Photo Management Challenge solution

1. Photo.cs

```
namespace API.Entities
{
    [Table("Photos")]
    public class Photo
    {
        public int Id { get; set; }
        public string Url { get; set; }
        public bool IsMain { get; set; }
        public bool IsApproved { get; set; }
        public string PublicId { get; set; }
        public AppUser AppUser { get; set; }
        public int AppUserId { get; set; }
    }
}
```

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}", Name = "GetUser")]
        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.DTOs
{
    public class PhotoForApprovalDto
```

```
    {
        public int Id { get; set; }
        public string Url { get; set; }
        public string Username { get; set; }
        public bool IsApproved { get; set; }
    }
}
```

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

```
using System.Collections.Generic;
using System.Threading.Tasks;
using API.DTOs;
using API.Entities;

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

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using API.DTOs;
using API.Entities;
using API.Interfaces;
```

```
using Microsoft.EntityFrameworkCore;

namespace API.Data
{
    public class PhotoRepository : IPhotoRepository
    {
        private readonly DataContext _context;
        public PhotoRepository(DataContext context)
        {
            _context = context;
        }

        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 async Task<Photo> GetPhotoById(int id)
        {
            return await _context.Photos
                .IgnoreQueryFilters()
                .SingleOrDefaultAsync(x => x.Id == id);
        }

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

```
}
```

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

```
  public class UnitOfWork : IUnitOfWork
    {
        private readonly IMapper _mapper;
        private readonly DataContext _context;
        public UnitOfWork(DataContext context, IMapper mapper)
        {
            _context = context;
            _mapper = mapper;
        }

        public IUserRepository UserRepository => new UserRepository(_context,
_mapper);

        public IMessageRepository MessageRepository => new
MessageRepository(_context, _mapper);

        public ILikesRepository LikesRepository => new
LikesRepository(_context);

        public IPhotoRepository PhotoRepository => new
PhotoRepository(_context);
```

10. Implement the AdminController GetPhotosForApproval method:

```
    public class AdminController : BaseApiController
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly IUnitOfWork _unitOfWork;
        public AdminController(UserManager<AppUser> userManager, IUnitOfWork
unitOfWork)
        {
            _unitOfWork = unitOfWork;
            _userManager = userManager;
        }


          // 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);

            photo.IsApproved = true;

            await _unitOfWork.Complete();

            return Ok();
        }
```

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

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

            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());

            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 CreatedAtRoute("GetUser", new { username =
user.UserName }, _mapper.Map<PhotoDto>(photo));
            }

            return BadRequest("Problem addding 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:

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

UserRepository:

```
        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:

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

            if (photo == null) return NotFound("Could not find photo");

            photo.IsApproved = true;

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

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

            await _unitOfWork.Complete();
```

```
            return Ok();
        }
```

15. 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.

16. 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;
}
```

17. 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, {});
  }
```

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

```
export class PhotoManagementComponent implements OnInit {
  photos: Photo[];

  constructor(private adminService: AdminService) { }

  ngOnInit(): void {
  }

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

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

  rejectPhoto(photoId) {
    this.adminService.rejectPhoto(photoId).subscribe(() => {
      this.photos.splice(this.photos.findIndex(p => p.id === photoId), 1);
    })
  }
```

19. 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'>
    <div class='col-sm-2' *ngFor='let photo of photos'>
        <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
mr-1' (click)='approvePhoto(photo.id)'>Approve</button>
            <button class='btn btn-sm btn-
danger' (click)='rejectPhoto(photo.id)'>Reject</button>
        </div>
    </div>
</div>
```

20. Add some style so the photos take up 150px height/width

```
img.img-thumbnail {
    height: 150px;
    min-width: 150px !important;
    margin-bottom: 2px;
  }
```

21. 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.ts

```
<div class="row">
    <div class="col-2 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;
    bottom: 30%;
}
```

22. Make sure that a user cannot select an unapproved photo as their main photo, but they can still delete unapproved photos.

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

```
                    class="btn btn-sm"
                >Main</button>
```

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