
                new() { Id = 3, PhotoId = 2, Title = "One Comment for Photo 2", Body = "Yadam ipsum" }
            };
    }
    public Task<Comment> AddCommentAsync(Comment comment) {
        comment.Id = comments.Max(c => c.Id) + 1;
        comments.Add(comment);
        return Task.FromResult(comment);
    }

    public Task<Comment?> GetCommentByIdAsync(int id) {
        return Task.FromResult(comments.FirstOrDefault(p => p.Id == id));
    }

    public Task<IEnumerable<Comment>> GetCommentsForPhotoAsync(int photoId) {
        return Task.FromResult(comments.Where(c => c.PhotoId == photoId));
    }
}
```

PhotoSharingApplication.Blazor.Client

- Register The service, the repository and the validator in the `Program.cs` file of the `PhotoSharingApplication.Blazor.Client` project, which becomes

```
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
using PhotoSharingApplication.Blazor.Core.Interfaces;
using PhotoSharingApplication.Blazor.Infrastructure.Repositories;
using PhotoSharingApplication.Blazor.Core.Services;
using PhotoSharingApplication.Shared.Validators;

var builder = WebAssemblyHostBuilder.CreateDefault(args);

builder.Services.AddSingleton<ICommentsRepository, CommentsRepositoryList>();
builder.Services.AddScoped<ICommentsService, CommentsService>();
builder.Services.AddSingleton<CommentValidator>();
builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });

await builder.Build().RunAsync();
```

PhotoSharingApplication.Core

As already stated, our component will be prerendered, so we will need to specify which service and repository to use server side to prerender the html to send to the client on first call. This is why we need a new implementation of the repository and service.

- In the `Service` folder, add a new `Client` folder
- In the `Services/Client` folder, add a new file called `CommentsService.cs` with the `CommentsService` class. Implement the `ICommentsService` interface of the `PhotoSharingApplication.Blazor.Core.Interfaces` namespace by using the validator and the repository.

```
using FluentValidation;
using PhotoSharingApplication.Shared.Entities;
using PhotoSharingApplication.Shared.Validators;
using PhotoSharingApplication.Blazor.Core.Interfaces;

namespace PhotoSharingApplication.Core.Services.Client;

public class CommentsService : ICommentsService {
    private readonly ICommentsRepository repository;
    private readonly CommentValidator validator;

    public CommentsService(ICommentsRepository repository, CommentValidator validator) {
        this.repository = repository;
        this.validator = validator;
    }
    public Task<Comment> AddCommentAsync(Comment comment) {
        validator.ValidateAndThrow(comment);
        return repository.AddCommentAsync(comment);
    }

    public Task<Comment?> GetCommentByIdAsync(int id) => repository.GetCommentByIdAsync(id);

    public Task<IEnumerable<Comment>> GetCommentsForPhotoAsync(int photoId) => repository.GetCommentsForPhotoAsync(photoId);
}
```

PhotoSharingApplication.Infrastructure

- In the `Repository` folder, add a new `Client` folder.
- In the `Repository/Client` folder, add a new file called `CommentsRepositoryList.cs` with the `CommentsRepositoryList` class. Implement the `ICommentsRepository` of the interface of the `PhotoSharingApplication.Blazor.Core.Interfaces` namespace by using a `List<Comment>`. Add a couple of comments for the first couple of photos, so that we can test the CommentsForPhotoComponent later. Make sure that the `AddCommentAsync` returns the newly created comment.

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

namespace PhotoSharingApplication.Infrastructure.Repositories.Client;

public class CommentsRepositoryList : ICommentsRepository {
    private readonly List<Comment> comments;
    public CommentsRepositoryList() {
        comments = new List<Comment>() {
                new() { Id = 1, PhotoId = 1, Title = "One Comment for Photo 1", Body = "Lorem ipsum" },
                new() { Id = 2, PhotoId = 1, Title = "Anoter Comment for Photo 1", Body = "Bacon ipsum" },
                new() { Id = 3, PhotoId = 2, Title = "One Comment for Photo 2", Body = "Yadam ipsum" }
            };
    }
    public Task<Comment> AddCommentAsync(Comment comment) {
        comment.Id = comments.Max(c => c.Id) + 1;
        comments.Add(comment);
        return Task.FromResult(comment);
    }

    public Task<Comment?> GetCommentByIdAsync(int id) {
        return Task.FromResult(comments.FirstOrDefault(p => p.Id == id));
    }

    public Task<IEnumerable<Comment>> GetCommentsForPhotoAsync(int photoId) {
        return Task.FromResult(comments.Where(c => c.PhotoId == photoId));
    }
}
```

PhotosSharingApplication.Web

- Register the Service and the Repository in the ServiceCollectionExtensions.cs file.

```
services.AddSingleton<PhotoSharingApplication.Blazor.Core.Interfaces.ICommentsRepository, Infrastructure.Repositories.Client.Comment
services.AddScoped<PhotoSharingApplication.Blazor.Core.Interfaces.ICommentsService, PhotoSharingApplication.Core.Services.Client.Com
```

PhotoSharing.Blazor.Client

With the logic and data access in place, we can now build the Blazor Component.
The first step is to create a new folder in the `PhotoSharingApplication.Blazor.Client` project called `Components`.
We'll add a new **Razor Component** called `CommentsForPhotoComponent` in the `Components` folder.
We can peak at the `FetchData` page to see how the component has to be built. - Instead of asking for an `HttpClient`, we'll use the `ICommentsService`. - Instead of an array of `WeatherForecast`, we'll have a `List<Comment>`. - In the table, we'll render the `Title` and the `Body` of each `comment` contained in our model. - We also need a `[Parameter]` to pass the `PhotoId` to the component. - During the `OnInitializedAsync` method, we'll call the `ICommentsService` to get the comments for the photo.

```
@inject ICommentsService service

<h3>Comments For Photo @PhotoId</h3>

@if (comments is null)
{
    <p><em>No Comment</em></p>
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Title</th>
                <th>Body</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var comment in comments)
            {
                <tr>
                    <td>@comment.Title</td>
                    <td>@comment.Body</td>
                </tr>
            }
        </tbody>
    </table>
}

@code {
    [Parameter]
    public int PhotoId { get; set; }

    private List<Comment>? comments;

    protected override async Task OnInitializedAsync(){
        comments = (await service.GetCommentsForPhotoAsync(PhotoId)).ToList();
    }
}
```

PhotoSharingApplication.Web

In the `Pages/Photos/Details.cshtml` file, add the blazor component by using the `component` tag helper and the script for Blazor WebAssembly:

```
@using PhotoSharingApplication.Blazor.Client.Components
```

```
<component type="typeof(CommentsForPhotoComponent)" render-mode="WebAssemblyPrerendered" param-PhotoId="@Model.Photo.Id"/>

@section Scripts {
    <script src="_framework/blazor.webassembly.js"></script>
}
```

If you run the application and navigate to the details of a photo, you should be able to see the comments for that photo.

PhotoSharing.Blazor.Client

Now it's time to implement the feature to add a comment. Let's go back to our `CommentsForPhotoComponent`.
We need a new variable of type `Comment` to hold the new comment. We will initialize it during the `OnInitializedAsync` with a new `Comment` object with a `PhotoId` equal to the component parameter with the same name.
In the html section, we'll add an `EditForm`, setting the `Model` to the comment variable. We'll have an `InputText` with a `bind-Value` set to `comment.Title` and a `InputTextArea` with a `bind-Value` set to `comment.Body`.
In order to use `FluentValidation`, let's add the `Blazored.FluentValidation` package and the `@using Blazored.FluentValidation` in the `_Imports.razor` file.
In the `EditForm`, instead of using the `<DataAnnotationValidator />`, let's use a `<FluentValidationValidator />`, together with a `<ValidationSummary />`.
Let's handle the `OnValiSubmit` of the `EditForm` by calling a `HandleValidSubmit` method, where we `await` the `service.AddCommentAsync(comment)` and then add the result to the comments list.

The component becomes:

```
@inject ICommentsService service

<h3>Add Comment</h3>

<EditForm Model="@comment" OnValidSubmit="@HandleValidSubmit">
    <FluentValidationValidator />
    <ValidationSummary  class="text-danger"/>

    <div class="mb-3">
        <label class="form-label">
            Title:
            <InputText @bind-Value="comment.Title" class="form-control"/>
        </label>
    </div>
    <div class="mb-3">
        <label class="form-label">
            Body:
            <InputTextArea @bind-Value="comment.Body" class="form-control"/>
        </label>
    </div>
    <div class="mb-3">
        <button type="submit" class="btn btn-primary">Submit</button>
    </div>
</EditForm>

<h3>Comments For Photo @PhotoId</h3>

@if (comments is null)
{
    <p><em>No Comment</em></p>
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Title</th>
                <th>Body</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var comment in comments)
```

```
            {
                <tr>
                    <td>@comment.Title</td>
                    <td>@comment.Body</td>
                </tr>
            }
        </tbody>
    </table>
}

@code {
    [Parameter]
    public int PhotoId { get; set; }

    private List<Comment>? comments;

    private Comment comment;

    protected override async Task OnInitializedAsync(){
        comment = new() { PhotoId = PhotoId };
        comments = (await service.GetCommentsForPhotoAsync(PhotoId)).ToList();
    }

    private async Task HandleValidSubmit()
    {
        comments.Add(await service.AddCommentAsync(comment));
    }
}
```

If you run the application now, you should be able to add a new comment for an existing photo.

In the next lab, we're going to create a WebApi to read and write the comments, then we're going to connect the BlazorComponent to the WebApi by using an `HttpClient` object.

Resources

- https://docs.microsoft.com/en-us/aspnet/core/blazor/components/prerendering-and-integration?view=aspnetcore-6.0&pivots=webassembly
- https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/component-tag-helper?view=aspnetcore-6.0
- https://docs.microsoft.com/en-us/aspnet/core/blazor/forms-validation?view=aspnetcore-6.0#example-form
- https://github.com/Blazored/FluentValidation