Лабораторна робота 3. Мікросервіси Orders та Customers. Створення класів та інтерфейсів, необхідних для функціонування.

Мета. Вивчити процес створення проектів в середовищі Visual Stidio 2019, освоїти побудову додатків ASP.NET API, отримати реальні практичні навички розробки мікросервісів.

Завдання: Розробити основу мікросервісів Orders та Customers. Створити класи і інтерфейси, необхідні для функціонування цих сервісів.

Порядок виконання

Ми повинні розробити наступні класи і інтерфейси.

Customers	Orders
CustomersDbContext	OrdersDbContext
Customer (entity and model)	Order (entity and model)
ICustomersProvider	OrderItem (entity and model)
CustomersProvider	IOrdersProvider
CustomersController	OrdersProvider
CustomerProfile	OrdersController
	OrderProfile

Рис. 1. Класи і інтерфейси, необхідні для мікросервісів Customers і Orders.

Ось сутності, необхідні нашим мікросервісам.

Customer	Order	OrderItem
Id	Id	Id
Name	CustomerId	OrderId
Address	OrderDate	ProductId
	Total	Quantity
	Items[]	UnitPrice

Рис. 2. Сутності.

Пам'ятайте також, що провайдери повертають класи моделей а не сутностей, тому ми повинні реалізувати ці класи по-різному.

Наприклад, клас моделі OrderItem не потребує властивості order ID.

Розглянемо реалізацію мікросервісів Customers і Orders. Зауважте, що мікросервіс Customers багато в чому подібний до мікросервіса Products.

Відкрийте проект ECommerce.Api.Customers в Solution Explorer. Зробіть цей проект стартовим.

По-перше, треба створити customer DB context клас (в папці Db).

```
namespace ECommerce.Api.Customers.Db
{
  public class CustomersDbContext : DbContext
  {
    public DbSet<Customer> Customers { get; set; }

 public CustomersDbContext(DbContextOptions options) : base(options)
        {
        }
    }
}
```

По-друге, треба створити клас сутності Customer (в папці Db).

```
namespace ECommerce.Api.Customers.Db
{
    public class Customer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
    }
}
```

Далі в папці Interfaces створіть інтерфейс ICustomersProvider.

```
namespace ECommerce.Api.Customers.Interfaces
{
    public interface ICustomersProvider
    {
        Task<(bool IsSuccess, IEnumerable<Customer> Customers,
string ErrorMessage)> GetCustomersAsync();
        Task<(bool IsSuccess, Customer Customer, string
ErrorMessage)> GetCustomerAsync(int id);
    }
}
```

Далі в папці Providers створіть клас CustomersProvider, що реалізує інтерфейс ICustomersProvider.

```
namespace ECommerce.Api.Customers.Providers
{
    public class CustomersProvider : ICustomersProvider
    {
        private readonly CustomersDbContext dbContext;
        private readonly ILogger<CustomersProvider> logger;
        private readonly IMapper mapper;

        public CustomersProvider(CustomersDbContext dbContext,
ILogger<CustomersProvider> logger, IMapper mapper)
        {
            this.dbContext = dbContext;
            this.logger = logger;
            this.mapper = mapper;
```

```
                SeedData();
            }

            private void SeedData()
            {
                if (!dbContext.Customers.Any())
                {
                    dbContext.Customers.Add(new Db.Customer() { Id = 1,
Name = "Jessica Smith", Address = "20 Elm St." });
                    dbContext.Customers.Add(new Db.Customer() { Id = 2,
Name = "John Smith", Address = "30 Main St." });
                    dbContext.Customers.Add(new Db.Customer() { Id = 3,
Name = "William Johnson", Address = "100 10th St." });
                    dbContext.SaveChanges();
                }
            }

            public         async         Task<(bool         IsSuccess,
IEnumerable<Models.Customer>     Customers,     string     ErrorMessage)>
GetCustomersAsync()
            {
                try
                {
                    logger?.LogInformation("Querying customers");
                    var           customers           =           await
dbContext.Customers.ToListAsync();
                    if (customers != null && customers.Any())
                    {
                        logger?.LogInformation($"{customers.Count}
customer(s) found");
                        var                   result                  =
mapper.Map<IEnumerable<Db.Customer>,
IEnumerable<Models.Customer>>(customers);
                        return (true, result, null);
                    }
                    return (false, null, "Not found");
                }
                catch (Exception ex)
                {
                    logger?.LogError(ex.ToString());
                    return (false, null, ex.Message);
                }
            }

            public async Task<(bool IsSuccess, Models.Customer Customer,
string ErrorMessage)> GetCustomerAsync(int id)
            {
                try
                {
                    logger?.LogInformation("Querying customers");
                    var           customer           =           await
dbContext.Customers.FirstOrDefaultAsync(c => c.Id == id);
                    if (customer != null)
                    {
                        logger?.LogInformation("Customer found");
                        var     result     =     mapper.Map<Db.Customer,
Models.Customer>(customer);
                        return (true, result, null);
                    }
```

```
                return (false, null, "Not found");
            }
            catch (Exception ex)
            {
                logger?.LogError(ex.ToString());
                return (false, null, ex.Message);
            }
        }
    }
}
```

Далі створіть папку Profiles і в ній створіть клас CustomerProfile.

```
namespace ECommerce.Api.Customers.Profiles
{
    public class CustomerProfile : AutoMapper.Profile
    {
        public CustomerProfile()
        {
            CreateMap<Db.Customer, Models.Customer>();
        }
    }
}
```

Далі створіть папку Models і в ній створіть клас Customer.

```
namespace ECommerce.Api.Customers.Models
{
    public class Customer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
    }
}
```

Далі створіть папку Controllers і в ній створіть клас CustomersController.

```
namespace ECommerce.Api.Customers.Controllers
{
    [ApiController]
    [Route("api/customers")]
    public class CustomersController : ControllerBase
    {
      private readonly ICustomersProvider customersProvider;

   public CustomersController(ICustomersProvider customersProvider)
        {
            this.customersProvider = customersProvider;
        }

        [HttpGet]
        public async Task<IActionResult> GetCustomersAsync()
        {
         var result = await customersProvider.GetCustomersAsync();
            if (result.IsSuccess)
            {
                return Ok(result.Customers);
            }
            return NotFound();
        }

        [HttpGet("{id}")]
        public async Task<IActionResult> GetCustomerAsync(int id)
```

```
            {
            var result = await customersProvider.GetCustomerAsync(id);
                if (result.IsSuccess)
                {
                    return Ok(result.Customer);
                }
                return NotFound();
            }
        }
    }
```

Один з методів дії цього контролера повертає всіх користувачів, а другий метод – одного користувача по його ID.

Далі модифікуйте клас Startup.

```
    public void ConfigureServices(IServiceCollection services)
            {
                services.AddDbContext<CustomersDbContext>(options =>
                {
                    options.UseInMemoryDatabase("Customers");
                });
                services.AddScoped<ICustomersProvider,
CustomersProvider>();
                services.AddAutoMapper(typeof(Startup));
                services.AddControllers();
            }
```

Далі перевірте роботу мікросервісу, натиснувши кнопку IIS Express. Отримайте всіх користувачів, і одного конкретного користувача.

Далі займіться мікросервісом Orders. Він трохи відрізняється від попередніх через те, що він потребує двох класів сутностей.

Відкрийте проект ECommerce.Api.Orders в Solution Explorer. Зробіть цей проект стартовим.

Тут ми матимемо класи сутностей Order і OrderItem (в папці Db).

```
    namespace ECommerce.Api.Orders.Db
    {
        public class Order
        {
            public int Id { get; set; }
            public int CustomerId { get; set; }
            public DateTime OrderDate { get; set; }
            public decimal Total { get; set; }
            public List<OrderItem> Items { get; set; }
        }
    }

    namespace ECommerce.Api.Orders.Db
    {
        public class OrderItem
        {
            public int Id { get; set; }
            public int OrderId { get; set; }
            public int ProductId { get; set; }
            public int Quantity { get; set; }
            public decimal UnitPrice { get; set; }
        }
    }
```

Додайте також в папку Db клас OrdersDbContext.

```
namespace ECommerce.Api.Orders.Db
{
    public class OrdersDbContext : DbContext
    {
        public DbSet<Order> Orders { get; set; }
        public    OrdersDbContext(DbContextOptions    options)    :
base(options)
        {
        }
    }
}
```

Далі створіть папку Models і в ній створіть класи Order і OrderItem.

```
namespace ECommerce.Api.Orders.Models
{
    public class Order
    {
        public int Id { get; set; }
        public int CustomerId { get; set; }
        public DateTime OrderDate { get; set; }
        public decimal Total { get; set; }
        public List<OrderItem> Items { get; set; }
    }
}

namespace ECommerce.Api.Orders.Models
{
    public class OrderItem
    {
        public int Id { get; set; }
        public int ProductId { get; set; }
        public int Quantity { get; set; }
        public decimal UnitPrice { get; set; }
    }
}
```

Далі створіть папку Interfaces і в ній створіть інтерфейс IOrdersProvider.

```
namespace ECommerce.Api.Orders.Interfaces
{
    public interface IOrdersProvider
    {
        Task<(bool   IsSuccess,   IEnumerable<Models.Order>   Orders,
string ErrorMessage)> GetOrdersAsync(int customerId);
    }
}
```

Тут ми маємо один метод, який повертає всі замовлення конкретного користувача.

Далі в папці Providers створіть клас OrdersProvider, що реалізує інтерфейс IOrdersProvider.

```
using AutoMapper;
using ECommerce.Api.Orders.Db;
using ECommerce.Api.Orders.Interfaces;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace ECommerce.Api.Orders.Providers
{
   public class OrdersProvider : IOrdersProvider
   {
      private readonly OrdersDbContext dbContext;
      private readonly ILogger<OrdersProvider> logger;
      private readonly IMapper mapper;

      public OrdersProvider(OrdersDbContext dbContext,
ILogger<OrdersProvider> logger, IMapper mapper)
      {
         this.dbContext = dbContext;
         this.logger = logger;
         this.mapper = mapper;
         SeedData();
      }

      private void SeedData()
      {
         if (!dbContext.Orders.Any())
         {
            dbContext.Orders.Add(new Order()
            {
               Id = 1,
               CustomerId = 1,
               OrderDate = DateTime.Now,
               Items = new List<OrderItem>()
               {
                  new OrderItem() { OrderId = 1, ProductId = 1, Quantity = 10,
UnitPrice = 10 },
                  new OrderItem() { OrderId = 1, ProductId = 2, Quantity = 10,
UnitPrice = 10 },
                  new OrderItem() { OrderId = 1, ProductId = 3, Quantity = 10,
UnitPrice = 10 },
                  new OrderItem() { OrderId = 2, ProductId = 2, Quantity = 10,
UnitPrice = 10 },
                  new OrderItem() { OrderId = 3, ProductId = 3, Quantity = 1,
UnitPrice = 100 }
               },
               Total = 100
            });
```

```
                dbContext.Orders.Add(new Order()
                {
                    Id = 2,
                    CustomerId = 1,
                    OrderDate = DateTime.Now.AddDays(-1),
                    Items = new List<OrderItem>()
                    {
                        new OrderItem() { OrderId = 1, ProductId = 1, Quantity = 10, UnitPrice = 10 },
                        new OrderItem() { OrderId = 1, ProductId = 2, Quantity = 10, UnitPrice = 10 },
                        new OrderItem() { OrderId = 1, ProductId = 3, Quantity = 10, UnitPrice = 10 },
                        new OrderItem() { OrderId = 2, ProductId = 2, Quantity = 10, UnitPrice = 10 },
                        new OrderItem() { OrderId = 3, ProductId = 3, Quantity = 1, UnitPrice = 100 }
                    },
                    Total = 100
                });
                dbContext.Orders.Add(new Order()
                {
                    Id = 3,
                    CustomerId = 2,
                    OrderDate = DateTime.Now,
                    Items = new List<OrderItem>()
                    {
                        new OrderItem() { OrderId = 1, ProductId = 1, Quantity = 10, UnitPrice = 10 },
                        new OrderItem() { OrderId = 2, ProductId = 2, Quantity = 10, UnitPrice = 10 },
                        new OrderItem() { OrderId = 3, ProductId = 3, Quantity = 1, UnitPrice = 100 }
                    },
                    Total = 100
                });
                dbContext.SaveChanges();
            }
        }

        public async Task<(bool IsSuccess, IEnumerable<Models.Order> Orders, string ErrorMessage)> GetOrdersAsync(int customerId)
        {
            try
            {
```

```
                var orders = await dbContext.Orders
                    .Where(o => o.CustomerId == customerId)
                    .Include(o => o.Items)
                    .ToListAsync();
                if (orders != null && orders.Any())
                {
                    var result = mapper.Map<IEnumerable<Db.Order>,
                        IEnumerable<Models.Order>>(orders);
                    return (true, result, null);
                }
                return (false, null, "Not Found");
            }
            catch (Exception ex)
            {
                logger?.LogError(ex.ToString());
                return (false, null, ex.Message);
            }
        }
    }
}
```

Далі створіть папку Profiles і в ній створіть клас OrderProfile.

```
namespace ECommerce.Api.Orders.Profiles
{
    public class OrderProfile : AutoMapper.Profile
    {
        public OrderProfile()
        {
            CreateMap<Db.Order, Models.Order>();
            CreateMap<Db.OrderItem, Models.OrderItem>();
        }
    }
}
```

Тут ми бачимо два відображення замість одного.

Далі створіть папку Controllers і в ній створіть клас OrdersController.

```
namespace ECommerce.Api.Orders.Controllers
{
    [ApiController]
    [Route("api/orders")]
    public class OrdersController : ControllerBase
    {
        private readonly IOrdersProvider ordersProvider;

        public OrdersController(IOrdersProvider ordersProvider)
        {
            this.ordersProvider = ordersProvider;
        }

        [HttpGet("{customerId}")]
        public async Task<IActionResult> GetOrdersAsync(int
customerId)
        {
```

```
                var result = await
ordersProvider.GetOrdersAsync(customerId);
                if (result.IsSuccess)
                {
                    return Ok(result.Orders);
                }
                return NotFound();
            }
        }
    }
```

Метод дії цього контролера `GetOrdersAsync` повертає всі замовлення одного користувача по його ID.

Далі модифікуйте клас Startup.

```
    public void ConfigureServices(IServiceCollection services)
            {
                services.AddDbContext<OrdersDbContext>(options =>
                {
                    options.UseInMemoryDatabase("Orders");
                });
                services.AddScoped<IOrdersProvider,
OrdersProvider>();
                services.AddAutoMapper(typeof(Startup));
                services.AddControllers();
            }
```

Тепер стартуйте цей проект, натиснувши кнопку IIS Express. Далі відредагуйте рядок URL, задавши наприклад в кінці /api/orders/2.

ВИКОРИСТАНІ ДЖЕРЕЛА

Lynda - Azure Microservices with .NET Core for Developers (2020). – URL: https://www.lynda.com/Azure-tutorials/Azure-Microservices-NET-Core-Developers/2825264-2.html