Photo Management Challenge solution

API Updates

1. Add isApproved to Photo entity.

```
using System;
using System.Text.Json.Serialization;

namespace API.Entities;

public class Photo
{
    public int Id { get; set; }
    public required string Url { get; set; }
    public string? PublicId { get; set; }
    public bool IsApproved { get; set; }

    // Navigation property
    [JsonIgnore]
    public Member Member { get; set; } = null!;
    public string MemberId { get; set; } = null!;
}
```

2. Update the Seed Users so the initial photo is approved for seeded users

```
    user.Member.Photos.Add(new Photo
    {
        Url = member.ImageUrl!,
        MemberUserId = member.Id,
        IsApproved = true
    });
```

3. Drop Db and add migration

```
dotnet ef database drop
dotnet ef migrations add PhotoApprovalAdded
```

4. Add a Query filter to only return approved photos

```
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<Photo>().HasQueryFilter(x => x.IsApproved);
```

5. Need to Ignore Query filter for the current user (GetMemberAsync) and update the repo.

```
// IMemberRepository
Task<IEnumerable<Photo>> GetPhotosForMemberAsync(string userId, bool isCurrentUser);
```

```
// MemberRepository.cs

    public async Task<IEnumerable<Photo>> GetPhotosForMemberAsync(string userId, bool
isCurrentUser)
    {
        var query = context.Members
            .Where(x => x.UserId == userId)
            .SelectMany(x => x.Photos);

```

```csharp
        if (isCurrentUser) query = query.IgnoreQueryFilters();

        return await query.ToListAsync();
    }
```

```csharp
    // MembersController.cs
    [HttpGet("{id}/photos")]
    public async Task<ActionResult<IReadOnlyList<Photo>>> GetMemberPhotos(string id)
    {
        var isCurrentUser = User.GetMemberId() == id;
        return Ok(await uow.MemberRepository.GetPhotosForMemberAsync(id, isCurrentUser));
    }
```

6. Add a PhotoForApprovalDto with the Photo Id, the Url, the Username and the isApproved status

```csharp
using System;

namespace API.DTOs;

public class PhotoForApprovalDto
{
    public int Id { get; set; }
    public required string Url { get; set; }
    public required string UserId { get; set; }
    public bool IsApproved { get; set; }
}

```

7. Add a PhotoRepository that supports the following methods:
    - GetUnapprovedPhotos
    - GetPhotoById
    - RemovePhoto

```csharp
// IPhotoRepository.cs
using System;
using API.DTOs;
using API.Entities;

namespace API.Interfaces;

public interface IPhotoRepository
{
    Task<IReadOnlyList<PhotoForApprovalDto>> GetUnapprovedPhotos();
    Task<Photo?> GetPhotoById(int id);
    void RemovePhoto(Photo photo);
}

```

```csharp
// PhotoRepository.cs
using System;
using API.DTOs;
using API.Entities;
using API.Interfaces;
using Microsoft.EntityFrameworkCore;

namespace API.Data;

public class PhotoRepository(AppDbContext context) : IPhotoRepository
{
    public async Task<Photo?> GetPhotoById(int id)
    {
        return await context.Photos
```

```csharp
                .IgnoreQueryFilters()
                .SingleOrDefaultAsync(x => x.Id == id);
        }

        public async Task<IReadOnlyList<PhotoForApprovalDto>> GetUnapprovedPhotos()
        {
            return await context.Photos
                .IgnoreQueryFilters()
                .Where(p => p.IsApproved == false)
                .Select(u => new PhotoForApprovalDto
                {
                    Id = u.Id,
                    UserId = u.MemberId,
                    Url = u.Url,
                    IsApproved = u.IsApproved
                }).ToListAsync();
        }

        public void RemovePhoto(Photo photo)
        {
            context.Photos.Remove(photo);
        }
    }

```

```csharp
// IUnitOfWork.cs

namespace API.Interfaces;

public interface IUnitOfWork
{
    IMemberRepository MemberRepository { get; }
    IMessageRepository MessageRepository { get; }
    ILikesRepository LikesRepository { get; }
    IPhotoRepository PhotoRepository {get;}
    Task<bool> Complete();
    bool HasChanges();
}
```

```csharp
// UnitOfWork.cs

using API.Interfaces;
using Microsoft.EntityFrameworkCore;

namespace API.Data;

public class UnitOfWork(AppDbContext context) : IUnitOfWork
{
    private IMemberRepository? _memberRepository;
    private IMessageRepository? _messageRepository;
    private ILikesRepository? _likesRepository;
    private IPhotoRepository? _photoRepository;

    public IMemberRepository MemberRepository =>
        _memberRepository ??= new MemberRepository(context);

    public IMessageRepository MessageRepository =>
        _messageRepository ??= new MessageRepository(context);

    public ILikesRepository LikesRepository =>
        _likesRepository ??= new LikesRepository(context);

    public IPhotoRepository PhotoRepository =>
        _photoRepository ??= new PhotoRepository(context);

    // rest of code omitted
}
```

8. Implement the AdminController GetPhotosForModeration method:

```
using API.Entities;
using API.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers;

public class AdminController(UserManager<AppUser> userManager, IUnitOfWork uow) :
BaseApiController
{
    // omitted code

    [Authorize(Policy = "ModeratePhotoRole")]
    [HttpGet("photos-to-moderate")]
    public async Task<ActionResult<IEnumerable<Photo>>> GetPhotosForModeration()
    {
        return Ok(await uow.PhotoRepository.GetUnapprovedPhotos());
    }
}
```

9. Add a method in the Admin Controller to Approve a photo

```
    [Authorize(Policy = "ModeratePhotoRole")]
    [HttpPost("approve-photo/{photoId}")]
    public async Task<ActionResult> ApprovePhoto(int photoId)
    {
        var photo = await unitOfWork.PhotoRepository.GetPhotoById(photoId);

        if (photo == null) return BadRequest("Could not get photo from db");

        photo.IsApproved = true;

        await unitOfWork.Complete();

        return Ok();
    }

```

10. Add a method in the Admin controller to reject a photo

```
// Inject the IPhotoService:

public class AdminController(UserManager<AppUser> userManager, IUnitOfWork unitOfWork,
    IPhotoService photoService) : BaseApiController
{

    // omitted

    [Authorize(Policy = "ModeratePhotoRole")]
    [HttpPost("reject-photo/{photoId}")]
    public async Task<ActionResult> RejectPhoto(int photoId)
    {
        var photo = await unitOfWork.PhotoRepository.GetPhotoById(photoId);

        if (photo == null) return BadRequest("Could not get photo from db");

        if (photo.PublicId != null)
        {
            var result = await photoService.DeletePhotoAsync(photo.PublicId);

            if (result.Result == "ok")
            {
                unitOfWork.PhotoRepository.RemovePhoto(photo);
            }
        }
        else
        {
            unitOfWork.PhotoRepository.RemovePhoto(photo);
        }

        await unitOfWork.Complete();

        return Ok();
    }

```

11. Remove the logic in the MembersController when adding a photo to automatically set the ImageUrl property for the Member and User if they do not have a main photo (no unapproved photos should be a users main photo).

```
    [HttpPost("add-photo")]
    public async Task<ActionResult<Photo>> AddPhoto(IFormFile file)
    {
        var member = await
unitOfWork.MemberRepository.GetMemberForUpdateAsync(User.GetUserId());

        if (member == null) return BadRequest("Cannot update user");

        var result = await photoService.AddPhotoAsync(file);

        if (result.Error != null) return BadRequest(result.Error.Message);

        var photo = new Photo
        {
            Url = result.SecureUrl.AbsoluteUri,
            PublicId = result.PublicId,
            MemberUserId = User.GetUserId()
        };

        member.Photos.Add(photo);

        if (await unitOfWork.Complete()) return photo;

        return BadRequest("Problem adding photo");
    }
```

12. Add the logic in the Admin controller approve photo method to check to see if the user has the ImageUrl property, if not then set the photo to main when approving. The Photo has a MemberId property we can use to get the MemberToUpdate().

```

    // AdminController
    [Authorize(Policy = "ModeratePhotoRole")]
    [HttpPost("approve-photo/{photoId}")]
    public async Task<ActionResult> ApprovePhoto(int photoId)
    {
        var photo = await uow.PhotoRepository.GetPhotoById(photoId);

        if (photo == null) return BadRequest("Could not get photo from db");

        var member = await uow.MemberRepository.GetMemberForUpdate(photo.MemberId);

        if (member == null) return BadRequest("Could not get member");

        photo.IsApproved = true;

        if (member.ImageUrl == null)
        {
            member.ImageUrl = photo.Url;
            member.User.ImageUrl = photo.Url;
        }

        await uow.Complete();

        return Ok();
    }
```

13. Update the MemberRepository.GetMemberForUpdate() method and ignore the query filter so that they can delete unapproved photos.

```
// MemberRepository.cs

    public async Task<Member?> GetMemberForUpdateAsync(string userId)
    {
        return await context.Members
            .Include(x => x.User)
            .Include(x => x.Photos)
            .IgnoreQueryFilters()
            .SingleOrDefaultAsync(x => x.UserId == userId);
    }
```

14. Test the requests in Postman.
    1. Login as Lisa
    2. Add 2 photos for Lisa
    3. Login as admin
    4. Get the unapproved photos as admin (should see 2 photos)
    5. Get lisa profile as Lisa. Should see both the unapproved photos.
    6. Login as todd
    7. Get lisa profile as Todd. Should not see unapproved photos.
    8. Login as admin
    9. Approve one of lisa's new photos.
    10. Reject one of lisa's new photos
    11. Get Lisa's profile again as todd. Should now see the approved photo.
    12. Register a new user (bob) and then login as Bob
    13. Add 2 new photos for bob - both should be unapproved and neither should be set as the main photo
    14. Approve one of bobs photos as admin - this should now be set as Bob's main photo.

Angular updates

15. Update the member.ts Photo type with the isApproved property and an optional username

```
export type Photo = {
  id: number
  url: string
  publicId?: string
  memberId: string
  isApproved: boolean
}
```

16. Add 3 new methods in the admin service:

1. getPhotosForApproval()
2. approvePhoto()
3. rejectPhoto()

```
import { Component, inject, signal } from '@angular/core';
import { Photo } from '../../../types/member';
import { AdminService } from '../../../core/services/admin-service';

@Component({
  selector: 'app-photo-management',
  imports: [],
  templateUrl: './photo-management.html',
  styleUrl: './photo-management.css'
})
export class PhotoManagement {
  photos = signal<Photo[]>([]);
  private adminService = inject(AdminService);

  ngOnInit(): void {
    this.getPhotosForApproval();
  }

  getPhotosForApproval() {
    this.adminService.getPhotosForApproval().subscribe({
      next: photos => this.photos.set(photos)
    })
  }

  approvePhoto(photoId: number) {
    this.adminService.approvePhoto(photoId).subscribe({
      next: () => this.photos.update(photos => {
        return photos.filter(x => x.id !== photoId)
      })
    })
  }

  rejectPhoto(photoId: number) {
    this.adminService.rejectPhoto(photoId).subscribe({
      next: () => this.photos.update(photos => {
        return photos.filter(x => x.id !== photoId)
      })
    })
  }
}
```

17. Display the photos in the photo-management.component.html and approve/reject buttons underneath the photos to call the appropriate methods when clicked.

```
<div class='grid grid-cols-6 gap-4'>
    @for (photo of photos(); track photo.id) {
        <div>
            <img src='{{photo.url}}' class='img-thumbnail p-1' alt='user image'>

            <div class='flex w-full gap-3'>
                <button class='btn btn-success flex-grow'
(click)='approvePhoto(photo.id)'>Approve</button>
                <button class='btn btn-error flex-grow'
(click)='rejectPhoto(photo.id)'>Reject</button>
            </div>
        </div>
    }
</div>

```

18. In the member-photos.html when a user uploads photos ensure that some text is positioned absolutely on any currently unapproved photos as “awaiting approval” in red. Also reduce the opacity to opacity-30 for any unapproved photos.

```
 @for (photo of photos; track photo.id) {
    <div class="relative">
        <img
            src="{{photo.url}}"
            alt="photo of member"
            class="w-full rounded-lg"
            [class.opacity-30]="!photo.isApproved"
        >
        @if (!photo.isApproved) {
            <div class="absolute bottom-2 badge badge-outline badge-error">Awaiting approval</div>
        }

```

19. Make sure that a user cannot select an unapproved photo as their main photo, but they can still delete unapproved photos in the photo-editor.component.html

```
        @if (accountService.currentUser()?.id === membersService.member()?.id) {
        <div class="absolute top-1 right-1">
            <app-star-button [disabled]="photo.url === membersService.member()?.imageUrl || !photo.isApproved"
                [filled]="photo.url === membersService.member()?.imageUrl" (clickEvent)="setMainPhoto(photo)" />
        </div>
```

20. Test it all in the browser and we are done!
21. Finished! Commit changes to GitHub.