Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Звіт
З виробничої практики

Виконав:
Студент 3-го курсу
Бухало Михайло Олександрович
Керівник практики:
Верес Максим Миколайович

Київ-2020

Завдання 1

Реалізація REST сервісу на ASP .NET core

Зміст

1. Завдання
    a. Постановка задачі
    b. Вимоги до оформлення звіту
    c. Варіанти завдань
2. Приклад виконання роботи

Постановка задачі

Розробіть REST web-service за допомогою ASP .NET core який реалізує методи доступу до СУБД(MySQL) - перегляд параметрів, додавання, видалення, зміна значень .

Вимоги до оформлення звіту

1. Титульний лист
2. Постановка задачі
3. Код написаної програми
4. Опис програми (опис класів, методів, полів)

Варіанти завдань

Вариант 1	
Предметная область	Карта мира
Объекты	Страны, Города
Примечание	Карта мира содержит множество *стран*. Для каждой *страны* определено множество *городов*.
Требуемые операции	Выдача полного списка городов с указанием названия страны

Вариант 2	
Предметная область	Библиотека
Объекты	Авторы, Книги
Примечание	Книги в библиотеке сгруппированы по *авторам*. У каждого *автора* имеется множество *книг*.
Требуемые операции	Выдача полного списка книг с указанием ФИО автора

Вариант 3	
Предметная область	Отдел кадров
Объекты	Подразделения, Сотрудники
Примечание	Имеется множество *подразделений* предприятия. В каждом *подразделении* работает множество *сотрудников*.
Требуемые операции	Выдача списка сотрудников с указанием названия подразделения

Вариант 4	
Предметная область	Учебный отдел
Объекты	Группы, Студенты
Примечание	Имеется множество учебных *групп*. Каждая группа включает в себя множество *студентов*.
Требуемые операции	Выдача полного списка студентов с указанием названия группы

Вариант 5	
Предметная область	Автосалон
Объекты	Производители автомобилей, Марки
Примечание	*Марки* автомобилей сгруппированы по производителям. У каждого *производителя* имеется множество *марок*.
Требуемые операции	Выдача полного списка марок с названием производителя

Вариант 6	
Предметная область	Агентство новостей
Объекты	Категории новостей, Новости
Примечание	Новости сгруппированы по *категориям*. У каждой *категории* имеется множество *новостей*.
Требуемые операции	Выдача полного списка новостей с указанием категории

Вариант 7	
Предметная область	Продуктовый магазин
Объекты	Категория продукта, Продукт
Примечание	*Продукты* в магазине сгруппированы по *категориям*. Для каждой *категории* определено множество *продуктов*.
Требуемые операции	Выдача списка продуктов с указанием категории

Вариант 8	
Предметная область	Футбол
Объекты	Команды, Игроки
Примечание	Имеется множество футбольных *команд*. Для каждой *команды* определено множество *игроков*.
Требуемые операции	Выдача полного списка игроков с указанием названия команды

Вариант 9	
Предметная область	Музыкальный магазин
Объекты	Исполнители, Альбомы
Примечание	В музыкальном магазине *альбомы* сгруппированы по *исполнителям*. Для каждого *исполнителя* задано множество *альбомов*.
Требуемые операции	Выдача полного списка альбомов с указанием исполнителя

Вариант 10	
Предметная область	Аэропорт
Объекты	Авиакомпании, Рейсы
Примечание	Имеется множество *авиакомпаний*. Для каждой *авиакомпании* определены ее *рейсы*.
Требуемые операции	Выдача полного списка рейсов с указанием названия авиакомпании

Вариант 11	
Предметная область	Файловая система
Объекты	Папки, Файлы
Примечание	Имеется множество *папок* (независимых друг от друга). Для каждой *папки* определено множество *файлов*.
Требуемые операции	Выдача списка файлов с указанием папки

Вариант 12	
Предметная область	Расписание занятий
Объекты	Дни недели, Занятия
Примечание	Имеется множество *дней*. Для каждого *дня* определен перечень *занятий*.
Требуемые операции	Выдача полного списка занятий с указанием дня

Вариант 13	
Предметная область	Записная книжка
Объекты	Календарные дни, Мероприятия
Примечание	Имеется множество *дней*. Для каждого *дня* определен перечень *мероприятий*.
Требуемые операции	Выдача полного списка мероприятий с указанием дня

Вариант 14	
Предметная область	Видеомагазин
Объекты	Жанры, Фильмы
Примечание	Имеется множество *жаров*. Для каждого *жанра* определен перечень *фильмов*.
Требуемые операции	Выдача списка фильмов с указанием жанра

Вариант 15	
Предметная область	Железная дорога
Объекты	Дороги, Станции
Примечание	Имеется множество *железных дорог*. В ведомстве каждой дороги находится множество *станций*.
Требуемые операции	Выдача полного списка станций с указанием названия дороги

Вариант 16	
Предметная область	Склад
Объекты	Секции, Товары
Примечание	*Товары* на складе сгруппированы по *секциям*. Для каждой *секции* задано множество *товаров*.
Требуемые операции	Выдача списка товаров с указанием секции

Вариант 17	
Предметная область	Кафедра университета
Объекты	Преподаватели, Дисциплины
Примечание	На кафедре имеется множество *преподавателей*. Для каждого *преподавателя* задано множество *дисциплин*.
Требуемые операции	Выдача списка дисциплин с указанием ФИО преподавателя

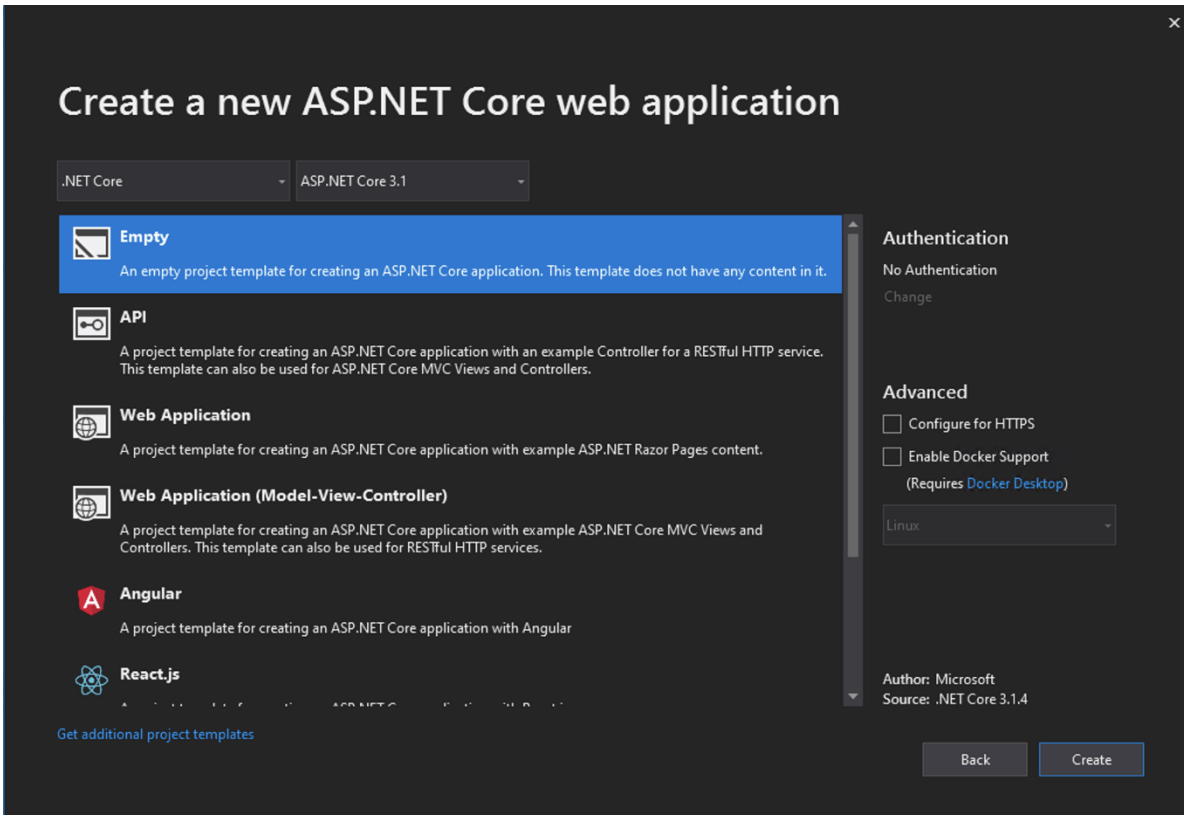
Вариант 18	
Предметная область	Программное обеспечение
Объекты	Производители, Программные продукты
Примечание	Программные *продукты* сгруппированы по *производителям*. Для каждого *производителя* задано множество *продуктов*.
Требуемые операции	Выдача списка продуктов с указанием производителя

Вариант 19	
Предметная область	Геометрия
Объекты	Многоугольники, Вершины
Примечание	Имеется множество *многоугольников*. Каждый *многоугольник* состоит из произвольного числа *вершин*.
Требуемые операции	Выдача полного списка многоугольников с указанием всех вершин

Вариант 20	
Предметная область	Схема метро
Объекты	Линии, Станции
Примечание	Имеется множество *линий* метрополитена. Каждая *линия* состоит из последовательности *станций*.
Требуемые операции	Выдача списка станций с указанием линии

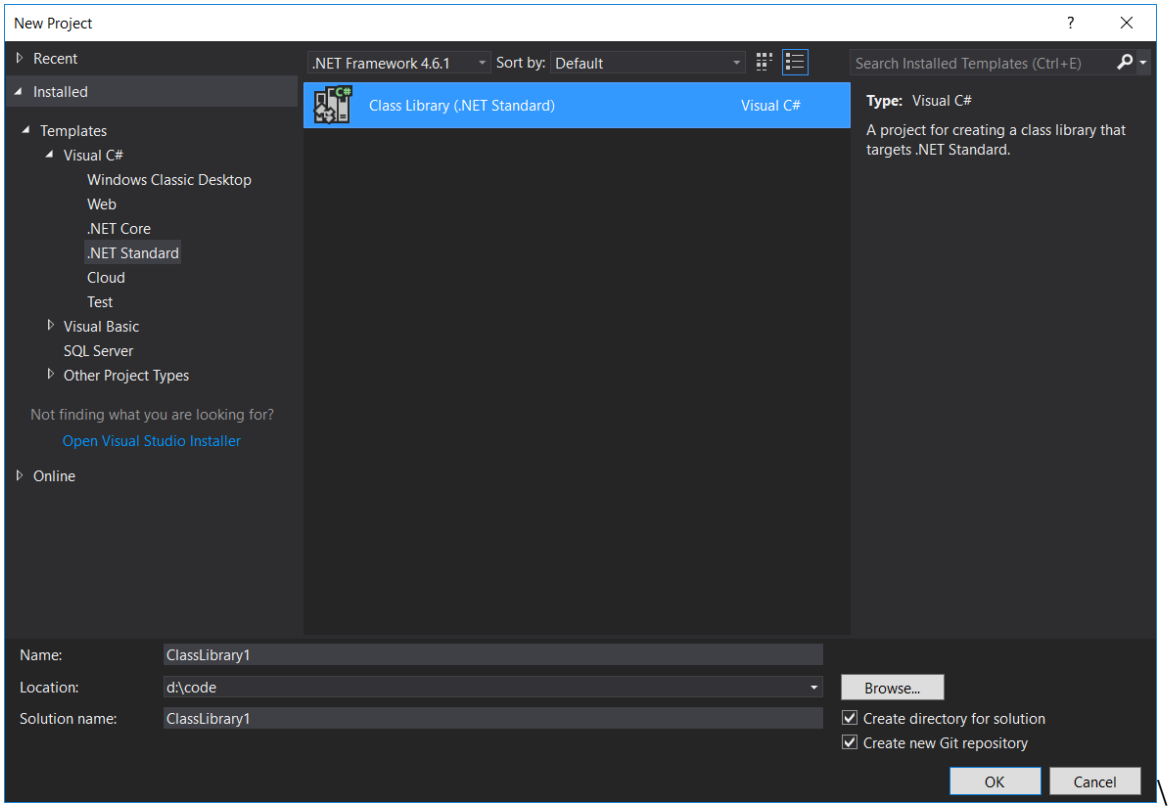
Приклад виконання роботи

Для виконання даної роботи створимо веб додаток на ASP .NET core з типом Empty.



При розробці додатку будемо використовувати архітектуру проектування, основану на шарах. Такий підхід дозволяє легко розширювати, тестувати, та не прив'язуватись до конкретних реалізацій сервісів, або роботою з базами даних. Ця архітектура добре показана на прикладі у цьому відео: https://www.youtube.com/watch?v=5OtUm1BLmG0. В цьому проекті я покажу лише маленьку частину тих переваг, які надає ця архітектура.

Отже для початку нам потрібно описати моделі даних для нашого варіанту. Для цього створимо бібліотеку класів .NET Standart і назвемо її Domain.



В цьому шарі ми будемо зберігати наші моделі баз даних. Тому оголосимо два класи для країн і для міст.

```
namespace RestfulService.Domain.Entities
{
    public class Country
    {
        public Country()
        {
            Cities = new List<City>();
        }
        public int Id { get; set; }
        public string Name { get; set; }
        public List<City> Cities { get; set; }
    }
}
```

```
namespace RestfulService.Domain.Entities
{
    public class City
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public bool isCapital { get; set; }
        public int Population { get; set; }

        public int CountryId { get; set; }
        public Country Country { get; set; }

    }
}
```

Після опису моделей даних, нам потрібно обрати, сховище даних для них. Ми будемо використовувати MS SQL Server. А для роботи з ним, ми будемо використовувати ORM-технологію Entity Framework. Перевагою цього фреймворку є те, що ми можемо абстрагуватися від структури конкретної бази даних і вести операції через моделі.

Щоб взаємодіяти з базою даних нам потрібно визначити контекст даних. Entity Framework використовує підхід Code First, при якому нам потрібно визначити модель та контекст даних, а вже на їх основі буде автоматично створюватись база даних з усіма необхідними таблицями. Тому для початку створимо новий шар Application також типу бібліотеки класів .NET Standart та визначимо там інтерфейс нашого контексту даних.

```
public interface IApplicationDbContext
{
    Ссылок: 6
    DbSet<City> Cities { get; set; }
    Ссылок: 6
    DbSet<Country> Countries { get; set; }
    Ссылок: 6
    Task<int> SaveChangesAsync(CancellationToken cancellationToken);

}
```

А для конкретних реалізації створимо новий шар Infrastructure по типу бібліотека класів .NET Core. І реалізуємо в ньому наш інтерфейс:

```
public class ApplicationDbContext: DbContext, IApplicationDbContext
{
    Ссылок: 0
    public DbSet<City> Cities { get; set; }

    Ссылок: 2
    public DbSet<Country> Countries { get; set; }

    Ссылок: 0
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :base(options)
    {
        Database.EnsureCreated();
    }

    Ссылок: 0
    public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = new CancellationToken())
    {
        return base.SaveChangesAsync(cancellationToken);
    }
}
```

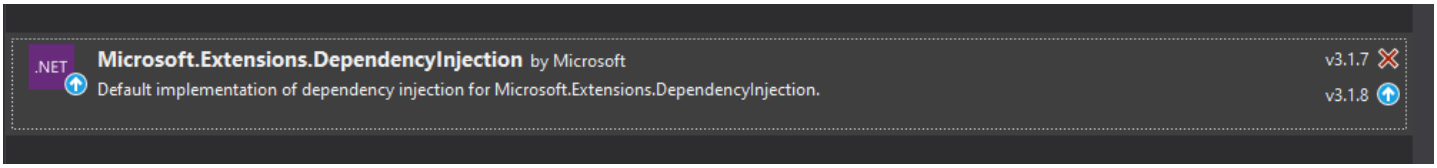
Для того, щоб створити контекст даних, нам потрібно створити клас, який буде наслідувати DbContext. І властивості по типу DbSet<City> фактично будуть допомагати нам працювати з набором даних по певному типу. Та також метод SaveChangesAsync, який дозволить нам зберігати всі зміни у базі даних. Щоб підключитися до бази даних, нам потрібно додати параметри підключення, для цього в файлі appsettings.json, який знаходиться в нашому веб додатку, додамо рядок.

```
"ConnectionStrings": {
  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=countriesdb;Trusted_Connection=True;MultipleActiveResultSets=true"
}
```

Для того, щоб в нашій базі даних, вже були деякі дані при запуску додатку, нам потрібно деякий клас, який буде виступати у ролі ініціалізатора даних. Для цього в шарі Infrastructure створимо такий клас.

```
namespace RestfulService.Infrastructure.Persistance
{
    Ссылок: 0
    public static class AppdlicationDbContextSeed
    {
        Ссылок: 0
        public static  void SeedSampleDataAsync(ApplicationDbContext context)
        {
            if (context.Countries.Count() == 0)
            {
                context.Countries.Add(new Country
                {
                    Name = "Ukraine",
                    Cities = new System.Collections.Generic.List<City>
                    {
                        new City { Name = "Kyiv", isCapital= true, Population= 1337},
                        new City { Name = "Rivne", isCapital= false, Population= 10000}
                    }
                });

                context.SaveChanges();
            }
        }
    }
}
```

Для того, щоб це все працювало, нам потрібно зареєструвати сервіси в класі Startup метода ConfigureServices. Але ж ми користуємося іншою архітектуою, а тому в шарі Infrastructure, додамо пакет DependencyInjection.



І створимо в ньому клас з методом розширення для інтерфейсу IServiceCollection. Де і зареєструємо необхідні для нас сервіси. За допомогою цього методу розширення ми можемо додавати наш контекст даних через конструктор, в потрібні нам класи.

```
public static class DependencyInjection
{
    Ссылок: 0
    public static IServiceCollection AddInfrastructure(this IServiceCollection services, IConfiguration configuration)
    {
        string connection = configuration.GetConnectionString("DefaultConnection");
        services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(connection));

        services.AddScoped<IApplicationDbContext>(provider => provider.GetService<ApplicationDbContext>());

        return services;
    }
}
```

В нашому веб додатку нам потрібно підключити залежність до цього шару. І вже в класі Startup зареєструвати наші сервіси

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddInfrastructure(_configuration);
    services.AddMediatR(typeof(GetCountriesQuery));
}
```

Також в класі Program нам потрібно ініціалізувати наші початкові дані. Таким чином при першому запуску нашої програми буде створена база даних, і в неї буде додано одну країну з двома містами.

```
public static void Main(string[] args)
{
    var webHost = CreateWebHostBuilder(args).Build();

    using (var scope = webHost.Services.CreateScope())
    {
        var services = scope.ServiceProvider;
        try
        {
            var context = services.GetRequiredService<ApplicationDbContext>();
            if(context.Database.IsSqlServer())
            {
                context.Database.Migrate();
            }

             AppdlicationDbContextSeed.SeedSampleDataAsync(context);
        }
        catch(Exception e)
        {
            throw;
        }
    }

     webHost.Run();
}
```

Все вже майже готово, отже тепер в нас є база даних і нам залишилося лише реалізувати роботу з нею. Для цього будемо користуватися зв’язкою патернів CQRS + Mediator. Якщо коротко, то SQRS патерн дозволяє розділити твої команди для

створення і редагування, тобто команд, він читання, тобто запитів. Кожен з яких повертає свою власну модель даних, і виконує свою функцію, таким чином в нас є чисте розділення функцій, для кожного екшена. А патерн Mediator сприяє слабозв'язності, цей шаблон реалізує об'єкт посередник, в якому інші об'єкти взаємодіють з ним, а не один з одним, таким чином у нас виходить чиста і крута архітектура, яка легко тестується і розширюється, а якщо ви захочете поміняти реалізацію якогось методу, ви з легкістю зможете це зробити.

Отже для початку нам потрібно встановити пакет MediatoR через менедер пакетів NuGet. І зареєструвати його в класі Startup. Для демонстрації роботи сервісу ми будемо використовувати стандартний патерн MVC, відмінністю буде лише те, що в нас не буде моделей в самому додатку, адже ми їх об'явили вже раніше в іншому шарі. І в нашого додатку є посилання на цей проект. Спочатку нам потрібно об'явити базовий контролер, в якому ми і зареєструємо медіатор.

```
public class ControlerBase :Controller
{
    private IMediator _mediator;
    Ссылок: 14
    protected IMediator Mediator => _mediator ?? (_mediator = HttpContext.RequestServices.GetService<IMediator>() );
}
```

Далі, в шарі Application реалізуємо наші запити і команди до баз даних, передавши контекст даних в них, за допомогою того ж Dependency Injection.

Спочатку розглянемо роботу з моделлю для країн.

Ось діставання усіх країн з бази даних. Передаємо через конструктор класу зареєстрований еконтекст даних. І за допомогою стандартниого методу Entity Framework дістаємо всі країни і приводимо їх до списку.

```
namespace RestfulService.Application.Countries
{
    Ссылок: 2
    public class GetCountriesQuery : IRequest<List<Country>>
    {
    }

    ссылка: 1
    public class GetCountriesQueryHandler : IRequestHandler<GetCountriesQuery, List<Country>>
    {
        private readonly IApplicationDbContext _applicationDbContext;

        Ссылок: 0
        public GetCountriesQueryHandler(IApplicationDbContext applicationDbContext)
        {
            _applicationDbContext = applicationDbContext;
        }
        Ссылок: 9
        public async Task<List<Country>> Handle(GetCountriesQuery request, CancellationToken cancellationToken)
        {
            return await _applicationDbContext.Countries.ToListAsync();
        }
    }
}
```

Ось діставання країни по її ідентифікатору. Для витягування потрібного запису з таблиці, можемо використовувати метод FirstOrDefault. Він приймає делегат або лямбда-вираз, який являє собою умову, по якій буде вибиратись записи. Поверне перший запис, який задовольняє умові лямбда-виразу, або, якщо таких не було, значення null.

```
namespace RestfulService.Application.Countries
{

    public class GetCountryByIdQuery: IRequest<Country>
    {

        public int ListId { get; set; }
    }


    public class GetCountryByIdQueryHandler : IRequestHandler<GetCountryByIdQuery, Country>
    {
        private readonly IApplicationDbContext _context;

        public GetCountryByIdQueryHandler(IApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<Country> Handle(GetCountryByIdQuery request, CancellationToken cancellationToken)
        {
            return await _context.Countries.Include(x=>x.Cities).FirstOrDefaultAsync(country=>country.Id == request.ListId);
        }
    }
}
```

Тепер розглянемо команди, тобто операції в яких проходять зміни в базі даних.

Додавання країни: створюємо новий екземпляр країни, за допомогоє контексту даних додаємо його в колекцію країн, та зберігаємо контекст даних за допомогою вище описаного методу SaveChangesAsync.

```
namespace RestfulService.Application.Countries.Commands
{
    Ссылок: 2
    public class CreateCountryCommand: IRequest<int>
    {
        ссылка: 1
        public string Name { get; set; }
    }

    ссылка: 1
    public class CrateCountryCommandHandler : IRequestHandler<CreateCountryCommand, int>
    {
        private readonly IApplicationDbContext _context;

        Ссылок: 0
        public CrateCountryCommandHandler(IApplicationDbContext context)
        {
            _context = context;
        }
        Ссылок: 9
        public async Task<int> Handle(CreateCountryCommand request, CancellationToken cancellationToken)
        {
            var entity = new Country();
            entity.Name = request.Name;
            _context.Countries.Add(entity);
            await _context.SaveChangesAsync(cancellationToken);
            return entity.Id;
        }
    }
}
```

Редагування країни: для редагування, нам потрібно спочатку знайти потрібну нам країну, це можна зробити за допомогою методу FindAsync. Якщо такий запис знайдений, то оновлюємо значення полей і зберігаємо зміни.

```
namespace RestfulService.Application.Countries.Commands
{
    public class EditCountryCommand:IRequest
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }

    public class EditCountryCommandHandler : IRequestHandler<EditCountryCommand>
    {
        private readonly IApplicationDbContext _context;

        public EditCountryCommandHandler(IApplicationDbContext context)
        {
            _context = context;
        }
        public async Task<Unit> Handle(EditCountryCommand request, CancellationToken cancellationToken)
        {
            var entity = await _context.Countries.FindAsync(request.Id);

            if(entity!= null)
            {
                entity.Name = request.Name;

                await _context.SaveChangesAsync(cancellationToken);

            }

            return Unit.Value;
        }
    }
}
```

Та видалення країни: для виделення, спочатку знаходимо по ідентифікатору нашу країну, видаляємо її з колекції за допомогою метода Remove, та ззберігаємо зміни.

```
namespace RestfulService.Application.Countries.Commands
{

    public class DeleteCountryCommand:IRequest
    {

        public int Id { get; set; }
    }


    public class DeleteCountryCommandHandler : IRequestHandler<DeleteCountryCommand>
    {
        private readonly IApplicationDbContext _context;


        public DeleteCountryCommandHandler(IApplicationDbContext context)
        {
            _context = context;
        }
        Ссылок: 9
        public async Task<Unit> Handle(DeleteCountryCommand request, CancellationToken cancellationToken)
        {
            var entity = await _context.Countries.FindAsync(request.Id);

            if(entity!=null)
            {
                _context.Countries.Remove(entity);
                await _context.SaveChangesAsync(cancellationToken);
            }
            return Unit.Value;
        }
    }
}
```

Аналогічним чином реалізовується робота з містами.

Далі необхідно реалізувати контроллери, які використовуєчи наш патерн посередник, будуть працювати з командами та запитами.

Контроллер для міст буде мати такий вигляд: тобто в кожному екшені ми за доппомогою посередника відправляємо потрібний нам запит або команду і по суті викликаємо код, який ми вже реалізували вище. Замітьте, що в нашому контролері немає ніяких сервісів, всі вони за необхідністю підключаються в зщапитах, або командах. Результатом надсилання запиту від посередника, буде той тип даних, який повертає запит чи команда. Наприклад в методі List повернеться список усіх міст з бизи даних, який ми передамо в View. Аналогычно в ынших методах, ми передаємо ті значення, які необхідні в наших запитах. Також нам необхідні по дві пари методів для редагування, та створення. Вони відрізняються тільки своїм типом: один з них GET, а другий POST. GET запит відповідає за відображення сторінки з створеннням нового місста, або редагування існуючого, а POST для внесення змін в базу даних.

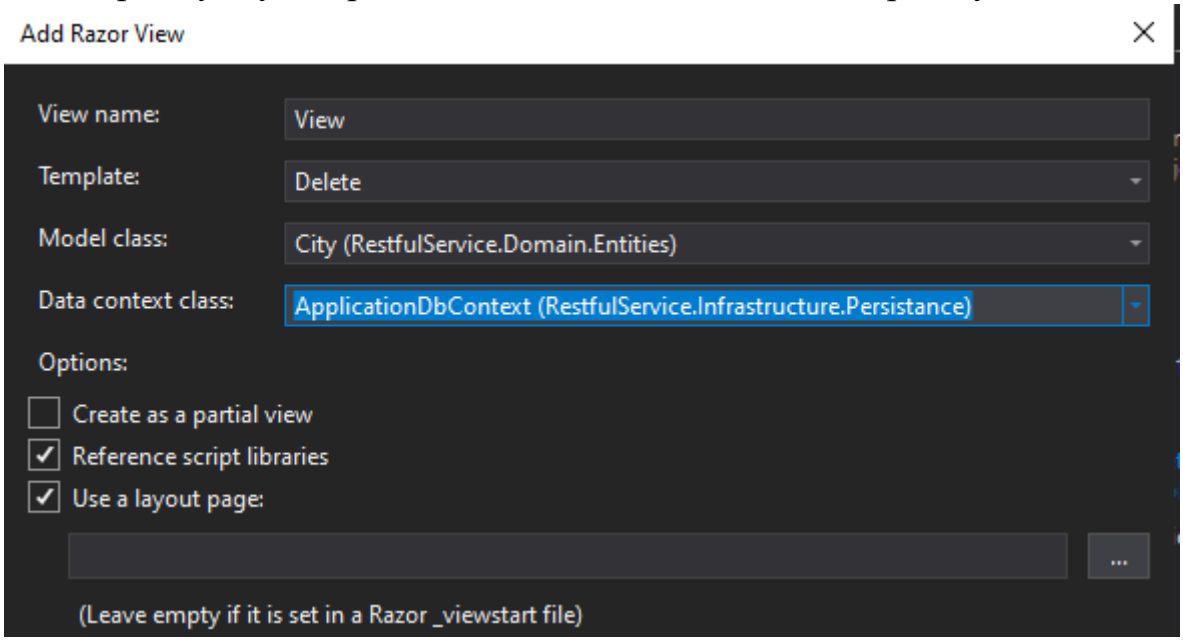
```
namespace RestfulService.Controlers
{
    Ссылок: 0
    public class CitiesController : ControlerBase
    {
        Ссылок: 0
        public async Task<IActionResult> List()
        {
            return View(await Mediator.Send(new ListCitiesQuery()));
        }

        [HttpGet]
        Ссылок: 0
        public async Task<IActionResult> Create()
        {
            var Countries = await Mediator.Send(new GetCountriesQuery());

            SelectList countries = new SelectList(Countries, "Id", "Name");

            ViewBag.countries = countries;
            return View();
        }

        [HttpPost]
        Ссылок: 0
        public async Task<IActionResult> Create(City city)
        {
            var id = await Mediator.Send(new CreateCityCommand { Name = city.Name, CountryId = city.CountryId, IsCapital = city.isCapital, Population = city.Population });
            return Redirect("/Cities/List/");
        }

        Ссылок: 0
        public async Task<IActionResult> Delete(int? id)
        {
            await Mediator.Send(new DeleteCityCommand { Id = id.Value });
            return Redirect("/Cities/List/");
        }

        [HttpGet]
        Ссылок: 0
        public async Task<IActionResult> Edit(int? id)
        {
            var Countries = await Mediator.Send(new GetCountriesQuery());

            SelectList countries = new SelectList(Countries, "Id", "Name");

            ViewBag.countries = countries;
            return View(await Mediator.Send(new GetCityByIdQuery { Id = id.Value }));
        }

        [HttpPost]
        Ссылок: 0
        public async Task<IActionResult> Edit(City city)
        {
            await Mediator.Send(new EditCityCommand { Id = city.Id, Name = city.Name, CountryId = city.CountryId, IsCapital = city.isCapital, Population = city.Population });
            return Redirect("/Cities/List/");
        }
        Ссылок: 0
        public async Task<IActionResult> Details(int?id)
        {

            return View(await Mediator.Send(new GetCityByIdQuery { Id = id.Value }));
        }
    }
}
```

А для країн такий, тобто все аналогічно, тільки працюємо з іншим типом даних і іншими запитами та командами, але логіка залишається незмінною.

```
namespace RestfulService.Controlers
{
    ссылка: 1
    public class CountriesController : ControlerBase
    {
        private readonly ApplicationDbContext _context;

        Ссылок: 0
        public CountriesController(ApplicationDbContext context)
        {
            _context = context;
        }
        Ссылок: 0
        public async Task<IActionResult> List()
        {
            return View(await Mediator.Send(new GetCountriesQuery()));
        }

        Ссылок: 0
        public async Task<IActionResult> Details(int? id)
        {
            return View(await Mediator.Send(new GetCountryByIdQuery { ListId = id.Value }));
        }

        [HttpGet]
        Ссылок: 0
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        Ссылок: 0
        public async Task<IActionResult> Create(Country country)
        {
            var id = await Mediator.Send(new CreateCountryCommand { Name = country.Name });

            return Redirect("/Countries/List/");
        }

        [HttpGet]
        Ссылок: 0
        public async Task<IActionResult> Edit(int? id)
        {
            return View(await Mediator.Send(new GetCountryByIdQuery { ListId = id.Value }));
        }

        [HttpPost]
        Ссылок: 0
        public async Task<IActionResult> Edit(Country country)
        {
            await Mediator.Send(new EditCountryCommand { Id = country.Id, Name = country.Name });
            return Redirect("/Countries/List");
        }
        Ссылок: 0
        public async Task<IActionResult> Delete(int? id)
        {
            await Mediator.Send(new DeleteCountryCommand { Id = id.Value });
            return Redirect("/Countries/List");
        }
    }
}
```

Наш сервіс майже готовий, залишилося створити найпростіші представлення для наших контролерів і все готово. Я наведу лише декілька прикладів, адже всі представлення представлення дуже прості і їх можна згенерувати автоматично, за допомогою вбудованого в Visual Studio функціоналу: вибираємо назву нашого представлення, шаблон(створення, видалення, редагування або список) та контекст

даних, решту студія зробить за вас. Далі вже можна відредагувати на свій смак.



Представлення, для показу всіх країн буде мати такий вигляд: тобто ми отримуємо список всіх країн, як модель і за допомогою циклу foreach відображаємо всі країни.

```
@model IEnumerable<RestfulService.Domain.Entities.Country>

@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<p>
    @Html.ActionLink("Create new", "Create", "Countries")
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Name)
                </td>
                <td>
                    @Html.ActionLink("Edit", "Edit", "Countries", new { id = item.Id }, null)|
                    @Html.ActionLink("Details", "Details", "Countries", new { id = item.Id }, null)|
                    @Html.ActionLink("Delete", "Delete", "Countries", new { id = item.Id }, null)
                </td>
            </tr>
        }
    </tbody>
</table>

@Html.ActionLink("Back to home", "Index", "Home")
```

А представлення, для додавання нового міста такий: робимо поля для вводу необхідних для нас полей. І також кнопку submit, за допомогою якої буде відправлено POST запит в наш контролер.

```
@model RestfulService.Domain.Entities.City

@{
    ViewData["Title"] = "Create";
}

<h1>Create</h1>

<h4>Country</h4>
<hr />
@using (Html.BeginForm())
{
    <p>
        City name <br />
        @Html.EditorFor(model => model.Name)
    </p>
    <p>
        City population <br />
        @Html.EditorFor(model => model.Population)
    </p>
    <p>
        Is capital <br />
        @Html.EditorFor(model => model.isCapital)
    </p>

    <p>
        Country <br />
        @Html.DropDownListFor(model => model.CountryId, ViewBag.countries as SelectList)
    </p>

    <p>
        <input type="submit" value="Add new city" />
    </p>
}

<div>

    @Html.ActionLink("Back to countries list", "List")
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Створивши всі необхідні представлення, можна сміливо запускати додаток, та тестувати його. А саму базу даних, можна подивитися через Оглядач серверів, який є в Visual Studio.