

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 <i>стран</i> . Для каждой <i>страны</i> определено множество <i>городов</i> .
Требуемые операции	Выдача полного списка городов с указанием названия страны

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

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

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

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

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

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

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

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

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

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

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

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

Вариант 14	
Предметная область	Видеомагазин
Объекты	Жанры, Фильмы
Примечание	Имеется множество <i>жанров</i> . Для каждого <i>жанра</i> определен перечень <i>фильмов</i> .
Требуемые операции	Выдача списка фильмов с указанием жанра

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

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

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

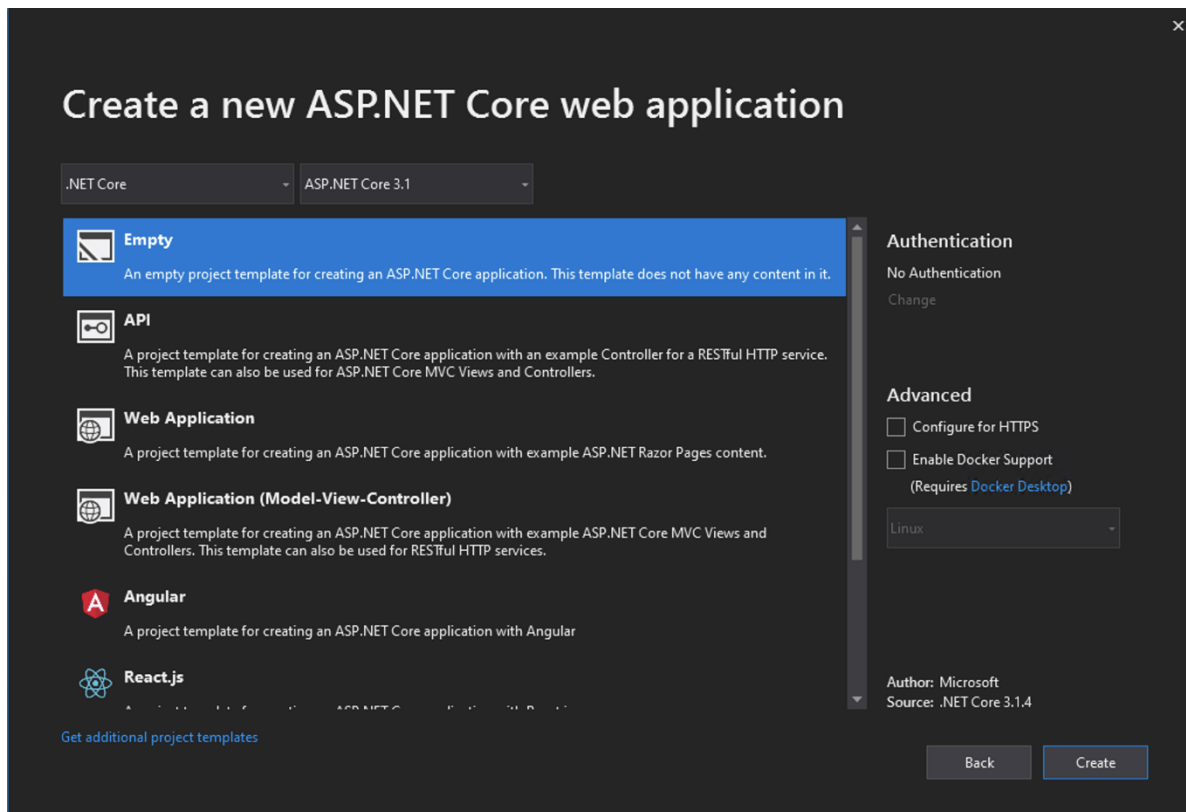
Вариант 18	
Предметная область	Программное обеспечение
Объекты	Производители, Программные продукты
Примечание	Программные <i>продукты</i> сгруппированы по <i>производителям</i> . Для каждого <i>производителя</i> задано множество <i>продуктов</i> .
Требуемые операции	Выдача списка продуктов с указанием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

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

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

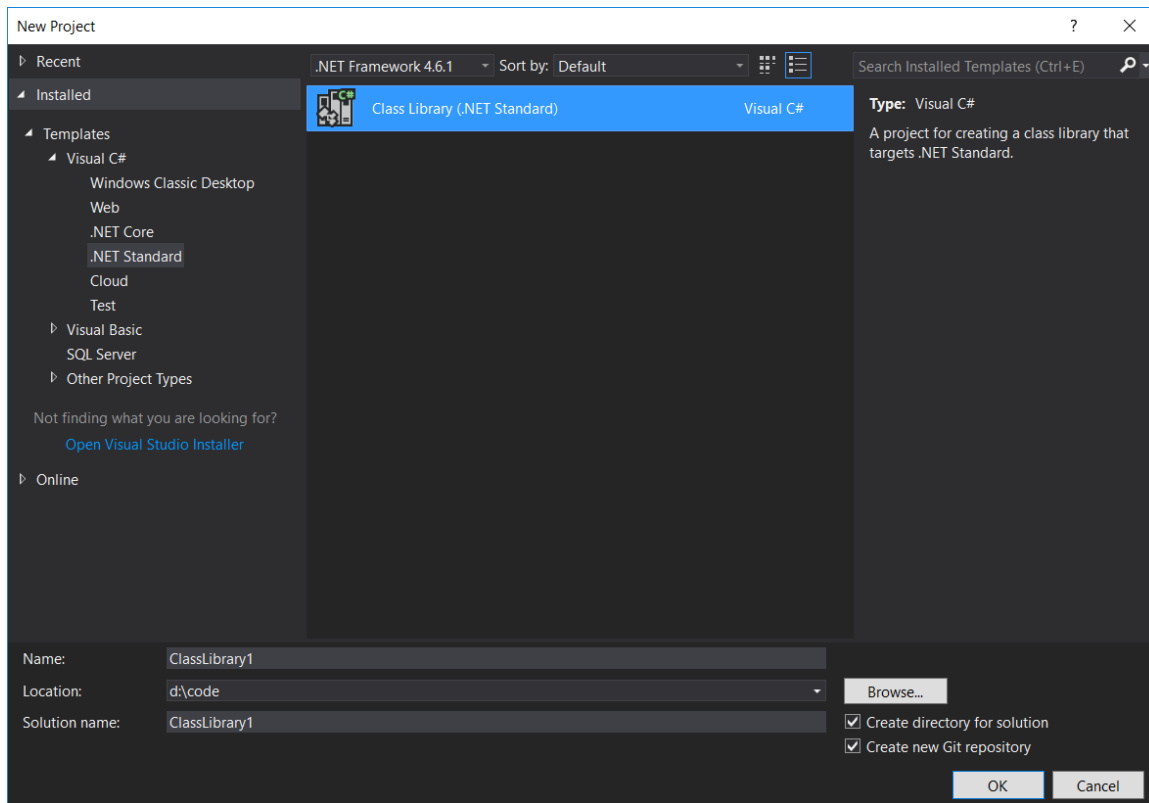
Приклад виконання роботи

Спочатку створимо веб додаток на ASP .NET core з типом Empty.



При розробці додатку будемо використовувати архітектуру проектування, основу 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.

```
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 Standard та визначимо там інтерфейс нашого контексту даних.

```
public interface IApplicationDbContext
{
    Ссылка: 6
    DbSet<City> Cities { get; set; }
    Ссылка: 6
    DbSet<Country> Countries { get; set; }
    Ссылка: 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
{
    Ссылка: 0
    public DbSet<City> Cities { get; set; }

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

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

    Ссылка: 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. 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
{
    Ссылка: 0
    public static class AppdlicationDbContextSeed
    {
        Ссылка: 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 архітектурою, 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.

```
public static class DependencyInjection
{
    Ссылка: 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 додати наші сервіси

```
Ссылка: 0
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddInfrastructure(_configuration);
    services.AddMediatR(typeof(GetCountriesQuery));
}
```

Також в класі Program нам потрібно ініціалізувати наші початкові дані.

```
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();
            }

            ApplicationDbContextSeed.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 іншим, 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 менеджер 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 ControllerBase : Controller
{
    private IMediator _mediator;
    Ссылка: 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.

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

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

        //ссылка: 0
        public GetCountriesQueryHandler(IApplicationDbContext applicationDbContext)
        {
            _applicationDbContext = applicationDbContext;
        }

        //ссылка: 9
        public async Task<List<Country>> Handle(GetCountriesQuery request, CancellationToken cancellationToken)
        {
            return await _applicationDbContext.Countries.ToListAsync();
        }
    }
}
```

Ось діставання країни по її ідентифікатору.

```
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:

```

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

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

        ССЫЛКА: 0
        public CreateCountryCommandHandler(IApplicationDbContext context)
        {
            _context = context;
        }

        ССЫЛКА: 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:

```

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:

```
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;
        }

        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 контролери, які використовуючи наш патерн посередник, будуть працювати з командами та запитами.

Контроллер для міст буде мати такий вигляд:

```
namespace RestfulService.Controllers
{
    [Authorize]
    public class CitiesController : ControllerBase
    {
        [Authorize]
        public async Task<IActionResult> List()
        {
            return View(await Mediator.Send(new ListCitiesQuery()));
        }

        [HttpGet]
        [Authorize]
        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]
        [Authorize]
        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/");
        }

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

        [HttpGet]
        [Authorize]
        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]
        [Authorize]
        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/");
        }

        [Authorize]
        public async Task<IActionResult> Details(int?id)
        {
            return View(await Mediator.Send(new GetCityByIdQuery { Id = id.Value }));
        }
    }
}
```

А для країн такий:

```
namespace RestfulService.Controllers
{
    Ссылка: 1
    public class CountriesController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

        Ссылка: 0
        public CountriesController(ApplicationDbContext context)
        {
            _context = context;
        }

        Ссылка: 0
        public async Task<IActionResult> List()
        {
            return View(await Mediator.Send(new GetCountriesQuery()));
        }

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

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

        [HttpPost]
        Ссылка: 0
        public async Task<IActionResult> Create(Country country)
        {
            var id = await Mediator.Send(new CreateCountryCommand { Name = country.Name });
            return Redirect("/Countries/List/");
        }

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

        [HttpPost]
        Ссылка: 0
        public async Task<IActionResult> Edit(Country country)
        {
            await Mediator.Send(new EditCountryCommand { Id = country.Id, Name = country.Name });
            return Redirect("/Countries/List/");
        }

        Ссылка: 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.

Представлення, для показу всіх країн буде мати такий вигляд:

```
@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:

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

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

<h1>Create</h1>
<h4>Country</h4>
<hr />
<div>
    <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>
<div>
    @Html.ActionLink("Back to countries list", "List")
</div>
<section Scripts >
    @await Html.RenderPartialAsync("_ValidationScriptsPartial");
</section>
```


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.