Contents

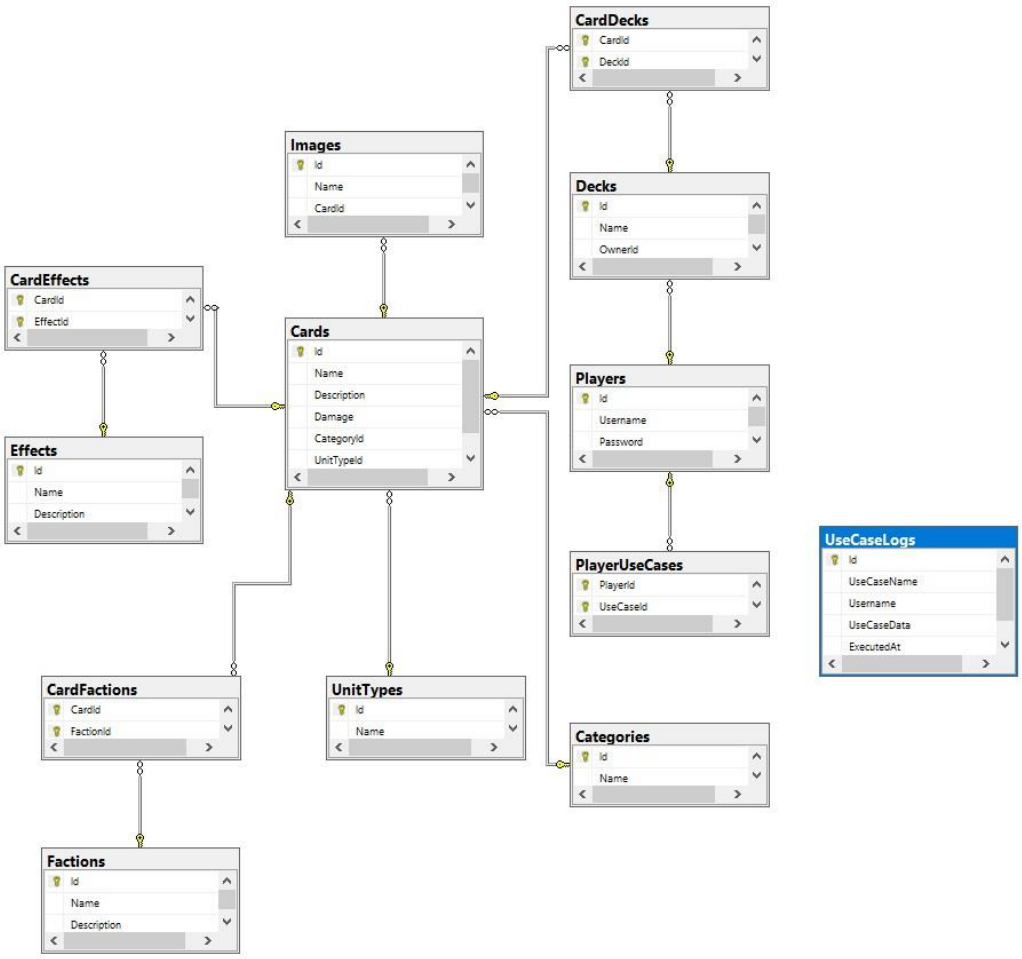
Opis Sistema:

Gwent je kartaška igra u kojoj postoje karte sa svojim:

- Fakcijama(svaka karta se može koristiti u više fakcija, ali postoje i specijalne karte koje pripadaju jednoj fakciji)
- Kategora(Npr Weather ili Vremenska karta koja upravlja napadom jedinica u odjeđenom tipu jedinice)
- Tip jedinice(postoje pešadija ili prva borbena linija, ranged jedinice ili druga borbena linija i siege jedinice ili treća borbena linija)
- Sistem je tako napravljen da postoji jedan korisnik koji može da upravlja celim sistemom dodavanjem novih entiteta
- Korisnici koji samo mogu da prave svoje deckove(špilove) i u njih da dodaju karte kao i da iz svojih špilova brišu karte(isključivo iz svojih što je regulisano validacijom i proverama)

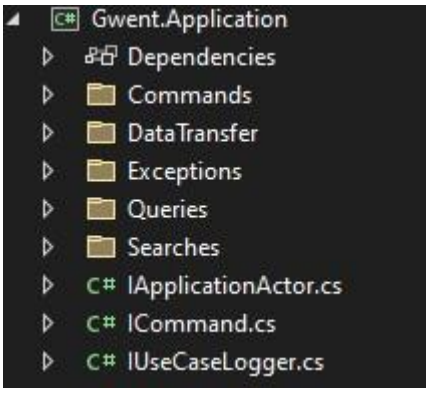
Sistem je osmišljen tako da predstavlja katalog za karte i da korisnici mogu da upravljaju svojim špilovima.

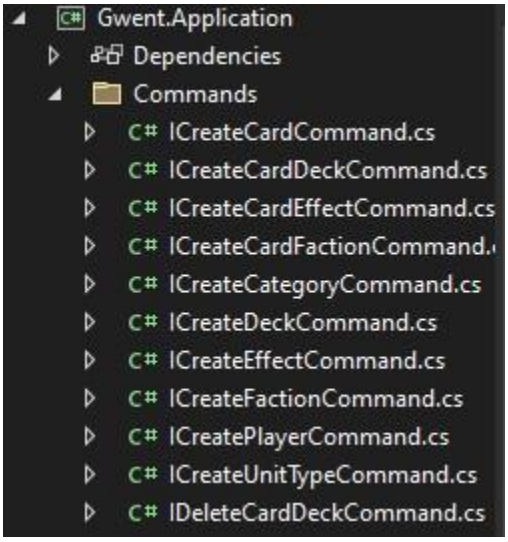
Dizajn baze:

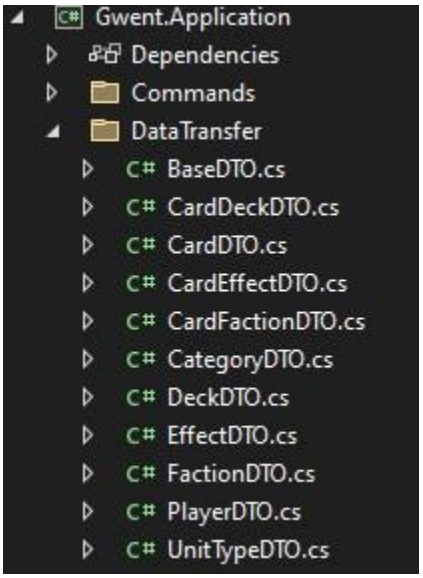


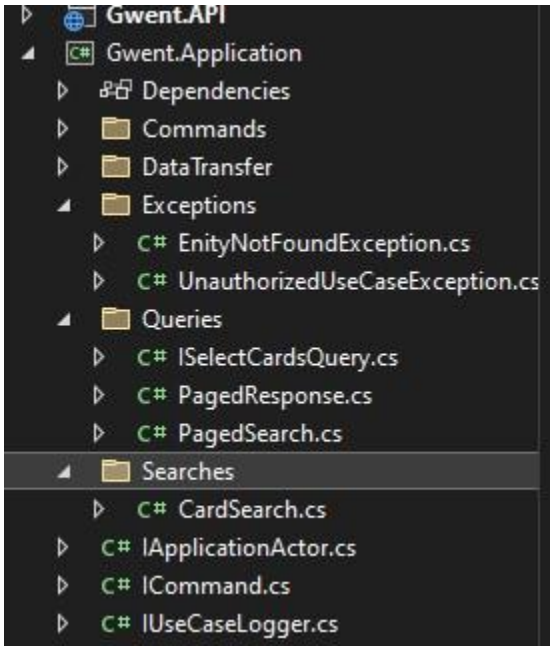
Arhitektura sistema:

Sloj aplikacije:

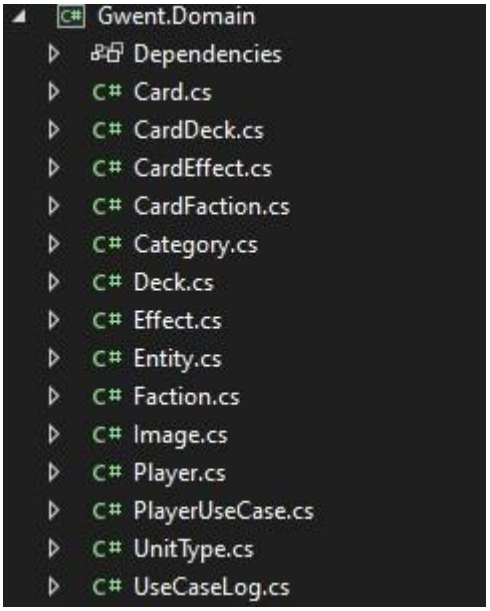






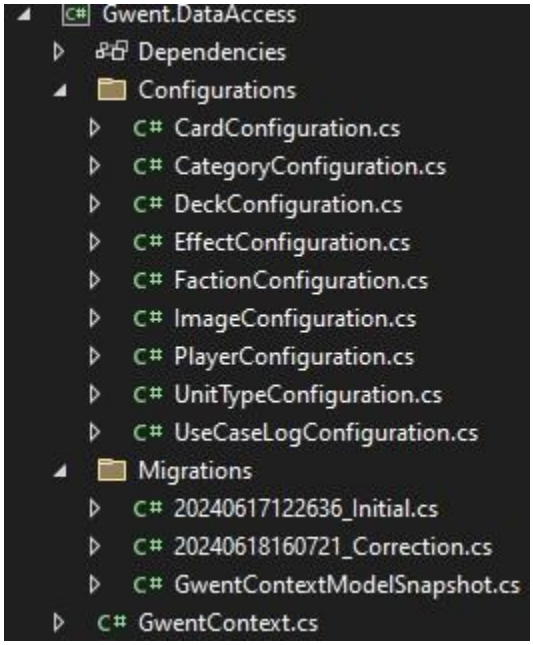


Domenski sloj:

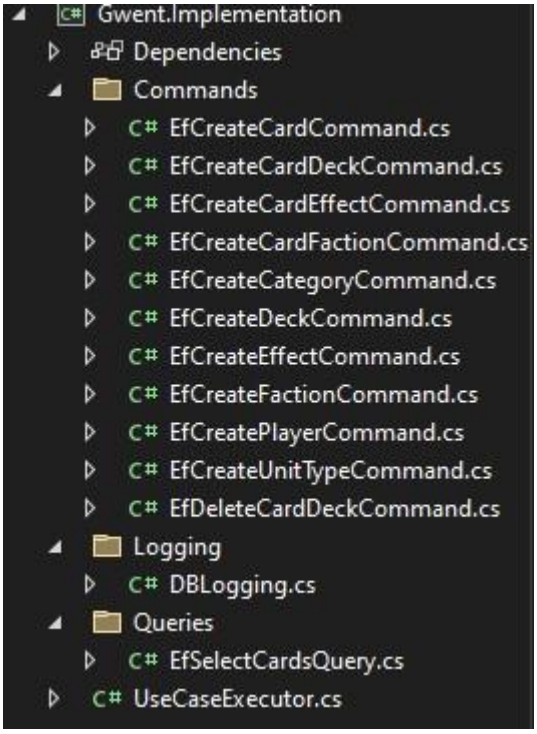


DataAccess sloj:

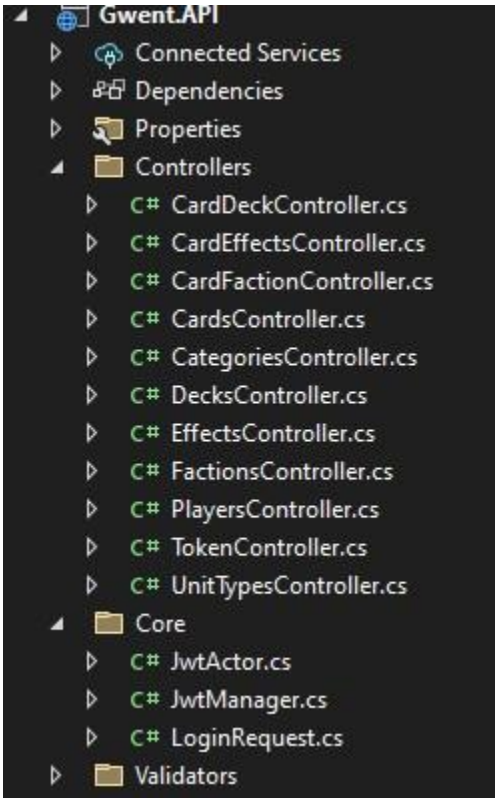
Baza je napravljena code-first pristupom

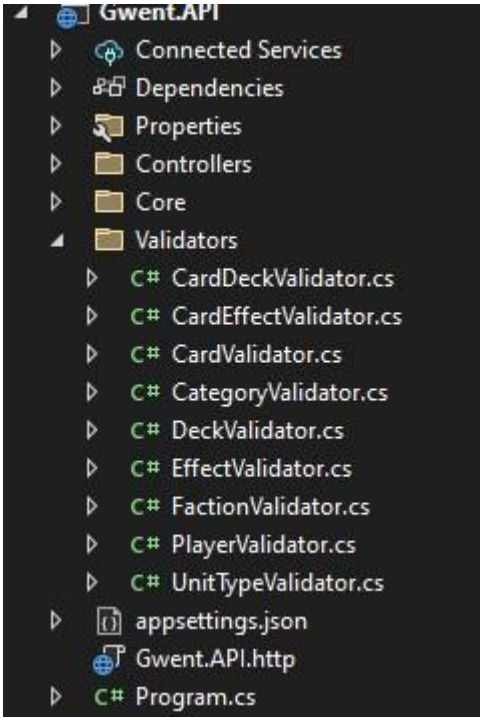


Sloj implementacije:


Gwent.Implementation
Dependencies
Commands
EfCreateCardCommand.cs
EfCreateCardDeckCommand.cs
EfCreateCardEffectCommand.cs
EfCreateCardFactionCommand.cs
EfCreateCategoryCommand.cs
EfCreateDeckCommand.cs
EfCreateEffectCommand.cs
EfCreateFactionCommand.cs
EfCreatePlayerCommand.cs
EfCreateUnitTypeCommand.cs
EfDeleteCardDeckCommand.cs
Logging
DBLogging.cs
Queries
EfSelectCardsQuery.cs
UseCaseExecutor.cs

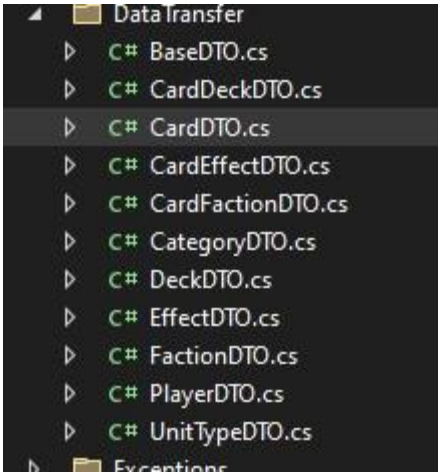
API sloj:


Gwent.API
Connected Services
Dependencies
Properties
Controllers
CardDeckController.cs
CardEffectsController.cs
CardFactionController.cs
CardsController.cs
CategoriesController.cs
DecksController.cs
EffectsController.cs
FactionsController.cs
PlayersController.cs
TokenController.cs
UnitTypesController.cs
Core
JwtActor.cs
JwtManager.cs
LoginRequest.cs
Validators



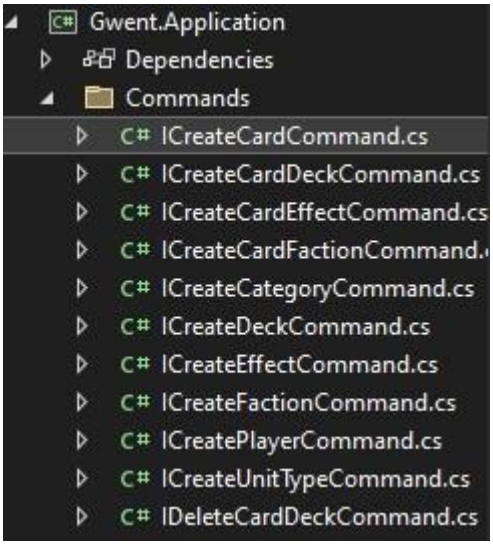
Univerzalni tok izvršavanja svake komande:

Za odgovarajuću komandu se pravi DTO objekat:




```
public class CardDTO : BaseDTO
{
    public string Name { get;set; }
    public string Description { get;set; }
    public int Damage { get; set; }
    public int UnitTypeId{ get;set; }
    public int CategoryId{ get;set; }
}
```

Kreira se interfejs u aplikativnom sloju koji se izvodi iz interfejsa ICommand koji implementira interfejs IUseCase

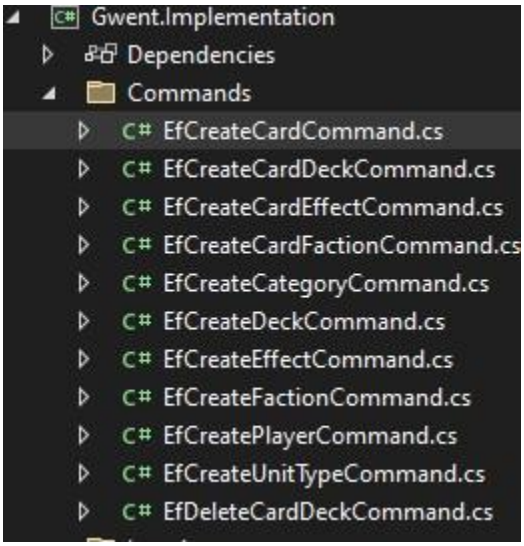



```
namespace Gwent.Application.Commands
{
    public interface ICreateCardCommand : ICommand<CardDTO>
    {
    }
}
```

Ova dva interfejsa kasnije definisu nas Query ili komandu time što komanda implementira interfejs iCommand i time se obezbeđuje poštovanje hijerarhije da je svaka komanda ili query use case koji korisnik izvršava

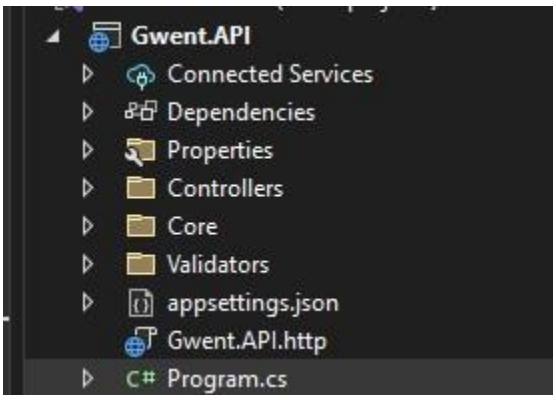
```
namespace Gwent.Application
{
    public interface ICommand<TRequest> : IUseCase
    {
        void Execute(TRequest request);
    }

    public interface IQuery<TSearch, TResult> : IUseCase
    {
        TResult Execute(TSearch search);
    }

    public interface IUseCase
    {
        int Id { get; }
        string Name { get; }
    }
}
```

Sloj implementacije sluzi da definiše šta svaka komanda treba da radi, ali poštujući implementaciju i pravila odgovarajućeg interfejsa.




```
public class EfCreateCardCommand : ICreateCardCommand
{
    private readonly GwentContext context;

    public EfCreateCardCommand(GwentContext context)
    {
        this.context = context;
    }

    public int Id => 5;

    public string Name => "Create card using Ef";

    public void Execute(CardDTO request)
    {
        Card c = new Card
        {
            Name = request.Name,
            Description = request.Description,
            Damage = request.Damage,
            CategoryId = request.CategoryId,
            UnitTypeId = request.UnitTypeId,
        };

        context.Cards.Add(c);
        context.SaveChanges();
    }
}
```

U API sloju pre svega moramo definisati u dependency injection kontejneru šta ćemo u kom slučaju koristiti da bi aplikacija mogla da radi.



Ovim smo rekli da kada se zatraži ovaj interfejs, izvršiće se klasa pored:

```
builder.Services.AddTransient<ICreateCardCommand, EfCreateCardCommand>();
```

```
builder.Services.AddTransient<GwentContext>();
builder.Services.AddTransient<UseCaseExecutor>();
builder.Services.AddTransient<IUseCaseLogger, DBLogging>();
builder.Services.AddTransient<ICreateFactionCommand, EfCreateFactionCommand>();
builder.Services.AddTransient<ICreateUnitTypeCommand, EfCreateUnitTypeCommand>();
builder.Services.AddTransient<ICreateCategoryCommand, EfCreateCategoryCommand>();
builder.Services.AddTransient<ICreateEffectCommand, EfCreateEffectCommand>();
builder.Services.AddTransient<ICreateCardCommand, EfCreateCardCommand>();
builder.Services.AddTransient<ICreateCardEffectCommand, EfCreateCardEffectCommand>();
builder.Services.AddTransient<ICreateCardFactionCommand, EfCreateCardFactionCommand>();
builder.Services.AddTransient<ICreatePlayerCommand, EfCreatePlayerCommand>();
builder.Services.AddTransient<ISelectCardsQuery, EfSelectCardsQuery>();
builder.Services.AddTransient<ICreateDeckCommand, EfCreateDeckCommand>();
builder.Services.AddTransient<ICreateCardDeckCommand, EfCreateCardDeckCommand>();
builder.Services.AddTransient<IDeleteCardDeckCommand, EfDeleteCardDeckCommand>();
builder.Services.AddTransient<JwtManager>();
builder.Services.AddHttpContextAccessor();
builder.Services.AddTransient<IApplicationActor>(x =>
{
    var accessor = x.GetService<IHttpContextAccessor>();

    if(user.FindFirst("ActorData") == null)
```

Kreiranjem kontrolera dobijamo endpoint koji će služiti za dodavanje nove Karte u bazu, a od provera sadrži:

- Validator koji proverava primljen objekat u i proverava podatke da li su popunjeni i da li su u skladu sa maksimalnim vrednostima u poređenju sa bazom
- Sve je u try i catch bloku koji hvata unauthorized exception specifično, u suprotnom vraća status kod 500
- Provera da li postoje entiteti sa zadatim id-jevima, preciznije da li postoji uopšte kategorija sa tim id-jem i to je vezano za svaki entitet, naravno u skladu sa svojim podacima i tabelama sa kojima je povezana

```
public IActionResult Post([FromBody] CardDTO dto, [FromServices] ICreateCardCommand command)
{

    try
    {
        if (dto == null)
        {
            return UnprocessableEntity(new { error = "No data to process!" });
        }

        CardValidator validator = new CardValidator();
        var result = validator.Validate(dto);

        if (!result.IsValid)
        {
            return UnprocessableEntity(result.Errors.Select(x => new
            {
                Errors = x.ErrorMessage,
                Property = x.PropertyName
            }));
        }

        GwentContext context = new GwentContext();

        Category check_category = context.Categories.Find(dto.CategoryId);
        UnitType check_unit_type = context.UnitTypes.Find(dto.UnitTypeId);

        if (check_category == null)
        {
            return NotFound(new { error = "Category with the provided ID doesn't exist!" });
        }

        if (check_unit_type == null)
        {
            return NotFound(new { error = "Unit type with the provided ID doesn't exist!" });
        }

        executor.ExecuteCommand(command, dto);
        return Ok(new { message = "Successful entry." });
    }
    catch (Exception ex)
    {
        if (ex is UnauthorizedUseCaseException)
        {
            return Unauthorized();
        }

        return StatusCode(500);
    }
```

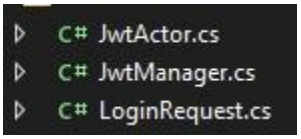
U svakom kontroleru se očekuje instanca klase UseCaseExecutor koja validira korisnika koji je izvučen iz JWT tokena i proverava use case id tekuće komande i da li se on nalazi u nizu usecase-ova koje korisnik sme da izvrši

- Preciznije samo prvi korisnik može da izvrši sve komande(1-11), dok svi ostali korisnici smeju da izvrše samo 9,10 i 11 što su dohvatanje svih karata sa pretragom i paginacijom, dodavanje svog novog špila i dodavanje karte u svoj špil(sve je obezveđeno validacijom)

Validacija JWT tokenom:

- 

- endpoint za dohvatanje tokena koji prima objekat sa poljima Username i Password koji proverava korisnika u bazi, ako postoji vraća token sa njegovim podacima, ako ne postoji vraća statusni kod 401 ○




```
public class JwtActor : IApplicationActor
{
    public int Id { get; set; }

    public string Identity { get; set; }

    public IEnumerable<int> AllowedUseCases{ get; set; }

}
```

Metod koji pravi token na osnovu korisnika iz baze Kada se zatraži objekat koji implementira interfejs IapplicationActor pokreće se ceo proces dohvatanja i validacije jwt tokena

```
public JwtManager(GwentContext context)
{
    _context = context;
}

public string MakeToken(string username, string password)
{
    var user = _context.Players.Include(p => p.UseCases)
        .FirstOrDefault(x => x.Username == username && x.Password == password);

    if (user == null)
    {
        return null;
    }

    var actor = new JwtActor
    {
        Id = user.Id,
        AllowedUseCases = user.UseCases.Select(x => x.UseCaseId),
        Identity = user.Username
    };

    var issuer = "asp_api";
    var secretKey = "this is my custom Secret key for authentication";
    var claims = new List<Claim> // Jti : "",
    {
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString(), ClaimValueTypes.String, issuer),
        new Claim(JwtRegisteredClaimNames.Iss, "asp_api", ClaimValueTypes.String, issuer),
        new Claim(JwtRegisteredClaimNames.Iat, DateTimeOffset.UtcNow.ToUnixTimeSeconds().ToString(), ClaimValueTypes.Integer64, issuer),
        new Claim("UserId", actor.Id.ToString(), ClaimValueTypes.String, issuer),
        new Claim("ActorData", JsonConvert.SerializeObject(actor), ClaimValueTypes.String, issuer)
    };

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));

    var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var now = DateTime.UtcNow;
    var token = new JwtSecurityToken(
        issuer: issuer,
        audience: "Any",
        claims: claims,
        notBefore: now,
        expires: now.AddSeconds(300),
        signingCredentials: credentials);

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

```
builder.Services.AddTransient<IApplicationActor>(x =>
{
    var accessor = x.GetService<IHttpContextAccessor>();

    var user = accessor.HttpContext.User;

    if(user.FindFirst("ActorData") == null)
    {
        throw new InvalidOperationException("Actor data doesn't exist in the token.");
    }

    var actorString = user.FindFirst("ActorData").Value;

    var actor = JsonConvert.DeserializeObject<JwtActor>(actorString);

    return actor;
});

builder.Services.AddAuthentication(options =>
{
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultSignInScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(cfg =>
{
    cfg.RequireHttpsMetadata = false;
    cfg.SaveToken = true;
    cfg.TokenValidationParameters = new TokenValidationParameters
    {
        ValidIssuer = "asp_api",
        ValidateIssuer = true,
        ValidAudience = "Any",
        ValidateAudience = true,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("this is my custom Secret key for authentication")),
        ValidateIssuerSigningKey = true,
        ValidateLifetime = true,
        ClockSkew = TimeSpan.Zero
    };
});

builder.Services.AddControllers();
```

Svaki kontroler sadrži ova polja koja se inicijalizuju konstruktorom istih

```
[Authorize]
public class CardDeckController : ControllerBase
{
    public IApplicationActor actor;
    public UseCaseExecutor executor;

    public CardDeckController(IApplicationActor actor, UseCaseExecutor executor)
    {
        this.actor = actor;
        this.executor = executor;
    }
```

Pretraga i paginacija

http://localhost:5191/api/cards?name=a

Na ovom endpointu dobija se rezultat pretrage karata sa opcijom paginacije

Granulacija privilegija na osnovu svakog korisnika i logovanje izvršavanja usecase-a u tabelu UseCaseLogs

Klasa UseCaseExecutor služi za obradu svakog usecase-a i takođe u sebi sadrži proveru da li korisnik koji je pokrenuo usecase može da izvrši isti time što proverava njegov niz int-ova koji se izvlači iz baze da li se u

njemu nalazi Id tekućeg usecase-a, ako se ne nalazi catch blok na svakom endpointu vraća statusni kod 401

```
public class UseCaseExecutor
{
    private readonly IApplicationActor actor;
    private readonly IUseCaseLogger logger;

    public UseCaseExecutor(IApplicationActor actor, IUseCaseLogger logger)
    {
        this.actor = actor;
        this.logger = logger;
    }

    public TResult ExecuteQuery<TSearch, TResult>(IQuery<TSearch, TResult> query, TSearch search)
    {
        logger.Log(query, actor, search);

        if (!actor.AllowedUseCases.Contains(query.Id))
        {
            throw new UnauthorizedUseCaseException(query, actor);
        }

        return query.Execute(search);
    }

    public void ExecuteCommand<TRequest>(
        ICommand<TRequest> command,
        TRequest request)
    {
        logger.Log(command, actor, request);
        if (!actor.AllowedUseCases.Contains(command.Id))
        {
            throw new UnauthorizedUseCaseException(command, actor);
        }

        command.Execute(request);
    }
}
```

Takođe očekuje se i objekat koji implementira interfejs IuseCaseLogger, a u ovom slučaju to radi klasa DBLogger koja loguje pokušaj izvršavanja u bazu podataka

```
public class DBLogging : IUseCaseLogger
{
    private readonly GwentContext context;

    public DBLogging(GwentContext context)
    {
        this.context = context;
    }


    public void Log(IUseCase useCase, IApplicationActor actor, object data)
    {
        UseCaseLog log = new UseCaseLog
        {
            UseCaseName = useCase.Name,
            Username = actor.Identity,
            UseCaseData = JsonConvert.SerializeObject(data),
            ExecutedAt = DateTime.Now,
        };

        context.UseCaseLogs.Add(log);
        context.SaveChanges();
    }
}
```

```
builder.Services.AddTransient<IUseCaseLogger, DBLogging>();
```

U program.cs klasi u apiju, dodavanjem ove linije u dependency injection kontejner naglasili smo da se koristi DBLogging kada god se zatraži IuseCaseLogger, a pošto se u svakom kontroleru poziva kroz konstruktor , sve je obezbeđeno za logovanje u bazu.

Prikaz tabele UseCaseLogs ❾

	Id	UseCaseName	Username	UseCaseData	ExecutedAt
1	1	Create effect using Ef	User1	{"Name":"Proba","Description":"Proba","Id":0}	2024-06-19 13:13:50.2070891
2	2	Browse cards with EF	User1	{"Name":"ge","PerPage":10,"Page":1}	2024-06-19 14:47:43.6110237
3	3	Browse cards with EF	User1	{"Name":"g","PerPage":10,"Page":1}	2024-06-19 14:47:53.9238515
4	4	Browse cards with EF	User1	{"Name":"a","PerPage":10,"Page":1}	2024-06-19 14:47:59.0547557
5	5	Create a deck using EF	User1	{"Name":"Deck1","OwnerId":1,"Id":0}	2024-06-19 16:26:47.8353171
6	6	Create a deck using EF	User1	{"Name":"Deck2","OwnerId":1,"Id":0}	2024-06-19 16:28:24.6783601
7	7	Create a deck using EF	User1	{"Name":"Deck3","OwnerId":1,"Id":0}	2024-06-19 16:31:28.4865619
8	8	Create a deck using EF	User1	{"Name":"Deck3","OwnerId":1,"Id":0}	2024-06-19 16:31:46.3448029
9	9	Create a deck using EF	User1	{"Name":"Deck3","OwnerId":1,"Id":0}	2024-06-19 16:32:10.1398965
10	10	Create a deck using EF	User1	{"Name":"Deck3","OwnerId":1,"Id":0}	2024-06-19 16:32:27.0933364
11	11	Create a deck using EF	User1	{"Name":"Deck3","OwnerId":1,"Id":0}	2024-06-19 16:42:44.2868150
12	12	Create a deck using EF	User1	{"Name":"Deck3","OwnerId":1,"Id":0}	2024-06-19 16:42:47.1287988
13	13	Create a deck using EF	User1	{"Name":"Deck3","OwnerId":1,"Id":0}	2024-06-19 16:43:41.0564130
14	14	Create a deck using EF	User1	{"Name":"Deck3","OwnerId":1,"Id":0}	2024-06-19 16:44:06.2331780
15	15	Create a deck using EF	User1	{"Name":"Deck3","OwnerId":1,"Id":0}	2024-06-19 16:45:37.9497819
16	16	Create a deck using EF	User1	{"Name":"Deck3","OwnerId":1,"Id":0}	2024-06-19 16:53:20.5522054
17	17	Create a deck using EF	User1	{"Name":"Deck4","OwnerId":1,"Id":0}	2024-06-19 16:55:52.8034389
18	18	Create a deck using EF	User1	{"Name":"Deck5","OwnerId":1,"Id":0}	2024-06-19 16:55:55.9834030
19	19	Create a deck using EF	User1	{"Name":"Deck6","OwnerId":1,"Id":0}	2024-06-19 16:55:58.8548486
20	20	Add a card to a deck	User1	{"CardId":1,"DeckId":1}	2024-06-19 18:16:43.7487681
21	21	Add a card to a deck	User1	{"CardId":1,"DeckId":1}	2024-06-19 18:18:49.5997519
22	22	Delete a card from a ...	User1	{"CardId":1,"DeckId":1}	2024-06-19 18:20:18.1697390
23	23	Add a card to a deck	User1	{"CardId":3,"DeckId":1}	2024-06-19 18:23:19.0232389
24	24	Add a card to a deck	User1	{"CardId":4,"DeckId":1}	2024-06-19 18:23:22.3361996
25	25	Add a card to a deck	User1	{"CardId":5,"DeckId":1}	2024-06-19 18:23:27.1710955
26	26	Create a deck using EF	User2	{"Name":"Deck2","OwnerId":2,"Id":0}	2024-06-19 18:28:24.1403501
27	27	Create a deck using EF	User2	{"Name":"Deck no1 user 2","OwnerId":2,"Id...	2024-06-19 18:30:04.6172335
28	28	Create a deck using EF	User2	{"Name":"Deck no1 user 2","OwnerId":2,"Id...	2024-06-19 18:32:44.5753170
29	29	Create a deck using EF	User1	{"Name":"Deck2","OwnerId":1,"Id":0}	2024-06-19 18:33:24.4740221
30	30	Create a deck using EF	User2	{"Name":"Deck1","OwnerId":2,"Id":0}	2024-06-19 18:34:24.0092229
31	31	Browse cards with EF	User2	{"Name":"a","PerPage":10,"Page":1}	2024-06-19 20:14:44.8420866