

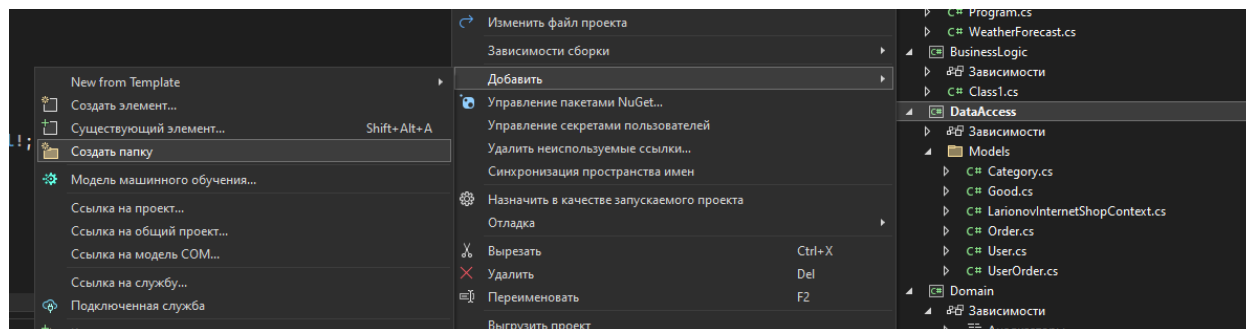
# Реконструирование в Entity Framework Core.

## Формирование архитектуры серверного приложения.

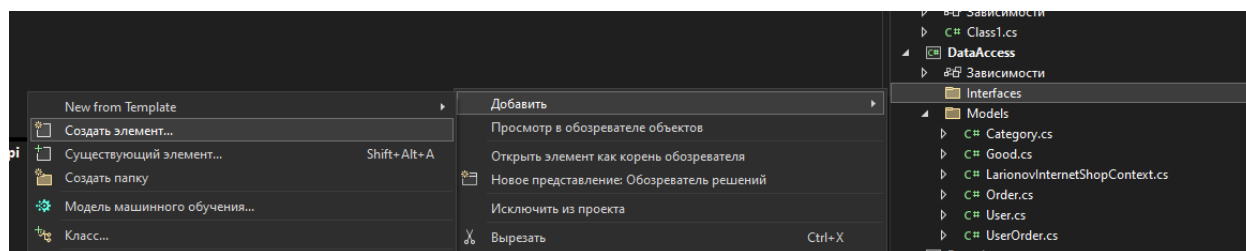
### (Часть 2)

Применение паттерна Repository. Реализация абстрактного класса RepositoryBase.

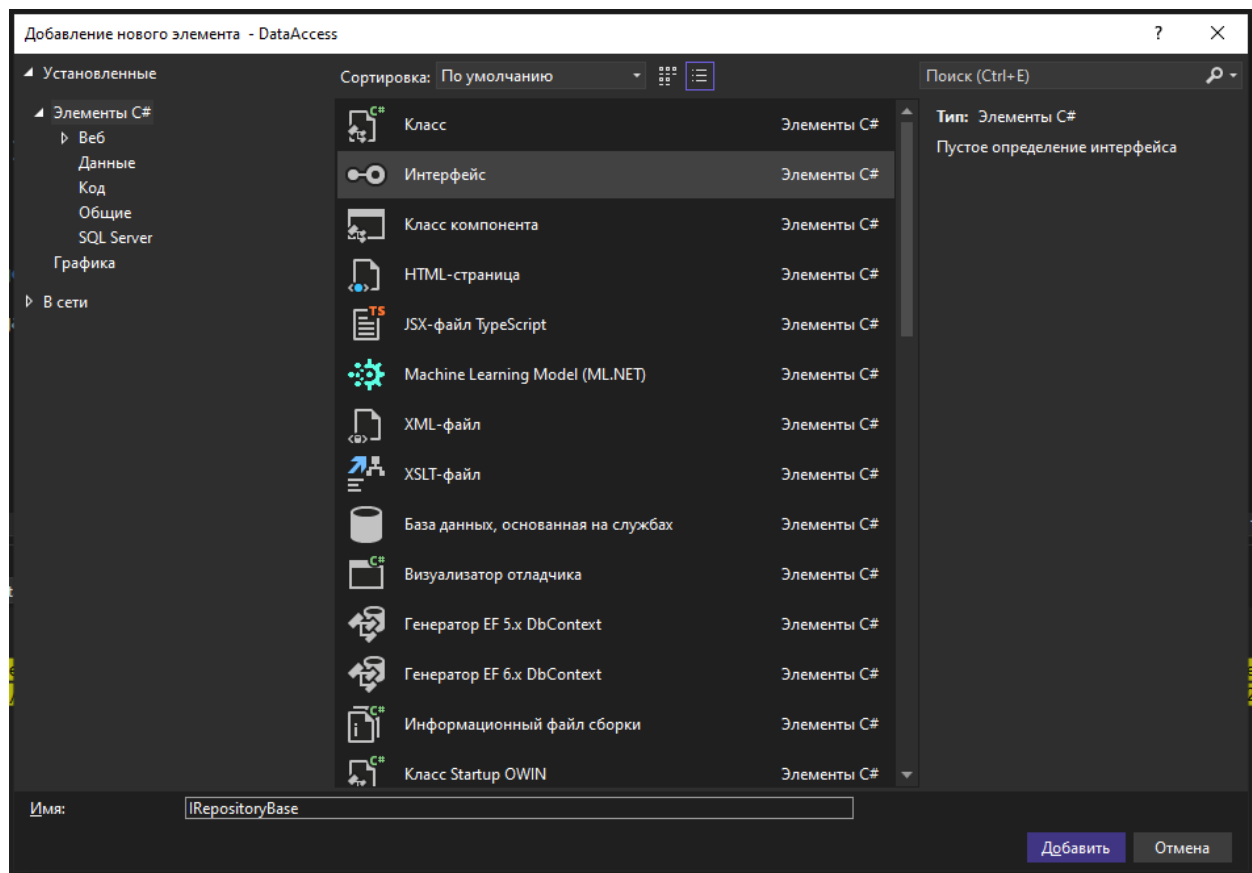
Создайте папку Interfaces в слое DataAccess



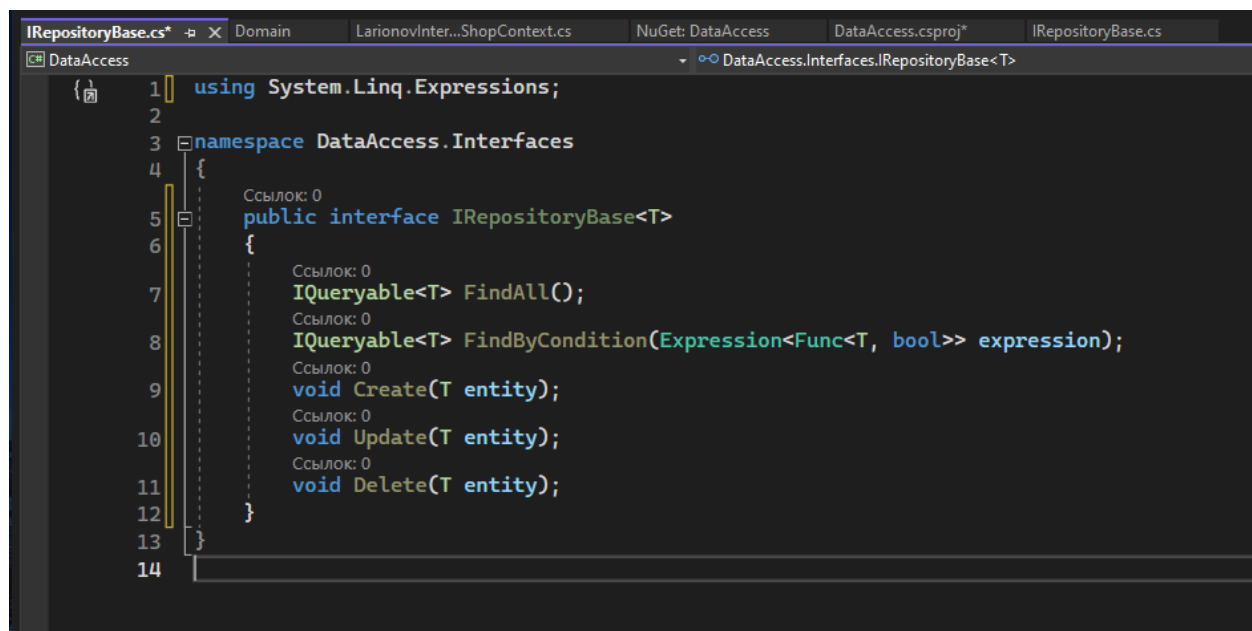
Вызовите окно для создания элементов



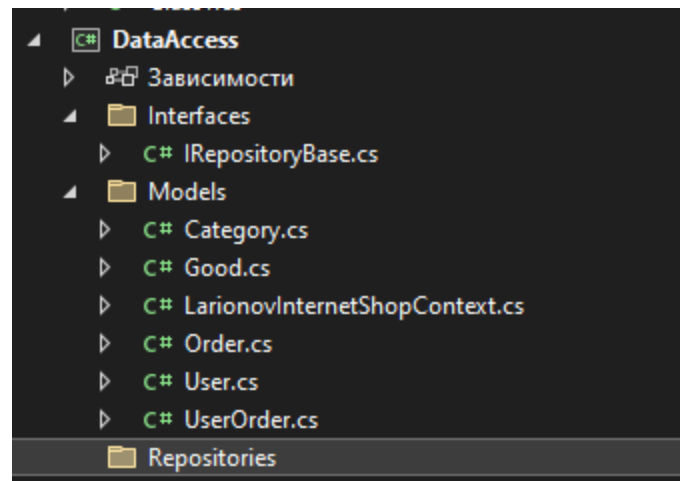
Выберите “Интерфейс” и назовите как IRepositoryBase



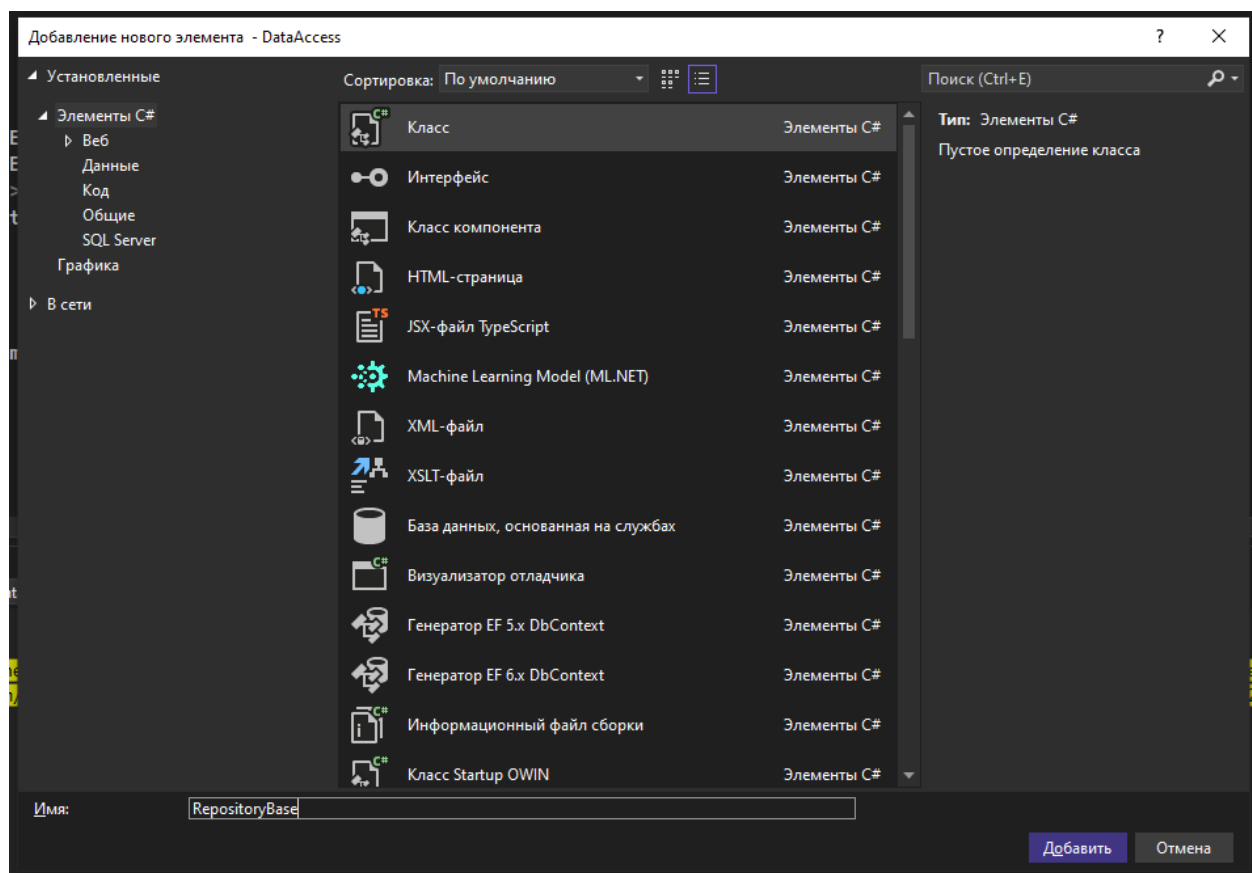
Опишите созданный репозиторий следующим образом



Создайте папку Repositories



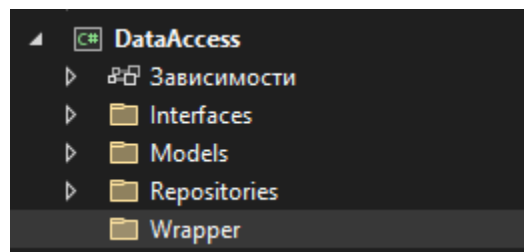
Внутри данной папки создайте класс RepositoryBase



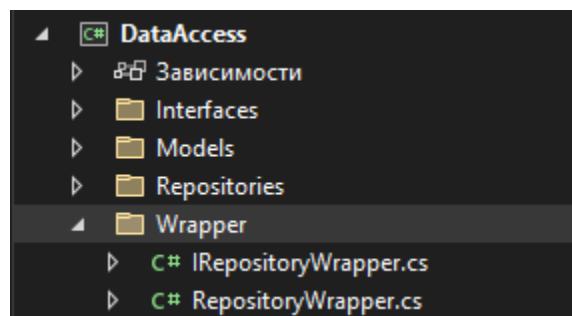
Опишите данный класс следующим образом

```
RepositoryBase.cs* x IRepositoryBase.cs Domain LarionovInter...ShopContext.cs NuGet: DataAccess DataAccess.csproj* IRepositoryBase.cs Update
DataAccess
1 using DataAccess.Models;
2 using System.Linq.Expressions;
3 using DataAccess.Interfaces;
4 using Microsoft.EntityFrameworkCore;
5
6 namespace DataAccess.Repositories
7 {
8     Ссылка: 1
9     public abstract class RepositoryBase<T> : IRepositoryBase<T> where T : class
10    {
11        Ссылка: 6
12        protected LarionovInternetShopContext RepositoryContext { get; set; }
13        Ссылка: 0
14        public RepositoryBase(LarionovInternetShopContext repositoryContext)
15        {
16            RepositoryContext = repositoryContext;
17        }
18        Ссылка: 1
19        public IQueryable<T> FindAll() => RepositoryContext.Set<T>().AsNoTracking();
20        Ссылка: 1
21        public IQueryable<T> FindByCondition(Expression<Func<T, bool>> expression) =>
22            RepositoryContext.Set<T>().Where(expression).AsNoTracking();
23        Ссылка: 1
24        public void Create(T entity) => RepositoryContext.Set<T>().Add(entity);
25        Ссылка: 1
26        public void Update(T entity) => RepositoryContext.Set<T>().Update(entity);
27        Ссылка: 1
28        public void Delete(T entity) => RepositoryContext.Set<T>().Remove(entity);
29    }
30 }
```

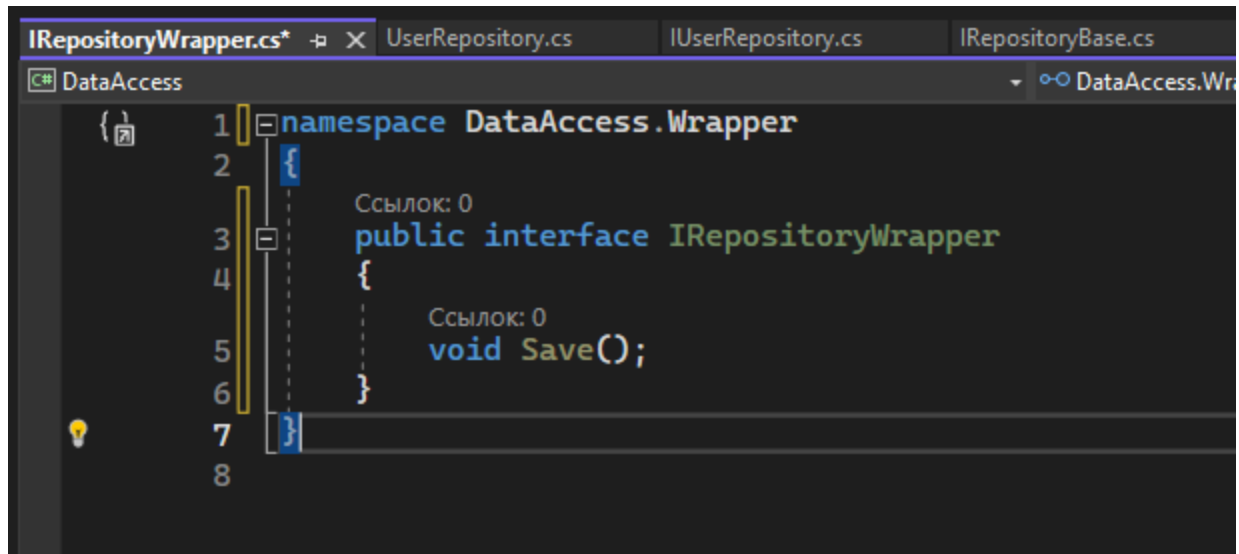
Создайте папку Wrapper



Внутри папки Wrapper создайте интерфейс IRepositoryWrapper и класс RepositoryWrapper

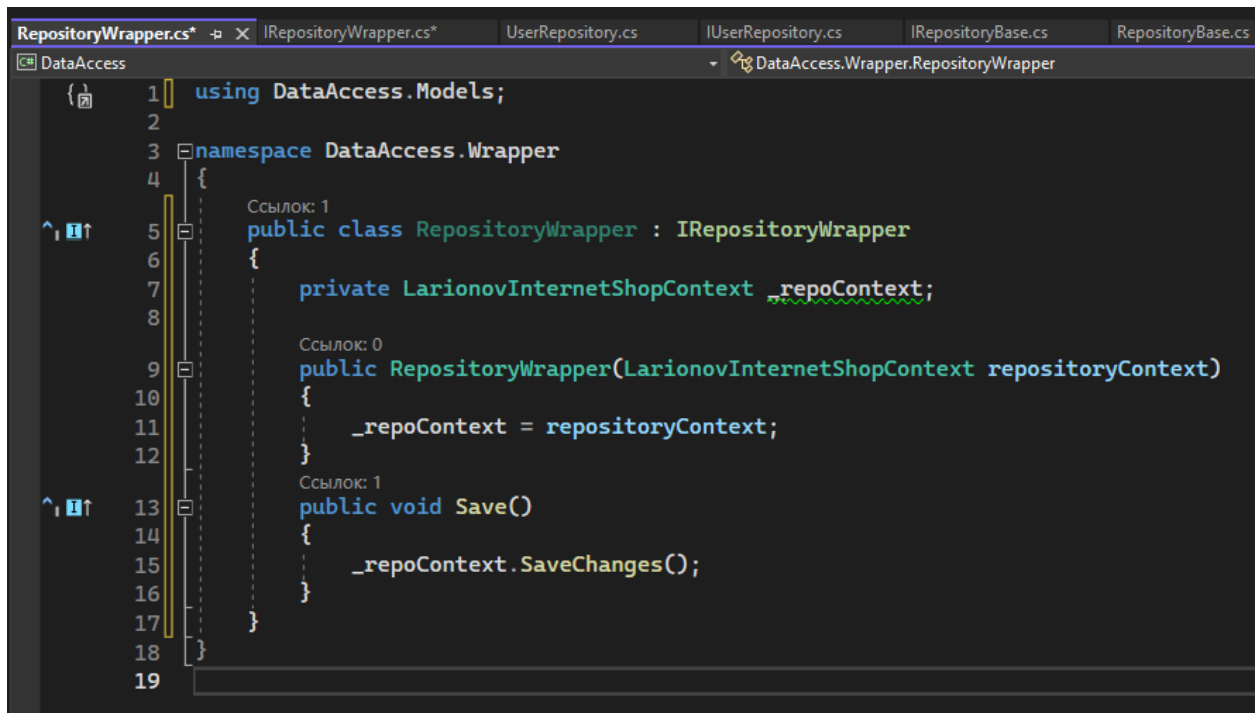


Интерфейс IRepositoryWrapper должен выглядеть следующим образом



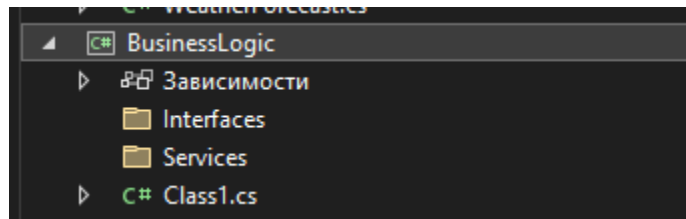
```
1 namespace DataAccess.Wrapper
2 {
3     Ссылка: 0
4     public interface IRepositoryWrapper
5     {
6         Ссылка: 0
7         void Save();
8     }
9 }
```

Класс RepositoryWrapper должен выглядеть следующим образом

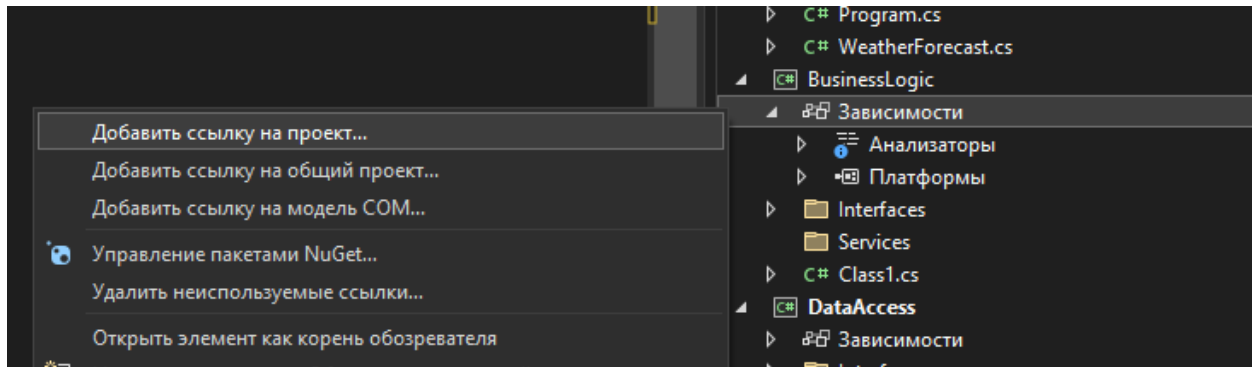


```
1 using DataAccess.Models;
2
3 namespace DataAccess.Wrapper
4 {
5     Ссылка: 1
6     public class RepositoryWrapper : IRepositoryWrapper
7     {
8         private LarionovInternetShopContext _repoContext;
9
10        Ссылка: 0
11        public RepositoryWrapper(LarionovInternetShopContext repositoryContext)
12        {
13            _repoContext = repositoryContext;
14        }
15
16        Ссылка: 1
17        public void Save()
18        {
19            _repoContext.SaveChanges();
20        }
21    }
22 }
```

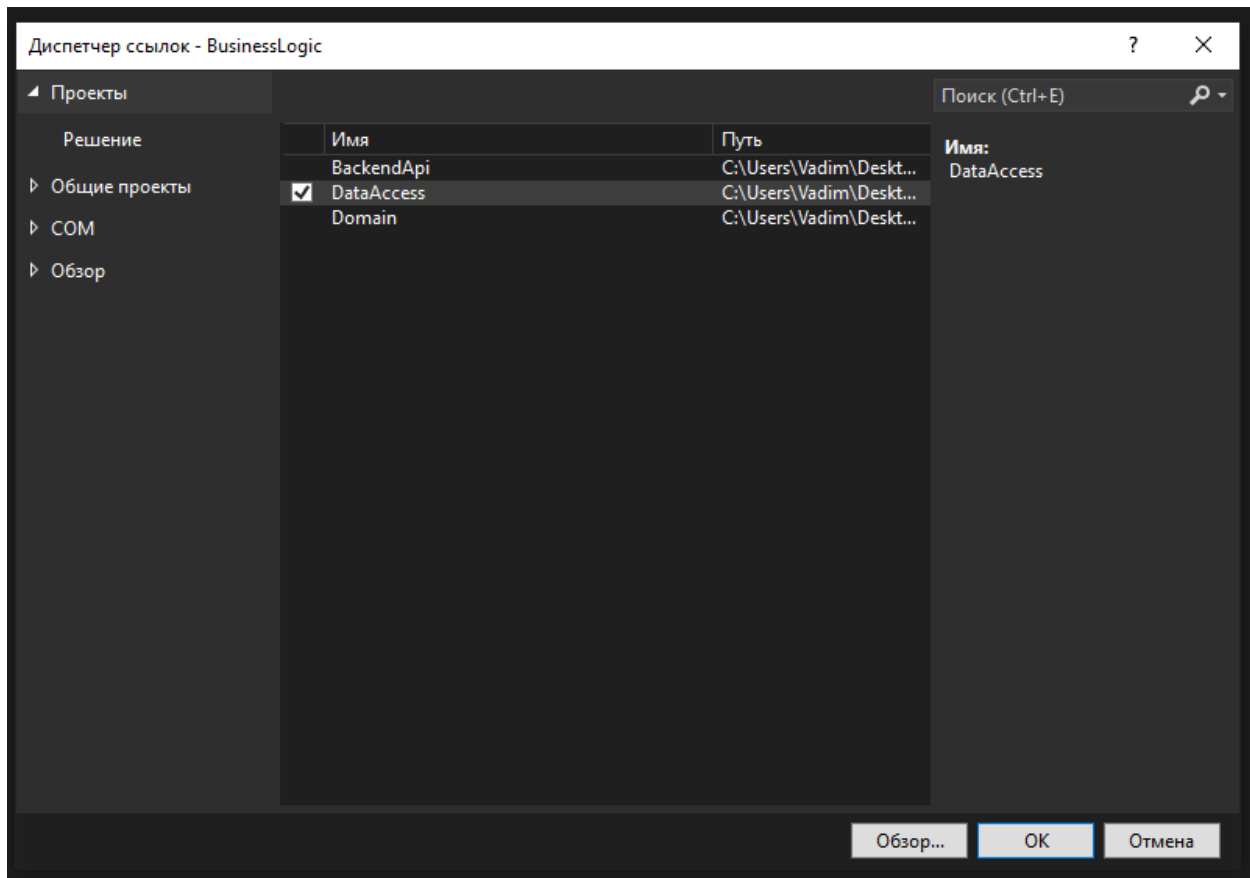
Добавьте для слоя BusinessLogic папки Services и Interfaces



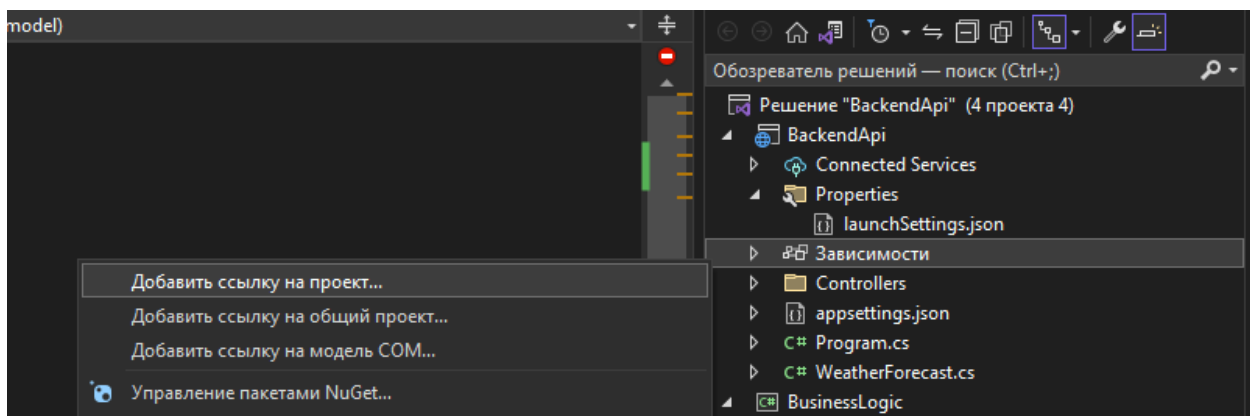
Добавьте зависимость к DataAccess для проекта BusinessLogic

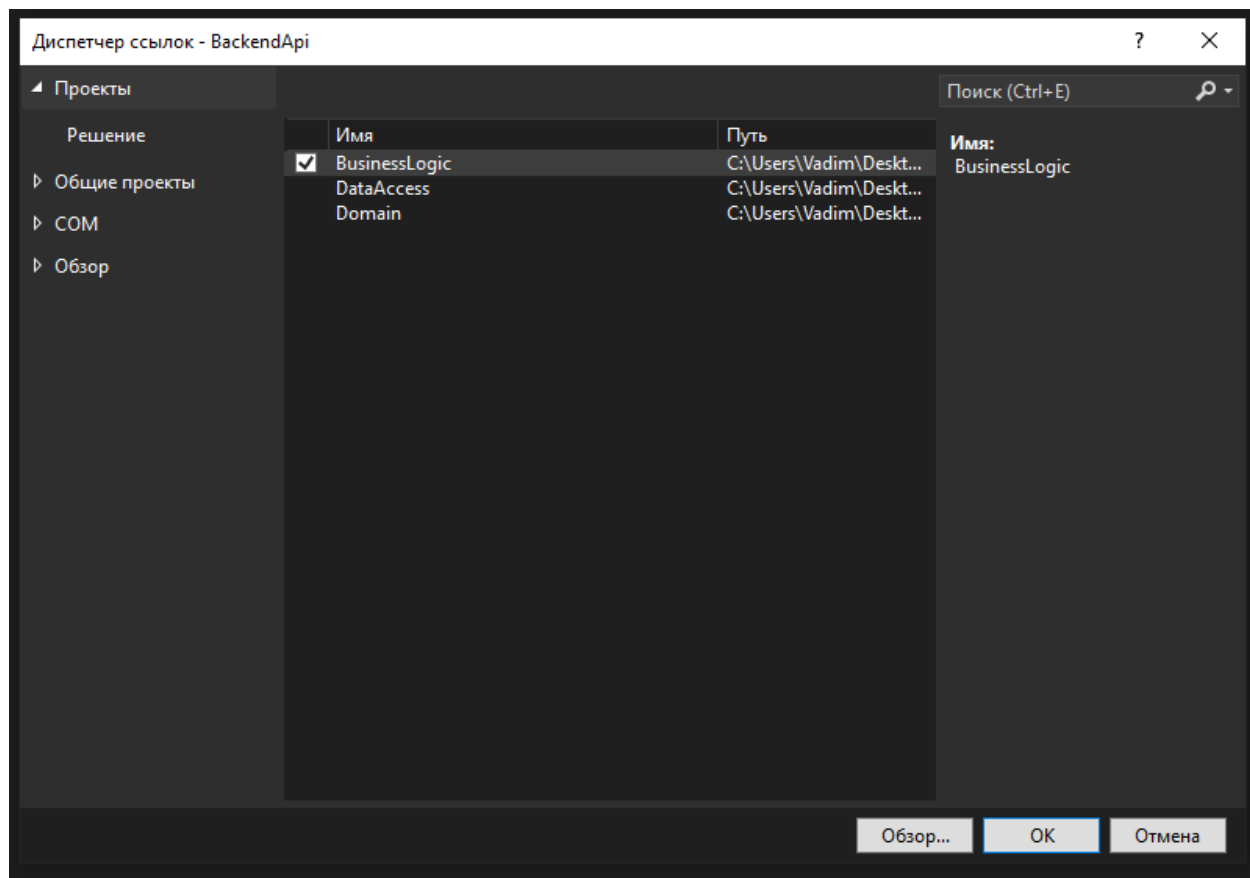


Укажите в качестве зависимости слой DataAccess



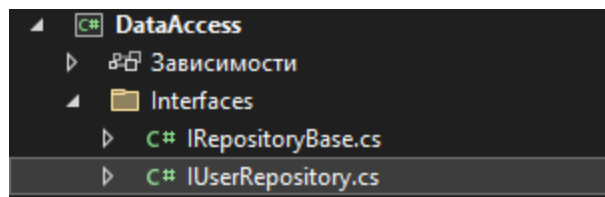
Добавьте зависимость к проекту BusinessLogic для Web API





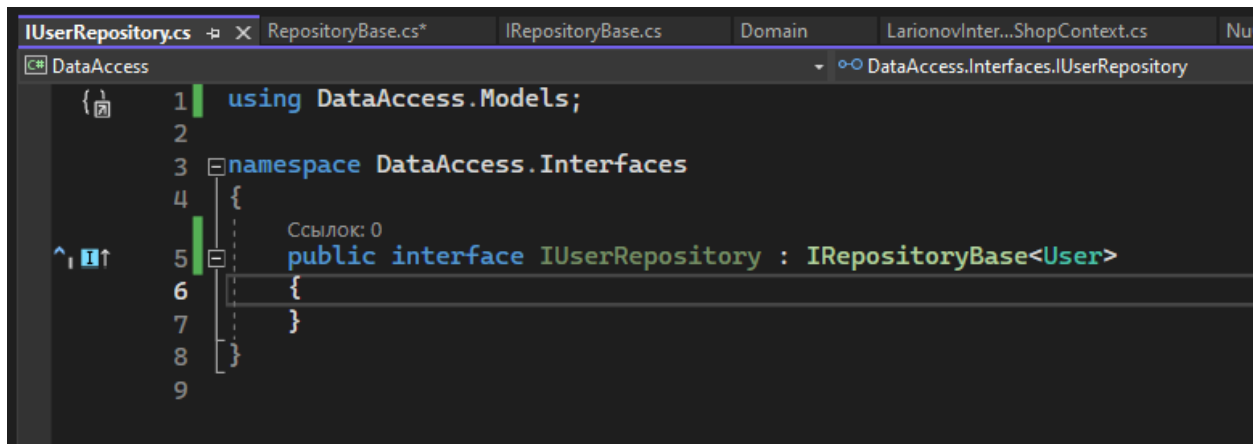
## Цикл разработки в многослойной архитектуре. Сущность User (Пользователь)

Добавьте в папку Interfaces интерфейс IUserRepository



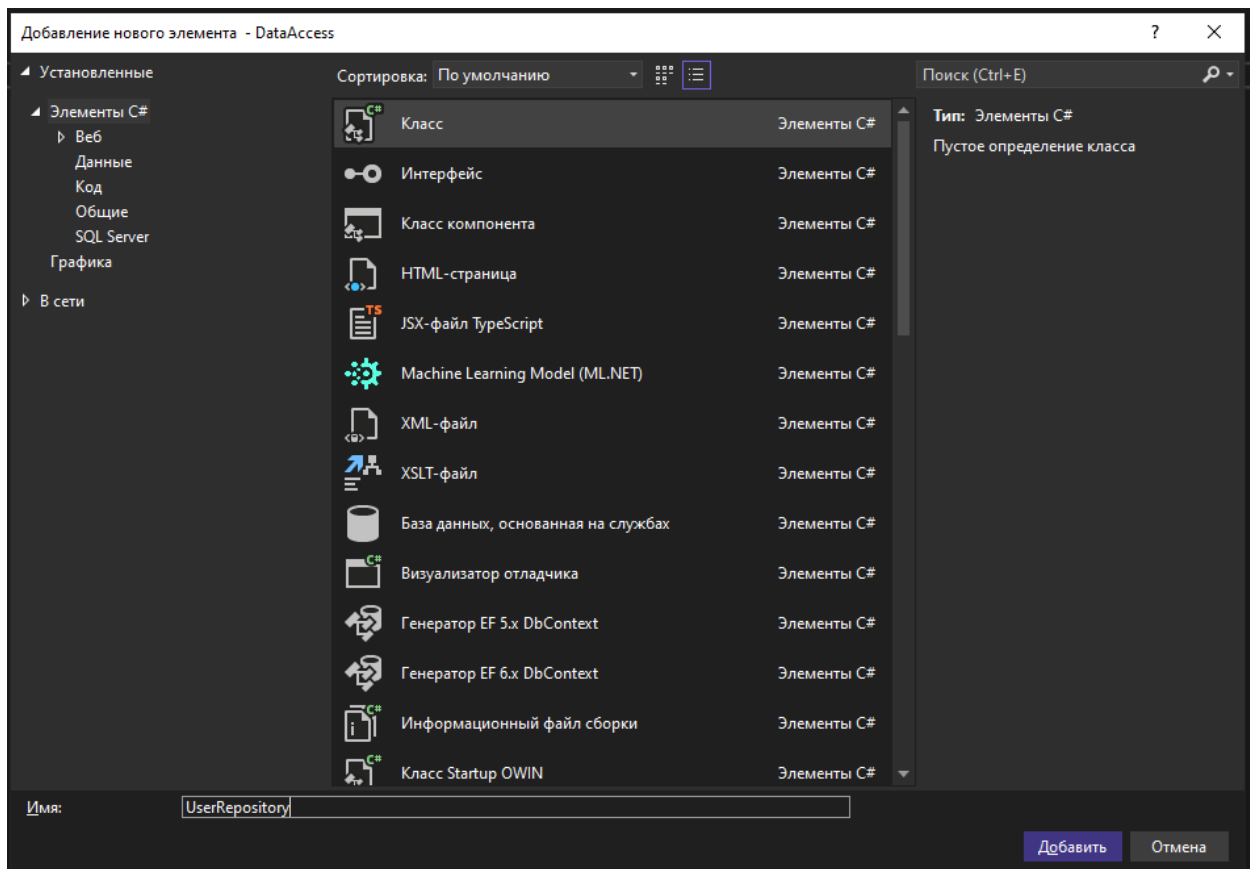
Опишите данный интерфейс следующим образом



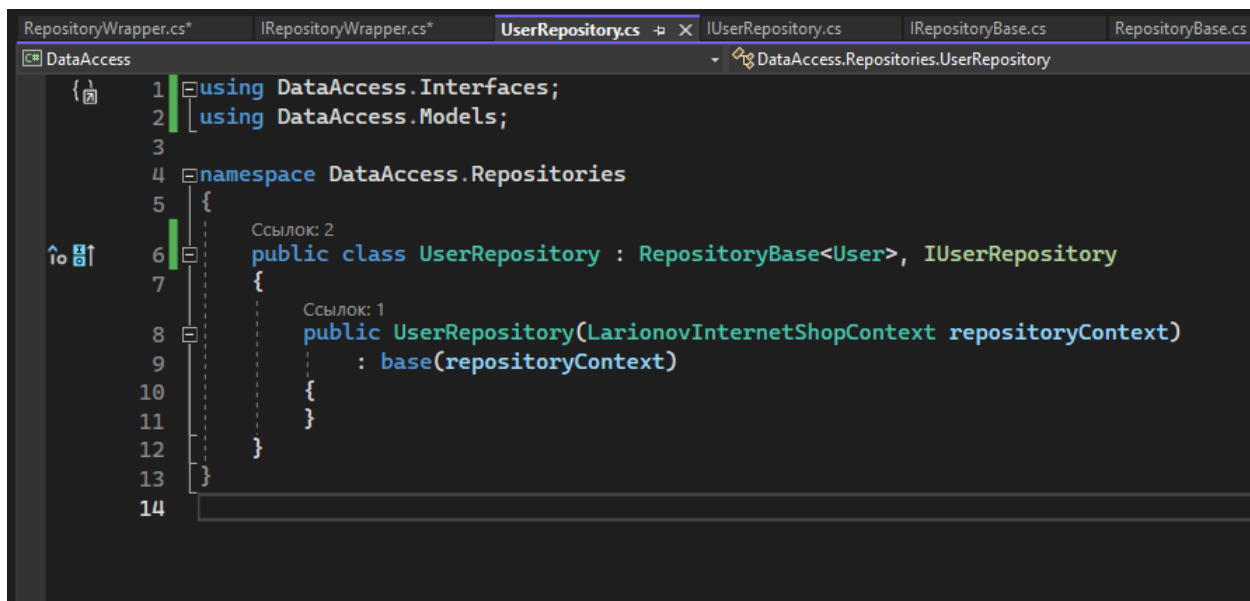


```
1 using DataAccess.Models;
2
3 namespace DataAccess.Interfaces
4 {
5     public interface IUserRepository : IRepositoryBase<User>
6     {
7     }
8 }
9
```

Добавьте в папку Repositories класс UserRepository

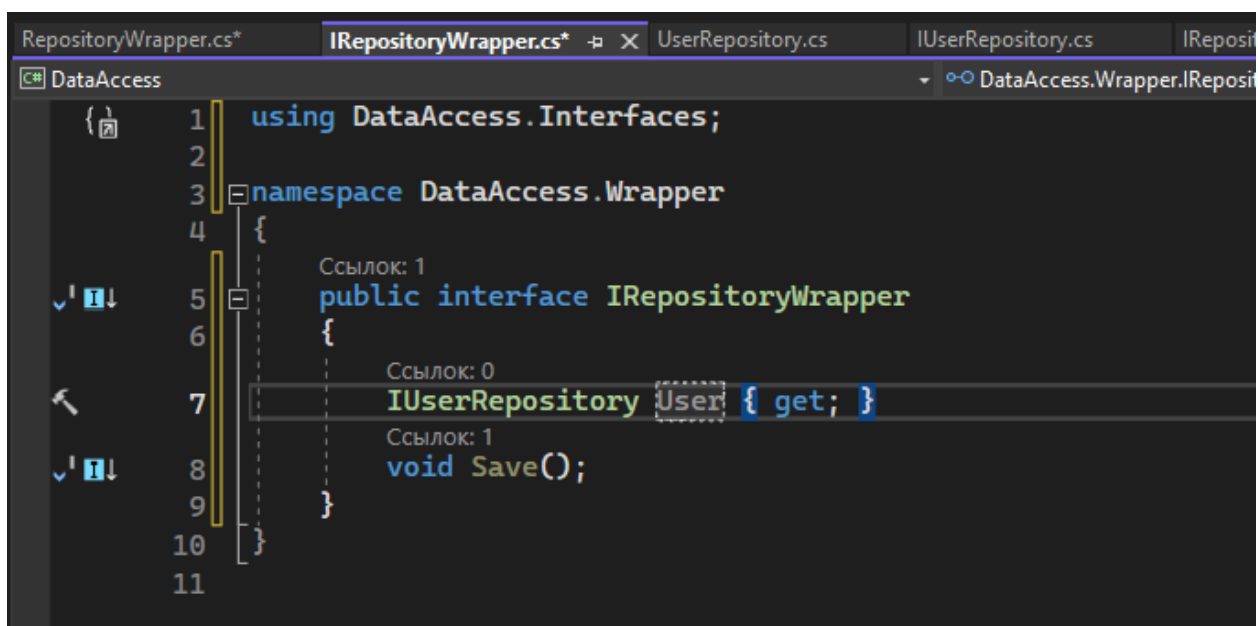


Опишите класс UserRepository следующим образом



```
1 using DataAccess.Interfaces;
2 using DataAccess.Models;
3
4 namespace DataAccess.Repositories
5 {
6     public class UserRepository : RepositoryBase<User>, IUserRepository
7     {
8         public UserRepository(LarionovInternetShopContext repositoryContext)
9             : base(repositoryContext)
10         {
11         }
12     }
13 }
14
```

Дополните интерфейс IRepositoryWrapper следующей строчкой кода



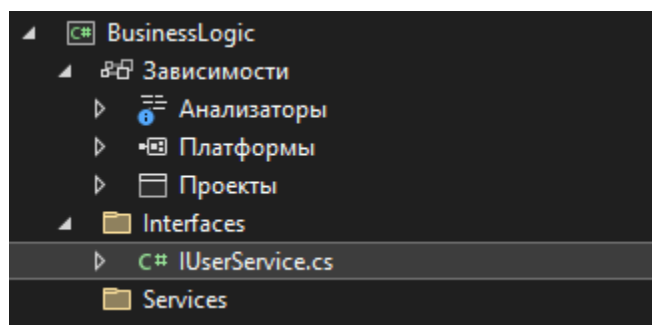
```
1 using DataAccess.Interfaces;
2
3 namespace DataAccess.Wrapper
4 {
5     public interface IRepositoryWrapper
6     {
7         IUserRepository User { get; }
8         void Save();
9     }
10 }
11
```

Дополните класс RepositoryWrapper следующей реализацией свойства User

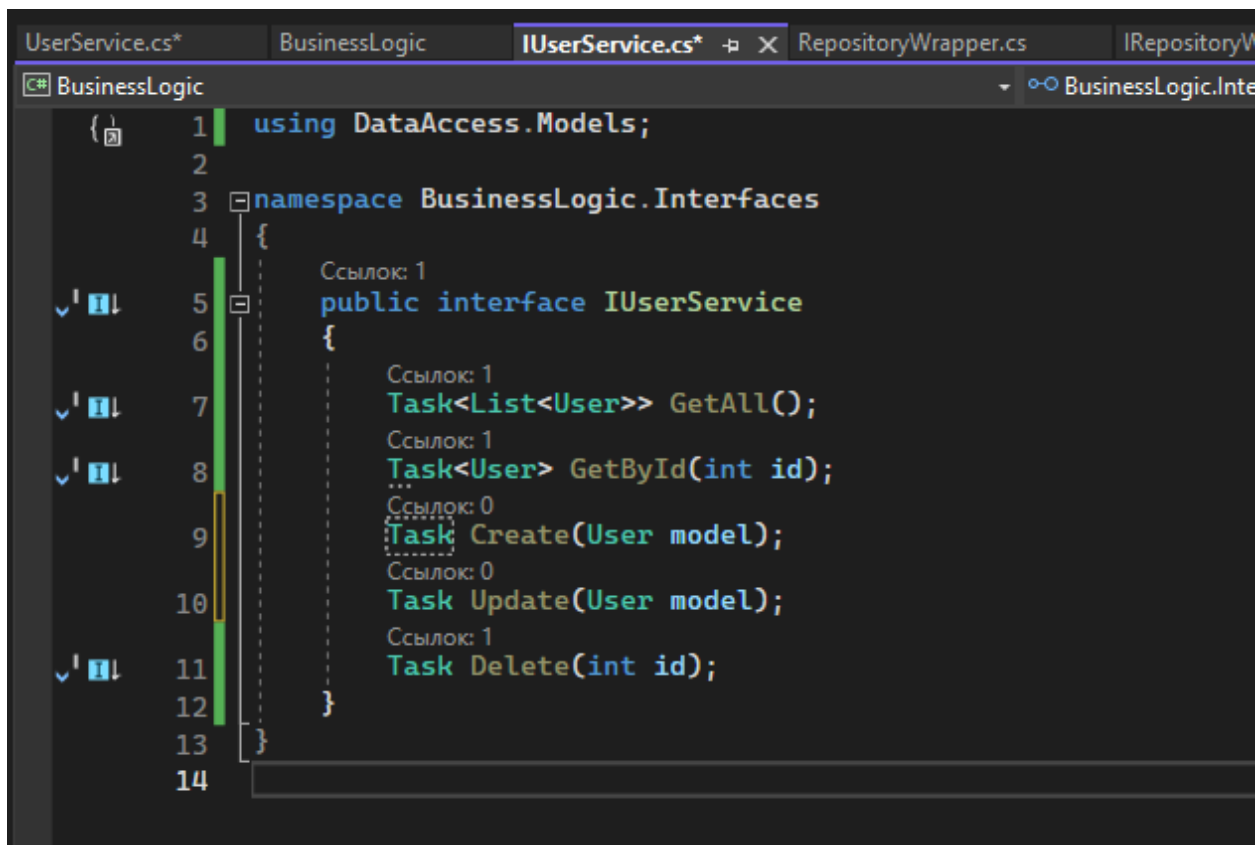
```
RepositoryWrapper.cs | IRepositoryWrapper.cs* | UserRepository.cs | IUserRepository.cs | IRepositoryBase.cs | RepositoryBase.cs
C# DataAccess | DataAccess.Wrapper.RepositoryWrapper

1 using DataAccess.Interfaces;
2 using DataAccess.Models;
3 using DataAccess.Repositories;
4
5 namespace DataAccess.Wrapper
6 {
7     Ссылка: 1
8     public class RepositoryWrapper : IRepositoryWrapper
9     {
10         private LarionovInternetShopContext _repoContext;
11
12         private IUserRepository _user;
13         Ссылка: 1
14         public IUserRepository User
15         {
16             get
17             {
18                 if (_user == null)
19                 {
20                     _user = new UserRepository(_repoContext);
21                 }
22                 return _user;
23             }
24         }
25
26         Ссылка: 0
27         public RepositoryWrapper(LarionovInternetShopContext repositoryContext)
28         {
29             _repoContext = repositoryContext;
30         }
31
32         Ссылка: 1
33         public void Save()
34         {
35             _repoContext.SaveChanges();
36         }
37     }
38 }
```

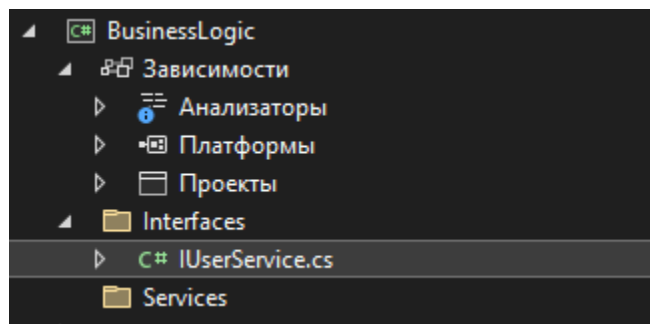
Добавьте в папку Interfaces интерфейс IUserService



Опишите интерфейс следующим образом



Добавьте в папку Services класс UserService



Опишите интерфейс следующим образом

```

using BusinessLogic.Interfaces;
using DataAccess.Models;
using DataAccess.Wrapper;
using Microsoft.EntityFrameworkCore;

namespace BusinessLogic.Services
{
    public class UserService : IUserService

```

```

{
    private IRepositoryWrapper _repositoryWrapper;

    public UserService(IRepositoryWrapper repositoryWrapper)
    {
        _repositoryWrapper = repositoryWrapper;
    }

    public Task<List<User>> GetAll()
    {
        return _repositoryWrapper.User.FindAll().ToListAsync();
    }

    public Task<User> GetById(int id)
    {
        var user = _repositoryWrapper.User
            .FindByCondition(x => x.Id == id).First();
        return Task.FromResult(user);
    }

    public Task Create(User model)
    {
        _repositoryWrapper.User.Create(model);
        _repositoryWrapper.Save();
        return Task.CompletedTask;
    }

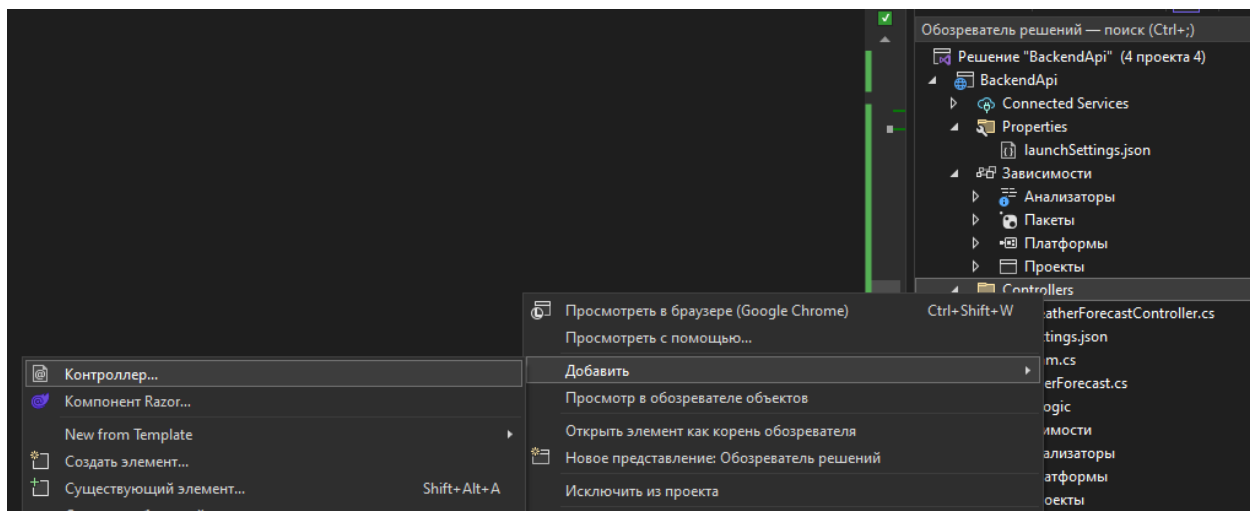
    public Task Update(User model)
    {
        _repositoryWrapper.User.Update(model);
        _repositoryWrapper.Save();
        return Task.CompletedTask;
    }

    public Task Delete(int id)
    {
        var user = _repositoryWrapper.User
            .FindByCondition(x => x.Id == id).First();

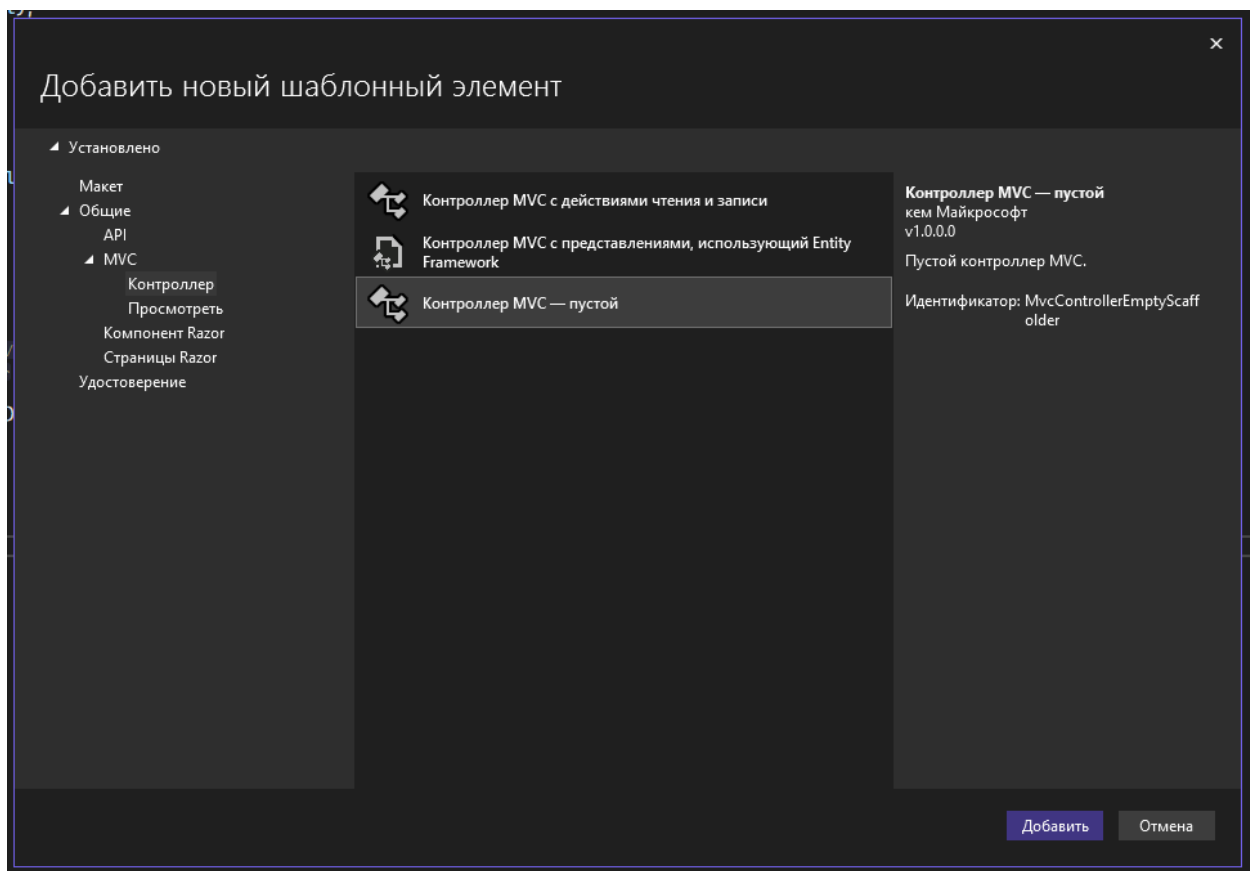
        _repositoryWrapper.User.Delete(user);
        _repositoryWrapper.Save();
        return Task.CompletedTask;
    }
}
}

```

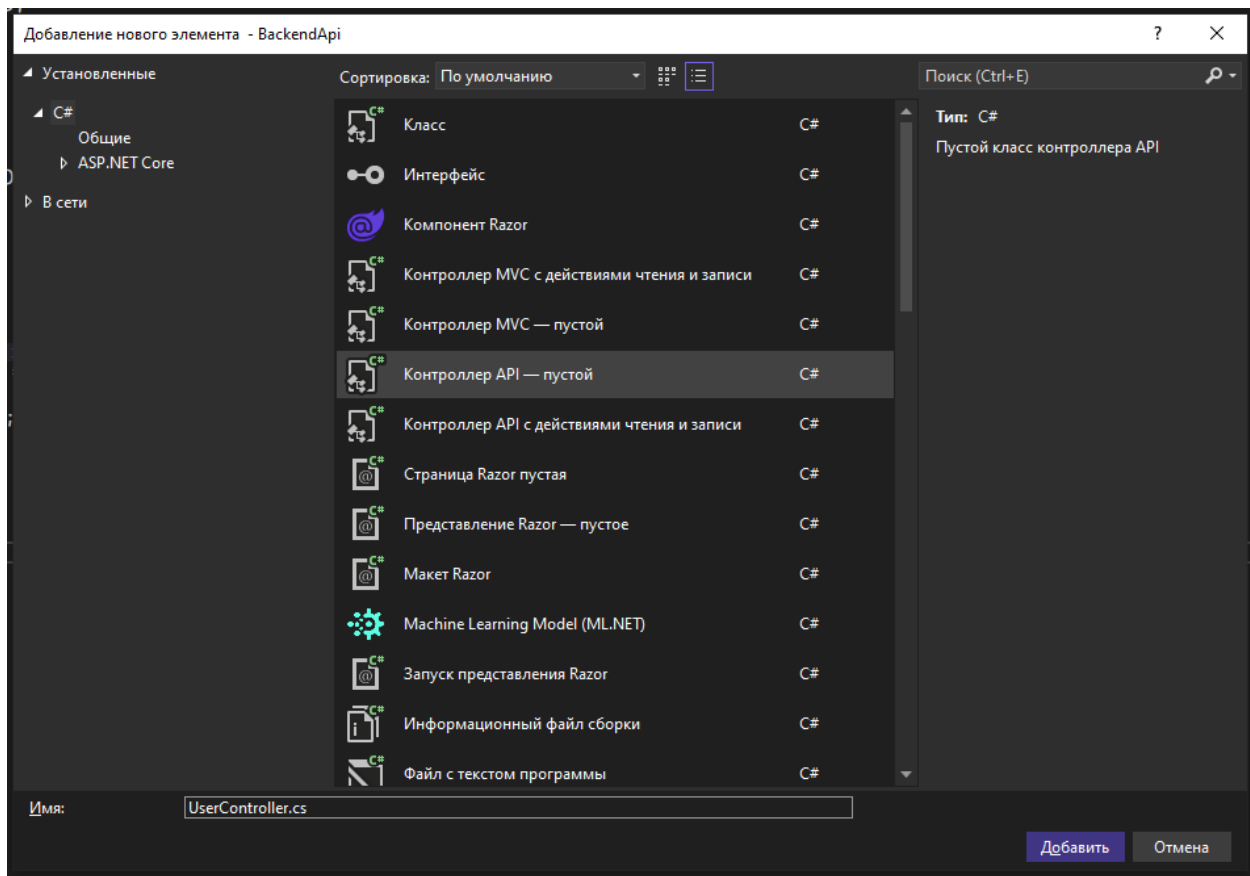
Создайте контроллер UserController



Выберите пункт “Пустой”



Выберите “Контроллер API - пустой” и назовите его как UserController



Реализуйте контроллер следующим образом

```
using BusinessLogic.Interfaces;
using DataAccess.Models;
using Microsoft.AspNetCore.Mvc;

namespace BackendApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class UserController : ControllerBase
    {
        private IUserService _userService;
        public UserController(IUserService userService)
        {
            _userService = userService;
        }

        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            return Ok(await _userService.GetAll());
        }
    }
}
```

```

[HttpGet("{id}")]
public async Task<IActionResult> GetById(int id)
{
    return Ok(await _userService.GetById(id));
}

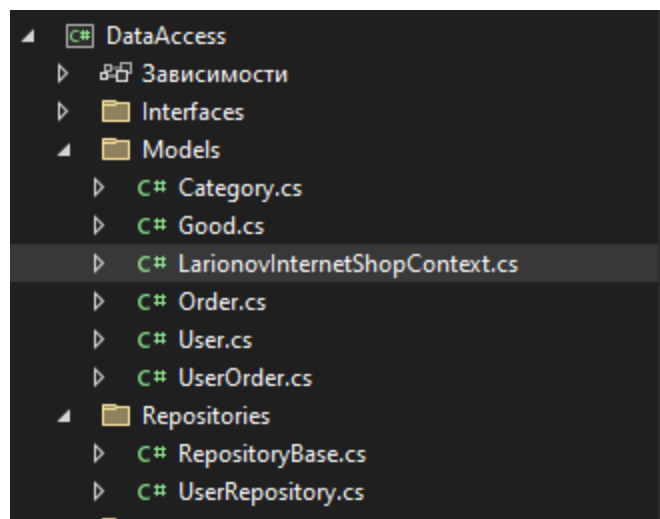
[HttpPost]
public async Task<IActionResult> Add(User user)
{
    await _userService.Create(user);
    return Ok();
}

[HttpPut]
public async Task<IActionResult> Update(User user)
{
    await _userService.Update(user);
    return Ok();
}

[HttpDelete]
public async Task<IActionResult> Delete(int id)
{
    await _userService.Delete(id);
    return Ok();
}
}
}

```

Перейдите в класс Context, описывающий модели таблиц базы данных, который расположен в слое DataAccess в папке Models





```
BusinessLogic WeatherForecastController.cs Program.cs* IUserService.cs RepositoryWrapper.cs IRepositoryWrapper.cs UserRepository.cs IUserRepository.cs LarionovInter...hopContext.cs x
DataAccess
4 using Microsoft.EntityFrameworkCore.Metadata;
5
6 namespace DataAccess.Models
7 {
8     public partial class LarionovInternetShopContext : DbContext
9     {
10         public LarionovInternetShopContext()
11         {
12         }
13
14         public LarionovInternetShopContext(DbContextOptions<LarionovInternetShopContext> options)
15             : base(options)
16         {
17         }
18
19         public virtual DbSet<Category> Categories { get; set; } = null!;
20         public virtual DbSet<Good> Goods { get; set; } = null!;
21         public virtual DbSet<Order> Orders { get; set; } = null!;
22         public virtual DbSet<User> Users { get; set; } = null!;
23         public virtual DbSet<UserOrder> UserOrders { get; set; } = null!;
24     }
25 }
```

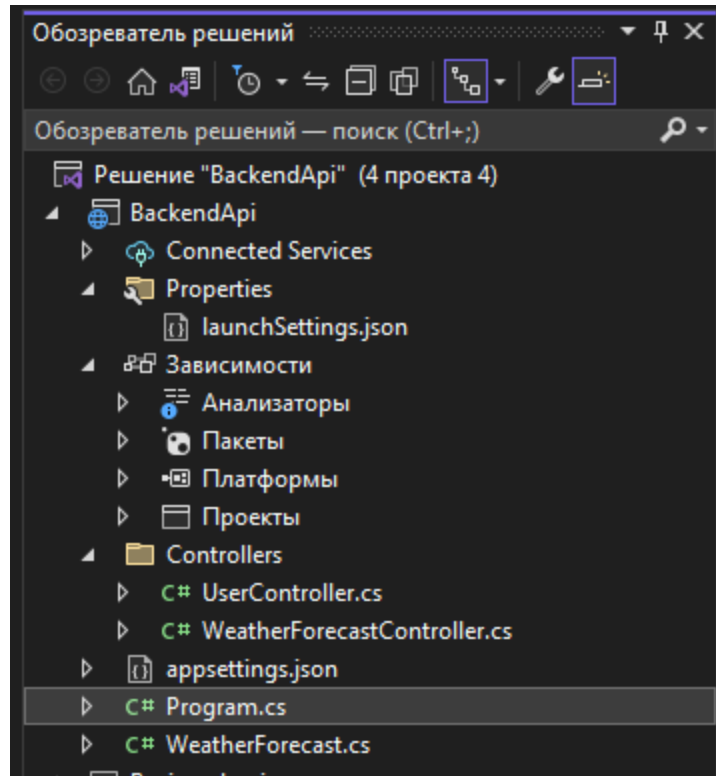
Скопируйте вашу строку подключения к вашей БД из метода OnConfiguring

```
20 public virtual DbSet<Good> Goods { get; set; } = null!;
21 public virtual DbSet<Order> Orders { get; set; } = null!;
22 public virtual DbSet<User> Users { get; set; } = null!;
23 public virtual DbSet<UserOrder> UserOrders { get; set; } = null!;
24
25 protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
26 {
27     if (!optionsBuilder.IsConfigured)
28     {
29         #warning To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding
30         optionsBuilder.UseSqlServer("Server=LAPTOP-MQNS35FH;Database=Larionov-InternetShop;User Id=sa;Password=12345;");
31     }
32 }
33
34 protected override void OnModelCreating(ModelBuilder modelBuilder)
35 {
36     modelBuilder.Entity<Category>(entity =>
37     {
38         entity.Property(e => e.Description).HasMaxLength(300);
39         entity.Property(e => e.Name).HasMaxLength(50);
40     });
41 }
```

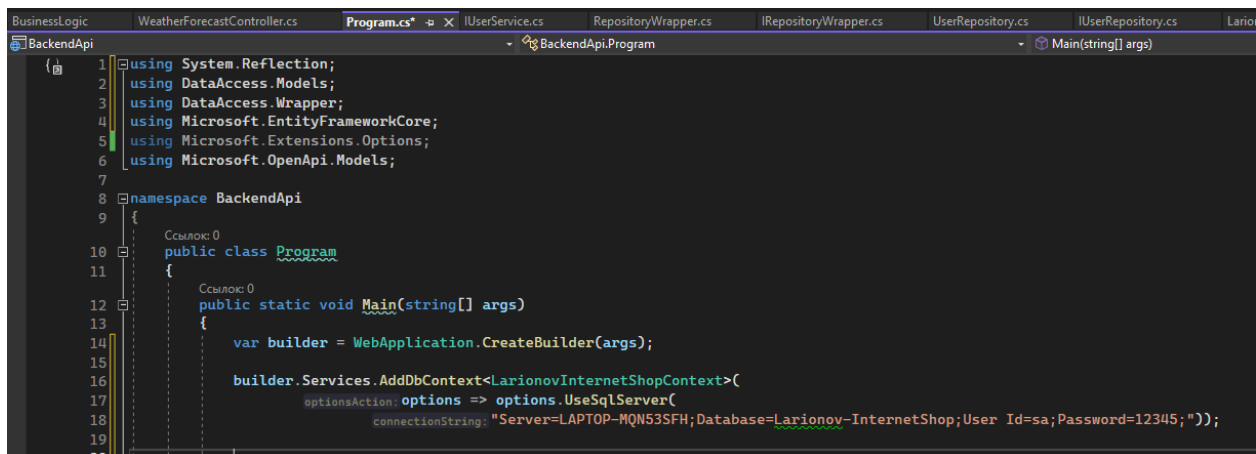
Удалите полностью метод OnConfiguring

```
21 public virtual DbSet<Order> Orders { get; set; } = null!;
22 public virtual DbSet<User> Users { get; set; } = null!;
23 public virtual DbSet<UserOrder> UserOrders { get; set; } = null!;
24
25 protected override void OnModelCreating(ModelBuilder modelBuilder)
26 {
27     modelBuilder.Entity<Category>(entity =>
28     {
29         entity.Property(e => e.Description).HasMaxLength(300);
30         entity.Property(e => e.Name).HasMaxLength(50);
31     });
32
33     modelBuilder.Entity<Good>(entity =>
34     {
35         entity.Property(e => e.CategoryId).HasColumnName("Category_id");
36         entity.Property(e => e.Description).HasMaxLength(300);
37     });
38 }
```

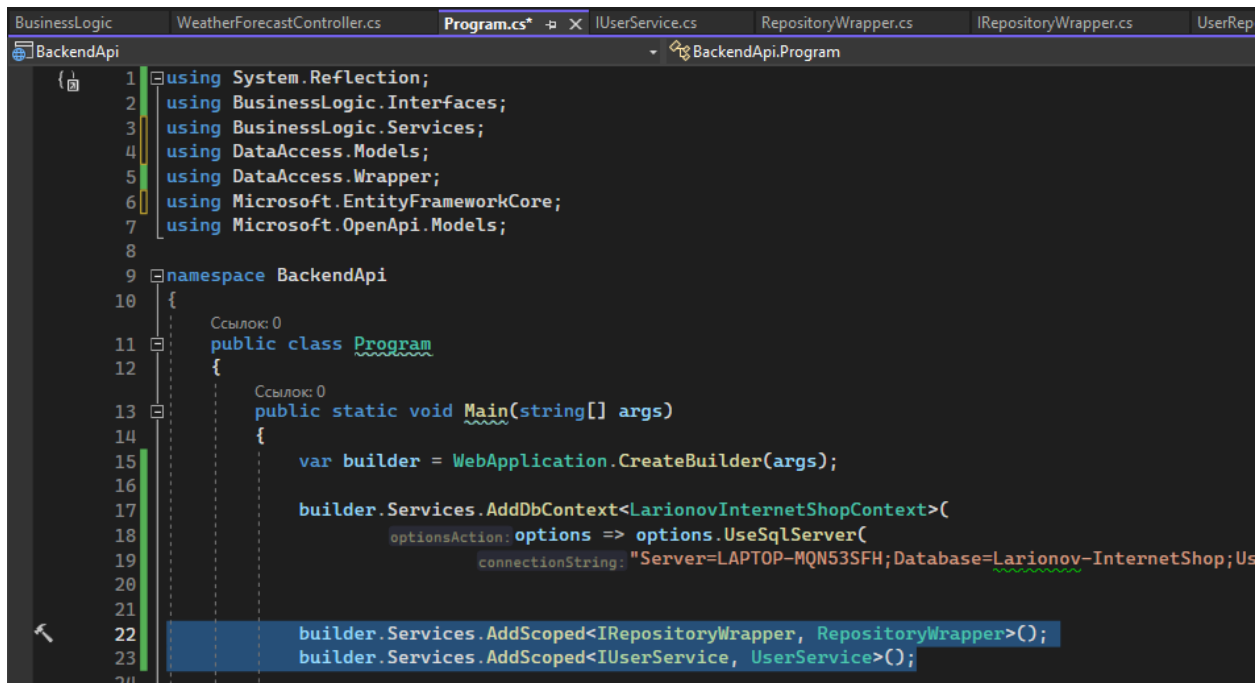
Перейдите в класс Program.cs



Добавьте следующую строку кода для описания подключения к БД

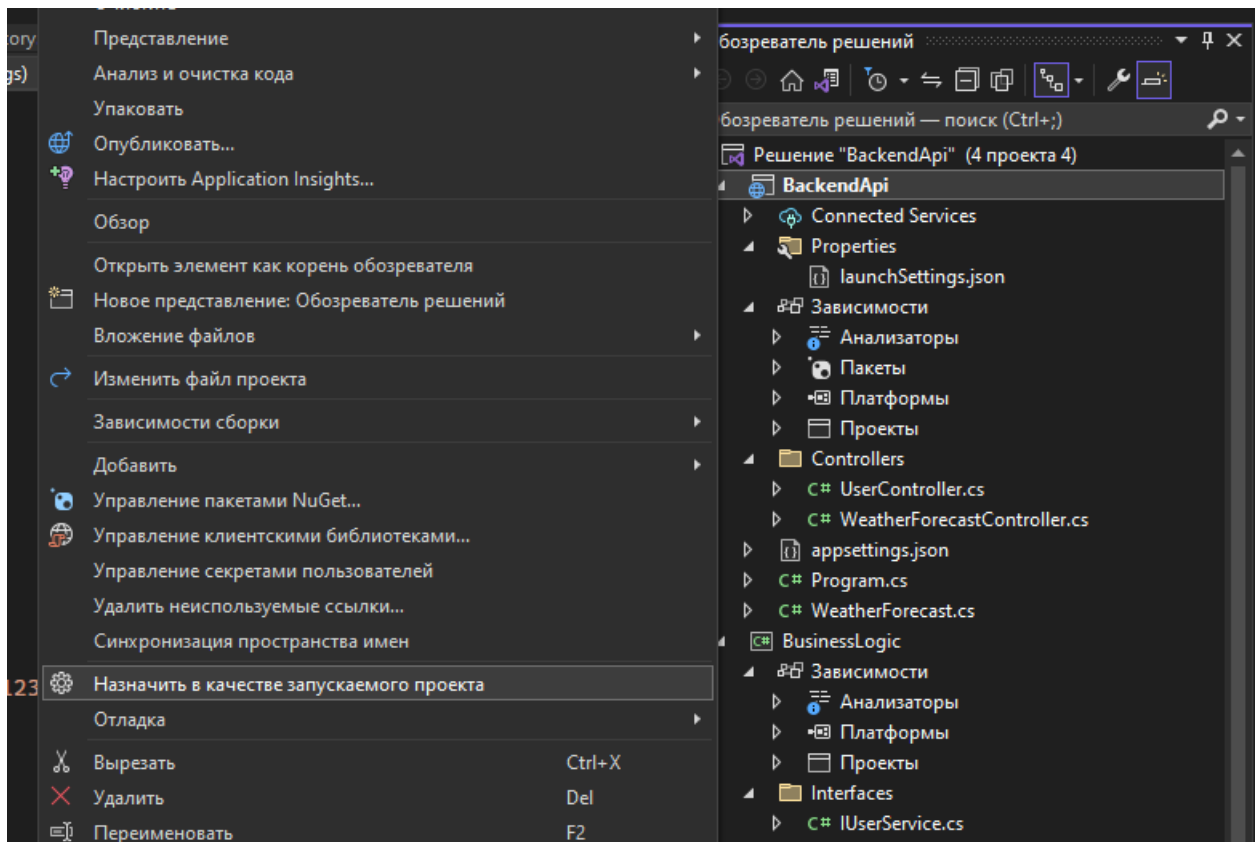


Добавьте строки для внедрения зависимостей следующих интерфейсов



```
1 using System.Reflection;
2 using BusinessLogic.Interfaces;
3 using BusinessLogic.Services;
4 using DataAccess.Models;
5 using DataAccess.Wrapper;
6 using Microsoft.EntityFrameworkCore;
7 using Microsoft.OpenApi.Models;
8
9 namespace BackendApi
10 {
11     public class Program
12     {
13         public static void Main(string[] args)
14         {
15             var builder = WebApplication.CreateBuilder(args);
16
17             builder.Services.AddDbContext<LarionovInternetShopContext>((
18                 optionsAction: options => options.UseSqlServer(
19                     connectionString: "Server=LAPTOP-MQN53SFH;Database=Larionov-InternetShop;Us
20
21
22             builder.Services.AddScoped<IRepositoryWrapper, RepositoryWrapper>();
23             builder.Services.AddScoped<IUserService, UserService>();
24         }
25     }
26 }
```

Назначьте проект Web API в качестве запускаемого проекта



Запустите проект и проверьте работу метода получения по его идентификатору

The screenshot displays the Swagger UI interface for an API. At the top, a red bar indicates a DELETE method for the endpoint `/api/User`. Below this, a blue bar shows the selected GET method for the endpoint `/api/User/{id}`. The **Parameters** section contains a table with one parameter: `id`, which is required, of type `integer($int32)`, and located in the `path`. The value `1` is entered in the input field. Below the parameters, there are `Execute` and `Clear` buttons. The **Responses** section shows a successful response with status code `200`. The **Response body** is displayed as a JSON object: 

```
{  "id": 1,  "firstname": "Meen",  "lastname": "Meenon",  "middlename": "Meenosee",  "birthdate": "2000-01-01T00:00:00",  "login": "ivan",  "email": "ivan@mail.ru",  "password": "12345",  "userOrders": []}
```

. A `Download` button is available next to the response body. The **Response headers** section shows `content-type: application/json; charset=utf-8`.