**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра ВТ**

# ОТЧЕТ

**по лабораторной работе №3**
**по дисциплине «Программирование в среде .NET»**
**Тема: РАЗРАБОТКА СЛОЯ ДОСТУПА К ДАННЫМ ПРИЛОЖЕНИЯ**

Студент гр. 6306 _____ Жукаускайте А.В.

Преподаватель _____ Пешехонов К.А.

Санкт-Петербург
2020

## Цель работы

Реализовать слой доступа к данным приложения.

## Задание

Написать слой доступа к данным: Entity Framework Code First + MS SQL Server Developer Edition

## Код программы

Примеры кода программы даны для сущности «автомобиль», другие сущности («депо» и «водитель») реализованы похожим образом.

1) Контекст для работы с базой данных

```
public partial class CinemaContext : DbContext
{
    public CinemaContext() { }

    public CinemaContext(DbContextOptions<CinemaContext> options) :
base(options)  { }

    public virtual DbSet<Cinema> Cinema { get; set; }
    public virtual DbSet<Screening> Screening { get; set; }
    public virtual DbSet<Movie> Movie { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)  { }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Cinema>(entity =>
            {
                entity.Property(c => c.Id).UseIdentityColumn().Metadata
                     .SetBeforeSaveBehavior(PropertySaveBehavior.Ignore);
                entity.Property(c => c.Name).IsRequired();
                entity.Property(c => c.Address).IsRequired();
            });

        modelBuilder.Entity<Movie>(entity =>
            {
                entity.Property(m => m.Id).UseIdentityColumn().Metadata
                    .SetBeforeSaveBehavior(PropertySaveBehavior.Ignore);
                entity.Property(m => m.Title).IsRequired();
                entity.Property(m => m.Director).IsRequired();
                entity.Property(m => m.DateOfPremiere).IsRequired();
                entity.Property((m => m.Age)).IsRequired();
            });

        modelBuilder.Entity<Screening>(entity =>
            {
                entity.Property(s=>s.Id).UseIdentityColumn().Metadata
                  .SetBeforeSaveBehavior(PropertySaveBehavior.Ignore);
                entity.Property(s => s.Date).IsRequired();
                entity.Property(s => s.Time).IsRequired();
                entity.HasOne(s => s.Cinema)
```

```
                    .WithMany(c => c.Screening)
                    .HasForeignKey(s => s.CinemaId)
                    .HasConstraintName("FK_Screening_Cinema");
                entity.HasOne(s => s.Movie)
                    .WithMany(m => m.Screenings).HasForeignKey(s=>s.MovieId)
                    .HasConstraintName("FK_Screening_Movie");
            });

        this.OnModelCreatingPartial(modelBuilder);
    }

    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}
```

2) Пример сущности (Screening)

```
public class Screening
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    public string Time { get; set; }
    public string Date { get; set; }
    public int? CinemaId { get; set; }
    public int? MovieId { get; set; }
    public virtual Movie Movie { get; set; }
    public virtual Cinema Cinema { get; set; }
}
```

3) Пример интерфейса с декларацией методов для доступа к данным (IScreeningDataAccess)

```
public interface IScreeningDataAccess
{
    Task<Screening> InsertAsync(ScreeningUpdateModel screening);
    Task<IEnumerable<Screening>> GetAsync();
    Task<Screening> GetAsync(IScreeningIdentity screeningId);
    Task<Screening> UpdateAsync(ScreeningUpdateModel screening);
    Task<Screening> GetByAsync(IScreeningContainer screening);

}
```

4) Пример класса с реализацией методов для доступа к данным (ScreeningDataAccess)

```
public class ScreeningDataAccess : IScreeningDataAccess
{
    private CinemaContext Context { get; }
    private IMapper Mapper { get; }

    public ScreeningDataAccess(CinemaContext context, IMapper mapper)
    {
        this.Context = context;
        Mapper = mapper;
    }
```

```
    public async Task<Screening> InsertAsync(ScreeningUpdateModel screening)
    {
        var result = await
this.Context.AddAsync(this.Mapper.Map<DataAccess.Entities.Screening>(screening));
        await this.Context.SaveChangesAsync();
        return this.Mapper.Map<Screening>(result.Entity);
    }

    public async Task<IEnumerable<Screening>> GetAsync()
    {
        return this.Mapper.Map<IEnumerable<Screening>>(await
this.Context.Screening.Include(x => x.Cinema).Include(x=>x.Movie).ToListAsync());
    }

    public async Task<Screening> GetAsync(IScreeningIdentity screeningId)
    {

        var result = await this.Get(screeningId);
        return this.Mapper.Map<Screening>(result);
    }

    private async Task<Cinemas.DataAccess.Entities.Screening>
Get(IScreeningIdentity screeningId)
    {
        if (screeningId == null)
            throw new ArgumentNullException(nameof(screeningId));
        return await this.Context.Screening.Include(x =>
x.Cinema).Include(x=>x.Movie).FirstOrDefaultAsync(x => x.Id == screeningId.Id);

    }

    public async Task<Screening> UpdateAsync(ScreeningUpdateModel screening)
    {
        var existing = await this.Get(screening);
        var result = this.Mapper.Map(screening, existing);
        this.Context.Update(result);
        await this.Context.SaveChangesAsync();
        return this.Mapper.Map<Screening>(result);
    }

    public async Task<Screening> GetByAsync(IScreeningContainer screening)
    {
        return screening.ScreeningId.HasValue
            ? this.Mapper.Map<Screening>(await this.Context.Screening.FirstOrDefaultAsync(x
=> x.Id == screening.ScreeningId)) : null;
    }
}
```

**Выводы:**

В процессе выполнения лабораторной работы был реализован доступа к данным приложения в среде .NET. Были получены навыки по разработке организации доступа приложения к внешней базе данных с помощью EntityFrameworkCore и MS SQL Server.