Projektni zadatak

Student: Stefan Erić

Broj indeksa: 4/21

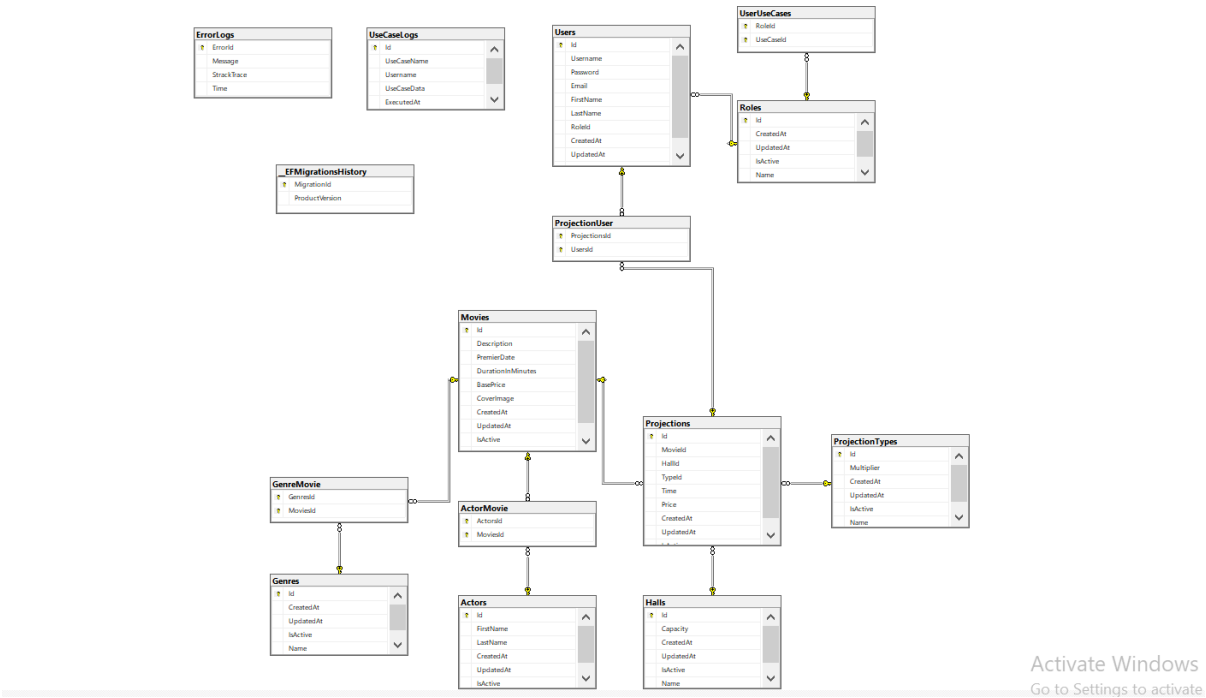
Predmet: Web programiranje ASP

Predmetni profesor: Luka Lukić

Sadrzaj

Uopsteno

Dizajn baze

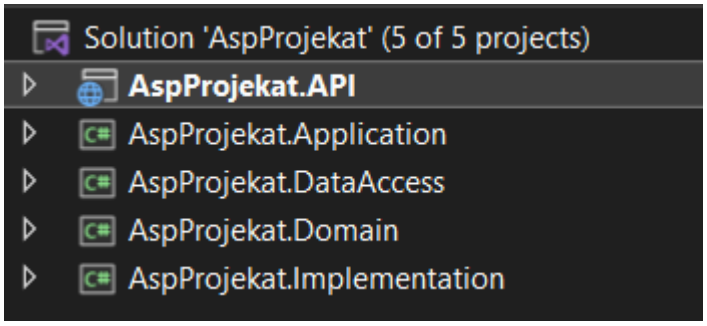


Opis projektnog rada

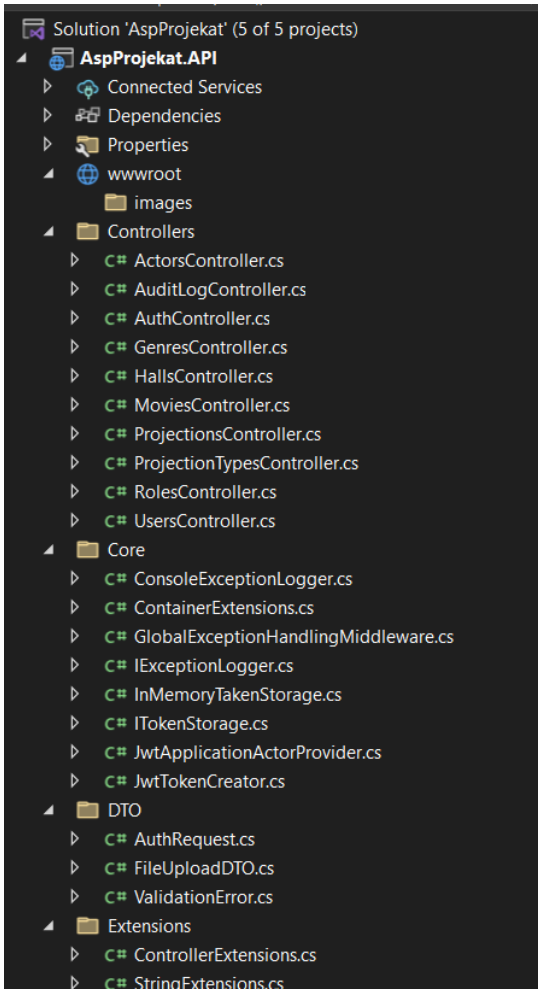
Ovaj projektni rad predstavlja aplikaciju koja simulira bioskop, odnosno bavi se pregledom filmova i dozvoljava rezervaciju njihovih projekcija ako je to moguće. Dalje funkcionalnosti aplikacije biće opisane redom, kroz stavke za ocenivanje projekta.

Arhitektura

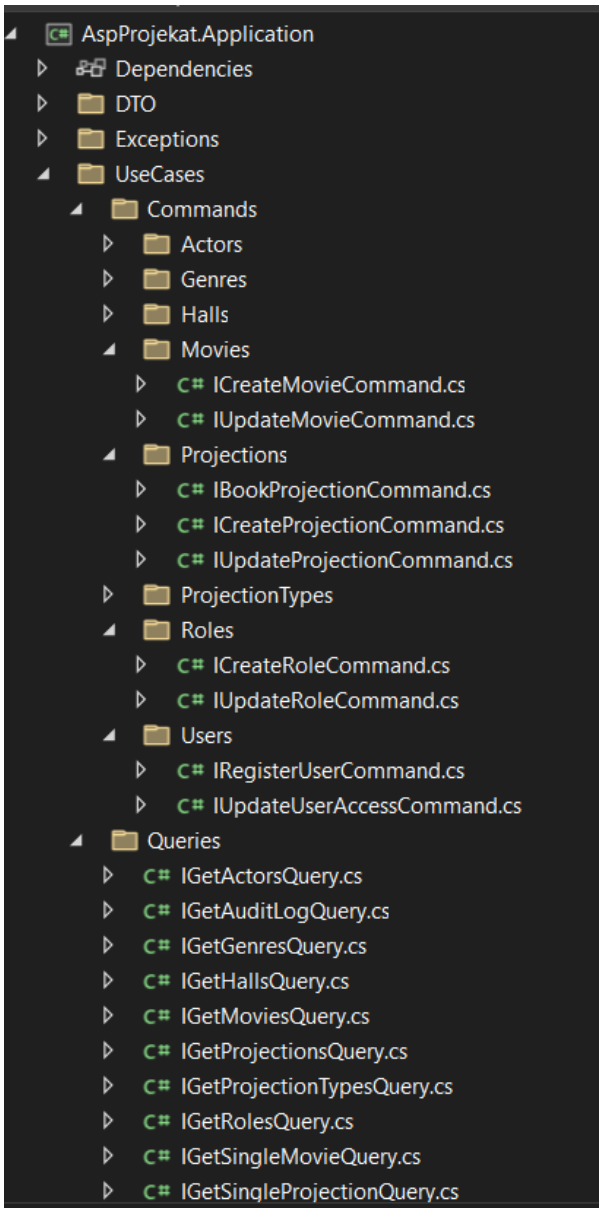
Projekat je ukupno razdvojen na 5 slojeva



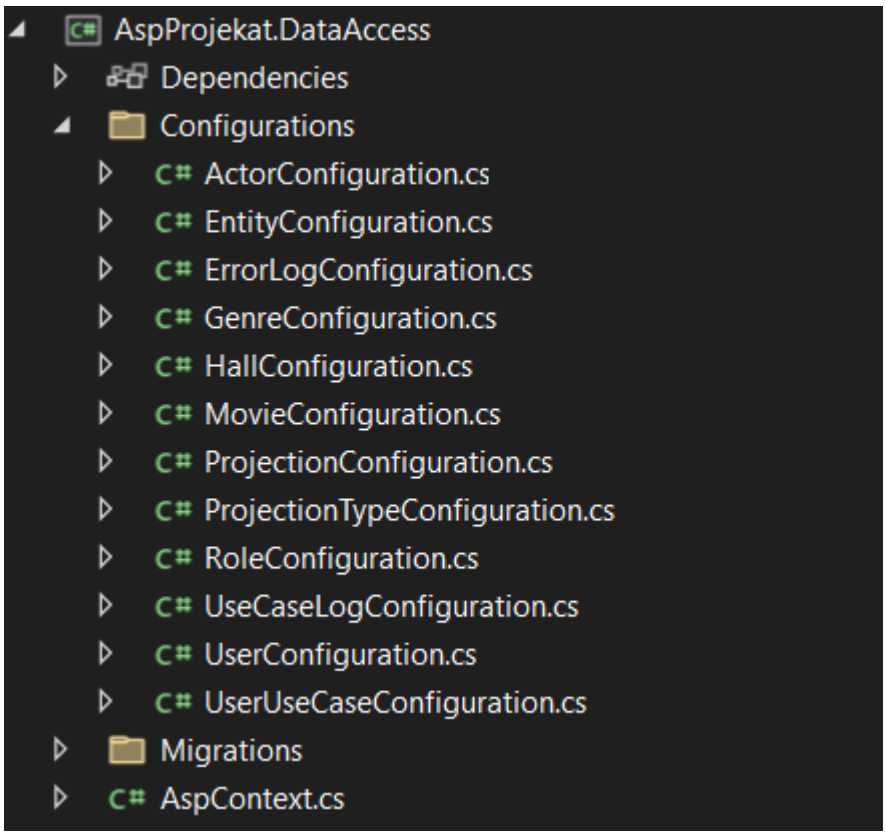
1. Api sloj (za komunikaciju sa klijentskom aplikacijom) - u njemu su definisani kontroleri



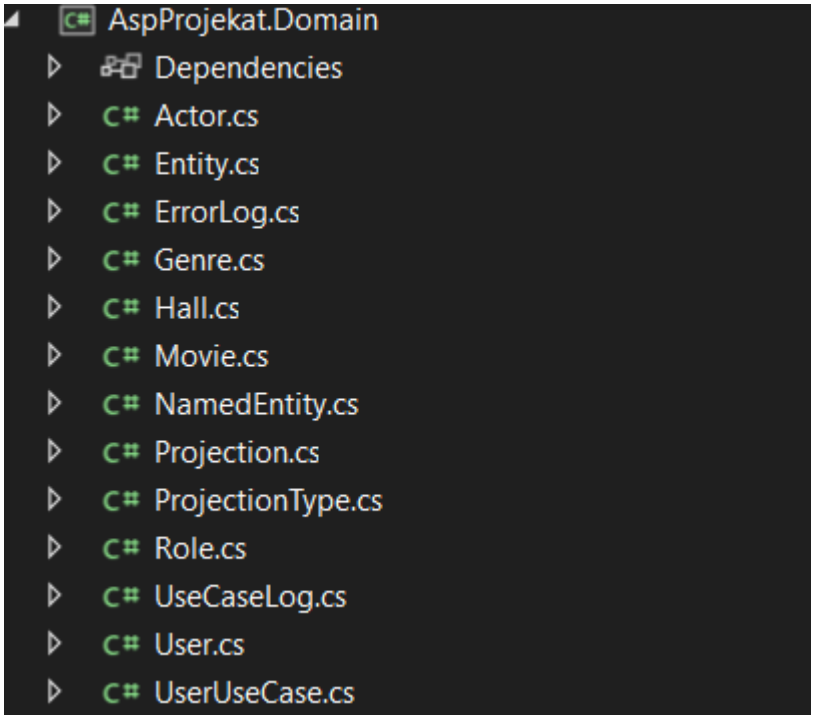
2. Aplikacioni sloj - (sloj poslovne logike) u njemu su definisani Intfejsi za primenu logike kao i DTO



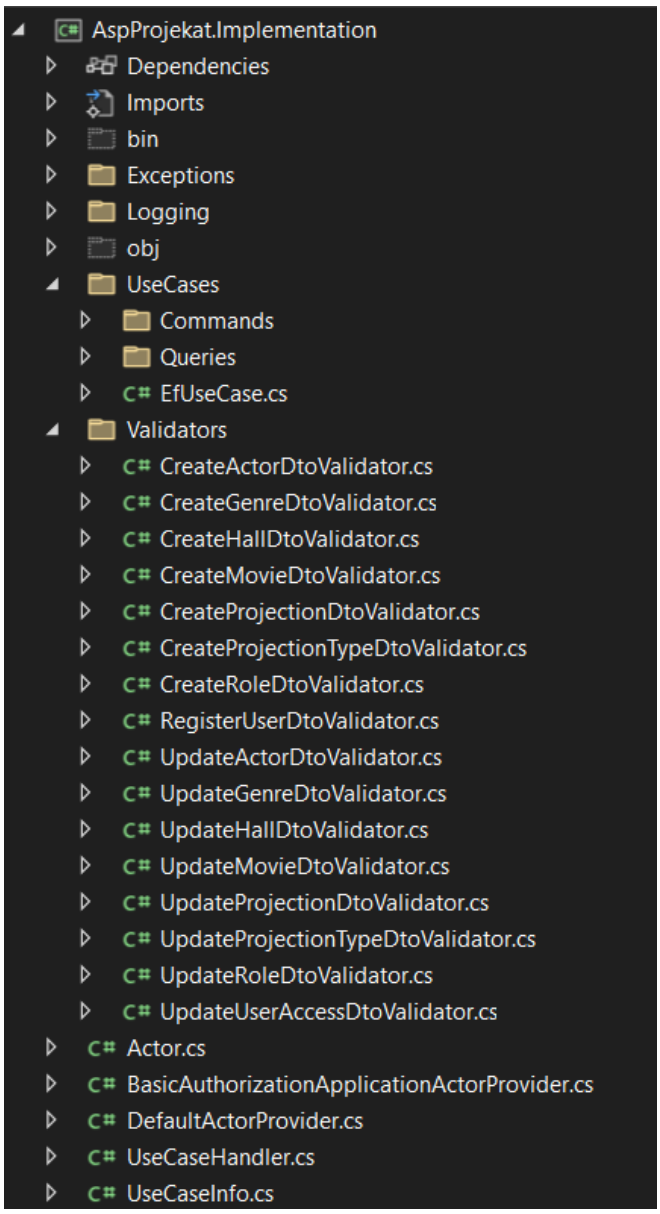
3. DataAccess sloj – konfiguracioni fajlovi za migraciju baze



4. Domenski sloj u kome se nalaze entiteti baze podataka



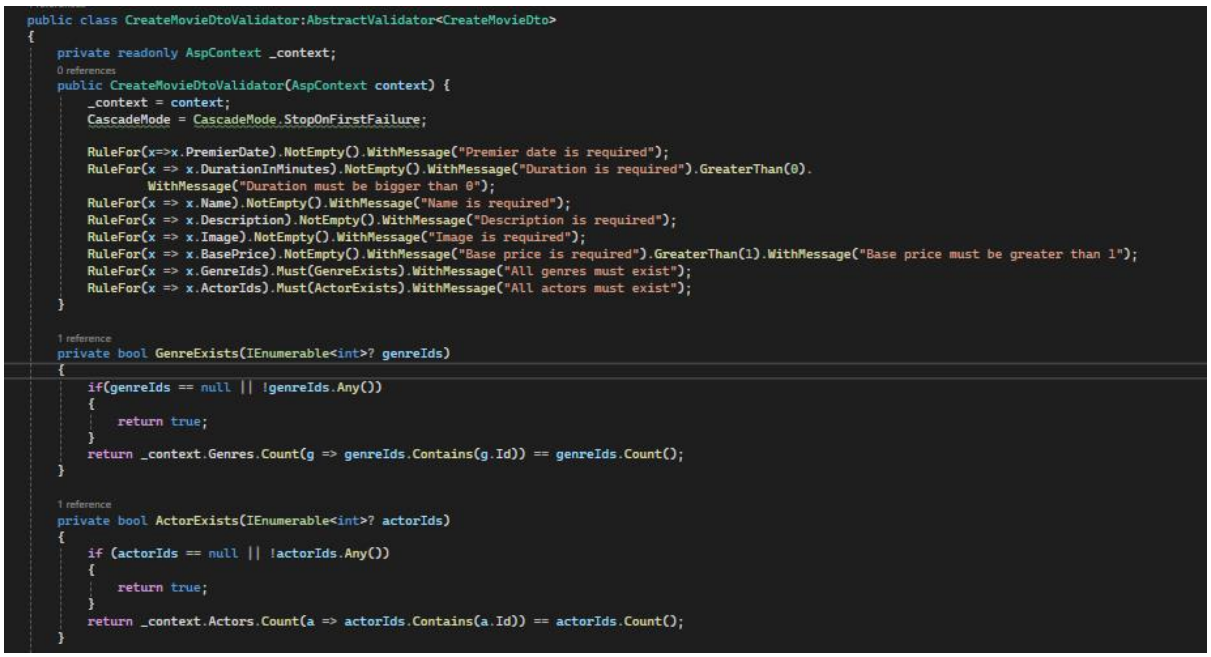
5. Implementacioni sloj u kome se primenjuje i validira biznis logika



Detaljna serverska validacija endpoint-a

Koriscena je FluentValidation i validira se svaki POST, PUT I DELETE endpoint. Proverava se logičnost kao i mogućnost upisa ili izmene podataka poslatih preko DTO-a

Primer: validator za kreiranje filma

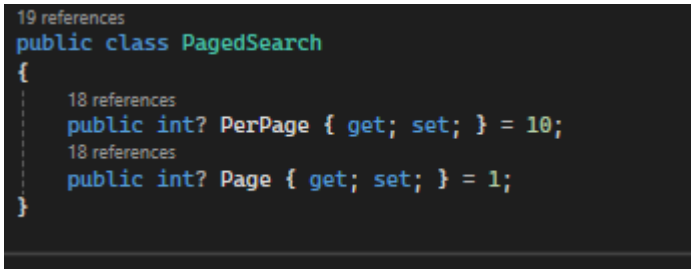


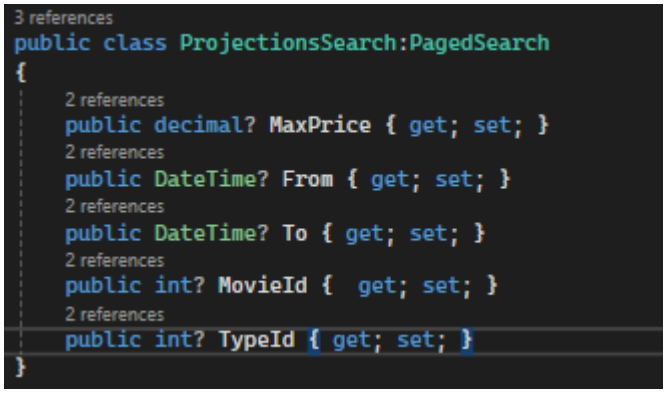
```
public class CreateMovieDtoValidator:AbstractValidator<CreateMovieDto>
{
    private readonly AspContext _context;
    0 references
    public CreateMovieDtoValidator(AspContext context) {
        _context = context;
        CascadeMode = CascadeMode.StopOnFirstFailure;

        RuleFor(x=>x.PremierDate).NotEmpty().WithMessage("Premier date is required");
        RuleFor(x => x.DurationInMinutes).NotEmpty().WithMessage("Duration is required").GreaterThan(0).
               WithMessage("Duration must be bigger than 0");
        RuleFor(x => x.Name).NotEmpty().WithMessage("Name is required");
        RuleFor(x => x.Description).NotEmpty().WithMessage("Description is required");
        RuleFor(x => x.Image).NotEmpty().WithMessage("Image is required");
        RuleFor(x => x.BasePrice).NotEmpty().WithMessage("Base price is required").GreaterThan(1).WithMessage("Base price must be greater than 1");
        RuleFor(x => x.GenreIds).Must(GenreExists).WithMessage("All genres must exist");
        RuleFor(x => x.ActorIds).Must(ActorExists).WithMessage("All actors must exist");
    }

    1 reference
    private bool GenreExists(IEnumerable<int>? genreIds)
    {
        if(genreIds == null || !genreIds.Any())
        {
            return true;
        }
        return _context.Genres.Count(g => genreIds.Contains(g.Id)) == genreIds.Count();
    }

    1 reference
    private bool ActorExists(IEnumerable<int>? actorIds)
    {
        if (actorIds == null || !actorIds.Any())
        {
            return true;
        }
        return _context.Actors.Count(a => actorIds.Contains(a.Id)) == actorIds.Count();
    }
```

Paginacija i pretraga

Za svaki GET zahtev omogućena je paginacija i pretraga koja je i proširena od standardne gde je to potrebno

Primer: Paginacija i pretraga za Projkecije



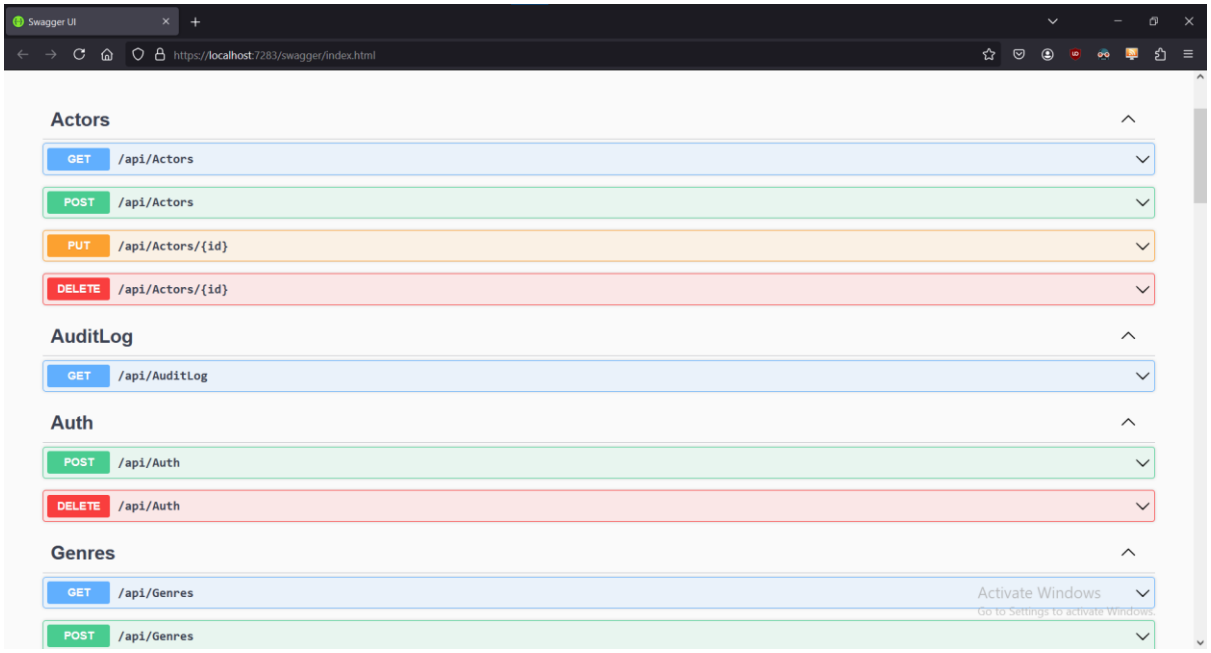


```
1 reference
public class EfGetProjectionsQuery : EfUseCase, IGetProjectionsQuery
{
    0 references
    public EfGetProjectionsQuery(AspContext context) : base(context) { }

    2 references
    public int Id => 2;
    4 references
    public string Name => "Search projections";
    2 references
    public PagedResponse<ProjectionDto> Execute(ProjectionsSearch search)
    {
        var query = Context.Projections.AsQueryable();

        if (search.MaxPrice.HasValue)
        {
            query = query.Where(x => (decimal)x.Price < search.MaxPrice);
        }
        if (search.MovieId.HasValue)
        {
            query = query.Where(x => x.MovieId == search.MovieId);
        }
        if (search.TypeId.HasValue)
        {
            query = query.Where(x => x.TypeId == search.TypeId);
        }
        if (search.To.HasValue)
        {
            query = query.Where(x => x.Time < search.To);
        }
        if(search.From.HasValue)
        {
            query = query.Where(x => x.Time > search.From);
        }
        query = query.Where(x => x.IsActive);

        int totalCount = query.Count();

        int perPage = search.PerPage.HasValue ? (int)Math.Abs((double)search.PerPage) : 10;
        int page = search.Page.HasValue ? (int)Math.Abs((double)search.Page) : 1;

        int skip = perPage * (page - 1);

        query = query.Skip(skip).Take(perPage);
```

```
return new PagedResponse<ProjectionDto>
{
    CurrentPage = page,
    Data = query.Select(x => new ProjectionDto
    {
        Id = x.Id,
        Price = (decimal)x.Price,
        Time = x.Time,
        Hall = new HallDto
        {
            Id = x.Hall.Id,
            Capacity = x.Hall.Capacity,
            Name = x.Hall.Name
        },
        Movie = new MovieDto
        {
            Id = x.Movie.Id,
            Name = x.Movie.Name,
            Description = x.Movie.Description,
            DurationInMinutes = x.Movie.DurationInMinutes,
            PremierDate = x.Movie.PremierDate,
            CoverImage = x.Movie.CoverImage
        },
        ProjectionType = new ProjectionTypeDto
        {
            Id = x.Type.Id,
            Name = x.Type.Name,
            Multiplier = x.Type.Multiplier
        },
        Users = x.Users.Select(y => new UserDto
        {
            Id = y.Id,
            FirstName = y.FirstName,
            LastName = y.LastName,
            Email = y.Email,
            Username = y.Username
        }).ToList()
    }).ToList(),
    TotalCount = totalCount,
    PerPage = perPage
};
```

Swagger specifikacija

Odrađena je za svaki endpoint – vidi se pokretanjem projekta



Dizajn baze

Dizajn baze već je dat slikom gore i odrađen je code first pristupom pokretanjem migracije

Autorizacija upotrebom JWT-a

Za autorizaciju koristi se JWT token, čija je logika kreiranja smeštena u API delu projekta u CORE folderu

Autorizacija na nivou slučaja korišćena

Autorizacija je odrađena tako da se svaki UseCase vezuje za rolu koja se vezuje za Usera, tako da bi korisnici iste grupe imali iste mogućnost, neautorizovani korisnici pregled, autorizovani pregled i rezervaciju projekcije, admin korisnici kreiranje i menjanje entiteta i owner ili super admin kreiranje rola i use case-eva

Slanje Email-a

Nije implementirano

Ispravno vraćanje statusnih kodova

Vraćanje statusnih kodova implementirano je tako da se za uspešnu pretragu ili update vraća 200, za uspešno kreiranje 201, za pokušaj korišćenja neautroizovanog use-case-a 401 i za Enitite koji nisu pronađeni 404 – entity not found

Primer Actor kontrolera

```
1 reference
public class ActorsController:ControllerBase
{
    private IApplicationActor _actor;
    private UseCaseHandler _handler;
    private AspContext _context;

    0 references
    public ActorsController(IApplicationActor actor, UseCaseHandler handler, AspContext context)
    {
        _actor = actor;
        _handler = handler;
        _context = context;
    }
    [HttpGet]
    0 references
    public IActionResult Get([FromQuery] PagedSearch search, [FromServices] IGetActorsQuery query) => Ok(_handler.HandleQuery(query, search));

    [Authorize]
    [HttpPost]
    0 references
    public IActionResult Post([FromBody] CreateActorDto dto, [FromServices] ICreateActorCommand cmd)
    {
        _handler.HandleCommand(cmd, dto);
        return StatusCode(201);
    }
    [Authorize]
    [HttpPut("{id}")]
    0 references
    public IActionResult Put(int id, [FromBody] UpdateActorDto dto, [FromServices] IUpdateActorCommand cmd)
    {
        dto.Id = id;
        _handler.HandleCommand(cmd, dto);

        return NoContent();
    }
    [Authorize]
    [HttpDelete("{id}")]

    0 references
    public IActionResult Delete(int id, [FromServices] IUpdateActorCommand cmd)
    {
        UpdateActorDto dto = new UpdateActorDto { Id = id, IsActive = false };
        _handler.HandleCommand(cmd, dto);
        return NoContent();
    }

}
```

AuditLogs

Napravljena je tabela u koju se upisuju svi use case logovi, a od podataka se vidi

```
public class AuditLogDto
{
    0 references
    public int Id { get; set; }
    1 reference
    public string UseCaseName { get; set; }
    1 reference
    public string Username { get; set; }

    1 reference
    public string UseCaseData { get; set; }

    1 reference
    public DateTime ExecutedAt { get; set; }
}
```

I omogućena je ptretraga po traženim parametrima, što se vidi i na swager konfiguraciji

```
public class AuditLogSearch : PagedSearch
{
    2 references
    public string? UseCaseName { get; set; }
    2 references
    public string? Username { get; set; }
    3 references
    public DateTime? From { get; set; }
    1 reference
    public DateTime? To {  get; set; }

}
```

```
2 references
public int Id => 4;

4 references
public string Name => "Search audit log";

2 references
public PagedResponse<AuditLogDto> Execute(AuditLogSearch search)
{
    var query = Context.UseCaseLogs.AsQueryable();

    if (!string.IsNullOrEmpty(search.Username))
    {
        query = query.Where(x => x.Username.Contains(search.Username));
    }

    if (!string.IsNullOrEmpty(search.UseCaseName))
    {
        query = query.Where(x => x.UseCaseName.Contains(search.UseCaseName));
    }

    if (search.From.HasValue)
    {
        query = query.Where(x => x.ExecutedAt > search.From);
    }

    if (search.To.HasValue)
    {
        query = query.Where(x => x.ExecutedAt < search.From);
    }

    int totalCount = query.Count();

    int perPage = search.PerPage.HasValue ? (int)Math.Abs((double)search.PerPage) : 10;
    int page = search.Page.HasValue ? (int)Math.Abs((double)search.Page) : 1;

    int skip = perPage * (page - 1);

    query = query.Skip(skip).Take(perPage);

    return new PagedResponse<AuditLogDto>
    {
        CurrentPage = page,
        Data = query.Select(x => new AuditLogDto
        {
            Username = x.Username,
            UseCaseData = x.UseCaseData,
            ExecutedAt = x.ExecutedAt,
            UseCaseName = x.UseCaseName
        }).ToList(),
        PerPage = perPage,
        TotalCount = totalCount,
    };
}
```

Upotreba automappera

Nije implementirana

Upload fajla za bar jedan entitet

Implementirana je za entitet Movie, gde se kroz form data šalje njegova cover image i smešta se u API delu projekta u folder wwwroot pa / images

```
2 references
public int Id => 4;

4 references
public string Name => "Create movie";

private CreateMovieDtoValidator _validator;

0 references
public EfCreateMovieCommand(AspContext context, CreateMovieDtoValidator validator) : base(context) { _validator = validator; }

2 references
public void Execute(CreateMovieDto data)
{
    _validator.ValidateAndThrow(data);

    var extension = Path.GetExtension(data.Image.FileName);
    var filename = Guid.NewGuid().ToString() + extension;
    var savePath = Path.Combine("wwwroot", "images", filename);

    Directory.CreateDirectory(Path.GetDirectoryName(savePath));

    using (var fs = new FileStream(savePath, FileMode.Create))
    {
        data.Image.CopyTo(fs);
    }

    var Movie = new AspProjekat.Domain.Movie
    {
        Name = data.Name,
        Description = data.Description,
        PremierDate = data.PremierDate,
        BasePrice = data.BasePrice,
        CoverImage = "/images/" + filename,
        DurationInMinutes = data.DurationInMinutes
    };
    if(data.ActorIds!=null && data.ActorIds.Any())
    {
        Movie.Actors = Context.Actors.Where(a => data.ActorIds.Contains(a.Id)).ToList();
    }
    if(data.GenreIds!=null && data.GenreIds.Any())
    {
        Movie.Genres = Context.Genres.Where(g => data.GenreIds.Contains(g.Id)).ToList();

    }
    Context.Movies.Add(Movie);
    Context.SaveChanges();
}
```