Photo Management Challenge solution

1. Photo.cs

```
using System.ComponentModel.DataAnnotations.Schema;

namespace API.Entities;

[Table("Photos")]
public class Photo
{
    public int Id { get; set; }
    public required string Url { get; set; }
    public bool IsMain { get; set; }
    public string? PublicId { get; set; }
    public bool IsApproved { get; set; } = false;

    // Navigation properties
    public int AppUserId { get; set; }
    public AppUser AppUser { get; set; } = null!;
}
```

2. DataContext:

```
public DbSet<Photo> Photos { get; set; }
```

3. Update the PhotoDto

```
    public class PhotoDto
    {
        public int Id { get; set; }
        public string Url { get; set; }
        public bool IsMain { get; set; }
        public bool IsApproved { get; set; }
    }
```

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

```
            foreach (var user in users)
```

```
            {
                user.Photos.First().IsApproved = true;
                user.UserName = user.UserName.ToLower();
                await userManager.CreateAsync(user, "Pa$$w0rd");
                await userManager.AddToRoleAsync(user, "Member");
            }
```

5. Drop Db and add migration

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

6. Add a Query filter to only return approved photos

```
            builder.Entity<Message>()
                .HasOne(u => u.Sender)
                .WithMany(m => m.MessagesSent)
                .OnDelete(DeleteBehavior.Restrict);

            builder.Entity<Photo>().HasQueryFilter(p => p.IsApproved);

            builder.ApplyUtcDateTimeConverter();
```

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

IUserRepository

```
        Task<MemberDto> GetMemberAsync(string username, bool isCurrentUser);
```

UserRepository:

```
 public async Task<MemberDto> GetMemberAsync(string username, bool
isCurrentUser)
        {
            var query = _context.Users
                .Where(x => x.UserName == username)
                .ProjectTo<MemberDto>(_mapper.ConfigurationProvider)
                .AsQueryable();

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

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

UserController:

```
        [HttpGet("{username}"]
        public async Task<ActionResult<MemberDto>> GetUser(string username)
        {
            var currentUsername = User.GetUsername();
            return await _unitOfWork.UserRepository.GetMemberAsync(username,
                isCurrentUser: currentUsername == username
            );
        }
```

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

```
namespace API;

public class PhotoForApprovalDto
{
    public int Id { get; set; }
    public required string Url { get; set; }
    public string? Username { get; set; } // optional as this matches the
AppUser entity prop
    public bool IsApproved { get; set; }
}
```

9. Add a PhotoRepository that supports the following methods:
    1. GetUnapprovedPhotos
    2. GetPhotoById
    3. RemovePhoto

```
using API.Entities;

namespace API.Interfaces;
```

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

```
using API.Entities;
using API.Interfaces;
using Microsoft.EntityFrameworkCore;

namespace API.Data;

public class PhotoRepository(DataContext context) : IPhotoRepository
{
    public async Task<Photo?> GetPhotoById(int id)
    {
        return await context.Photos
            .IgnoreQueryFilters()
            .SingleOrDefaultAsync(x => x.Id == id);
    }

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

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

```
namespace API.Interfaces;

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

```
using API.Interfaces;

namespace API.Data;

public class UnitOfWork(DataContext context, IUserRepository userRepository,
    ILikesRepository likesRepository, IMessageRepository messageRepository,
    IPhotoRepository photoRepository) : IUnitOfWork
{
    public IUserRepository UserRepository => userRepository;

    public IMessageRepository MessageRepository => messageRepository;

    public ILikesRepository LikesRepository => likesRepository;
    public IPhotoRepository PhotoRepository => photoRepository;

    public async Task<bool> Complete()
    {
        return await context.SaveChangesAsync() > 0;
    }

    public bool HasChanges()
    {
        return context.ChangeTracker.HasChanges();
    }
}
```

Add to application service extensions:

```
services.AddScoped<IPhotoRepository, PhotoRepository>();
```

10. Implement the AdminController GetPhotosForApproval method:

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

namespace API;

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

    [Authorize(Policy = "ModeratePhotoRole")]
    [HttpGet("photos-to-moderate")]
    public async Task<ActionResult> GetPhotosForModeration()
    {
        var photos = await unitOfWork.PhotoRepository.GetUnapprovedPhotos();

        return Ok(photos);
    }
}
```

11. 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();
    }
```

12. 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();
    }
```

13. Remove the logic in the UsersController when adding a photo to automatically set a photo to main if they do not have a main photo (no unapproved photos should be a users main photo).

```
    [HttpPost("add-photo")]
    public async Task<ActionResult<PhotoDto>> AddPhoto(IFormFile file)
    {
        var user = await
unitOfWork.UserRepository.GetUserByUsernameAsync(User.GetUsername());

        if (user == 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
        };

        user.Photos.Add(photo);

        if (await unitOfWork.Complete())
            return CreatedAtAction(nameof(GetUser),
                new {username = user.UserName}, mapper.Map<PhotoDto>(photo));

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

14. Add the logic in the Admin controller approve photo method to check to see if the user has any photos that are set to main, if not then set the photo to main when approving. Will need a method to get a user by the photo Id in the user repository to do this:

```
// IUserRepository.cs

Task<AppUser?> GetUserByPhotoId(int photoId);
```

```
// UserRepository.cs

    public async Task<AppUser?> GetUserByPhotoId(int photoId)
    {
        return await context.Users
            .Include(p => p.Photos)
            .IgnoreQueryFilters()
            .Where(p => p.Photos.Any(p => p.Id == photoId))
            .FirstOrDefaultAsync();
    }
```

Admin Controller Approve Photo method:

```
    [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;

        var user = await unitOfWork.UserRepository.GetUserByPhotoId(photoId);

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

        if (!user.Photos.Any(x => x.IsMain)) photo.IsMain = true;
```

```
        await unitOfWork.Complete();

        return Ok();
    }
```

15. Update the Delete Photo method in the UsersController to use the GetPhotoById method in the PhotoRepository so they have the ability to delete photos that are yet to be approved:

```
    [HttpDelete("delete-photo/{photoId:int}")]
    public async Task<ActionResult> DeletePhoto(int photoId)
    {
        var user = await
unitOfWork.UserRepository.GetUserByUsernameAsync(User.GetUsername());

        if (user == null) return BadRequest("User not found");

        var photo = await unitOfWork.PhotoRepository.GetPhotoById(photoId);

        if (photo == null || photo.IsMain) return BadRequest("This photo cannot
be deleted");

        if (photo.PublicId != null)
        {
            var result = await photoService.DeletePhotoAsync(photo.PublicId);
            if (result.Error != null) return BadRequest(result.Error.Message);
        }

        user.Photos.Remove(photo);

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

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

16. Test the requests in Postman.
    1. Login as Lisa,Todd,Admin and save the 3 tokens to the env
    2. Add 2 photos for Lisa
    3. Get the unapproved photos as admin (should see 2 photos)
    4. Get lisa profile as Lisa. Should see both the unapproved photos.

5. Get lisa profile as Todd. Should not see unapproved photos.
6. Approve one of lisa's new photos.
7. Reject one of lisa's new photos
8. Get Lisa's profile again as todd. Should now see the approved photo.
9. Register a new user (bob) and save token as bob_token
10. Add 2 new photos for bob - both should be unapproved and neither should be set as the main photo
11. Approve one of bobs photos as admin - this should now be set as Bob's main photo.

17. Update the photo.ts with the isApproved property and an optional username

```
export interface Photo {
    id: number;
    url: string;
    isMain: boolean;
    isApproved: boolean;
      username?: string;
}
```

18. Add 3 new methods in the admin service:
    1. getPhotosForApproval()
    2. approvePhoto()
    3. rejectPhoto()

```
  getPhotosForApproval() {
    return this.http.get<Photo[]>(this.baseUrl + 'admin/photos-to-moderate');
  }

  approvePhoto(photoId: number) {
    return this.http.post(this.baseUrl + 'admin/approve-photo/' + photoId, {});
  }

  rejectPhoto(photoId: number) {
    return this.http.post(this.baseUrl + 'admin/reject-photo/' + photoId, {});
  }
```

19. Add the corresponding methods in the photo-management.component.ts:

```
export class PhotoManagementComponent implements OnInit {
  photos: Photo[] = [];
  private adminService = inject(AdminService);

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

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

  approvePhoto(photoId: number) {
    this.adminService.approvePhoto(photoId).subscribe({
      next: () => this.photos.splice(this.photos.findIndex(p => p.id ===
photoId), 1)
    })
  }

  rejectPhoto(photoId: number) {
    this.adminService.rejectPhoto(photoId).subscribe({
      next: () => this.photos.splice(this.photos.findIndex(p => p.id ===
photoId), 1)
    })
  }

}
```

20. Display the photos in the photo-management.component.html along with the username of the user and approve/reject buttons underneath the photos to call the appropriate methods when clicked.

```
<div class='row'>
    @for (photo of photos; track photo.id) {
        <div class='col-sm-2'>
            <h4>{{photo.username}}</h4>
            <img src='{{photo.url}}' class='img-thumbnail p-1'
alt='{{photo.username}}'>
```

```
            <div class='text-center'>
                <button class='btn btn-sm btn-success me-1'
(click)='approvePhoto(photo.id)'>Approve</button>
                <button class='btn btn-sm btn-danger'
(click)='rejectPhoto(photo.id)'>Reject</button>
            </div>
        </div>
    }
</div>
```

21. Add some style so the photos take up 175px height/width

```
img.img-thumbnail {
    height: 175px;
    min-width: 175px !important;
    margin-bottom: 2px;
  }
```

22. In the photo-editor 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 0.2 for any unapproved photos.

Photo-editor.component.html

```
<div class="row">
    <div class="col-2 mb-1 img-wrapper" *ngFor="let photo of member.photos">
        <img src="{{photo.url}}" alt="{{photo.url}}" class="img-thumbnail p-1"
            [class.not-approved]="!photo.isApproved"
        >

        <div class="text-center img-text" *ngIf="!photo.isApproved">
            <span class="text-danger">Awaiting approval</span>
        </div>
```

Photo-editor.component.css

```
.not-approved {
```

```
    opacity: 0.2;
}

.img-wrapper {
    position: relative
}

.img-text {
    position: absolute;
    left: 0;
    right: 0;
    margin-left: auto;
    margin-right: auto;
    bottom: 30%;
    text-align: center;
    word-wrap: break-word;
    white-space: normal;
    width: 100%;
    padding: 0 10px;
    box-sizing: border-box;
}
```

23. 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

```
            <button
                [disabled]="photo.isMain || !photo.isApproved"
                (click)="setMainPhoto(photo)"
                [ngClass]='photo.isMain ? "btn-success active" : "btn-outline-
success"'
                class="btn btn-sm"
            >Main</button>
```

24. Test it all in the browser and we are done!

#datingapp-final/section16