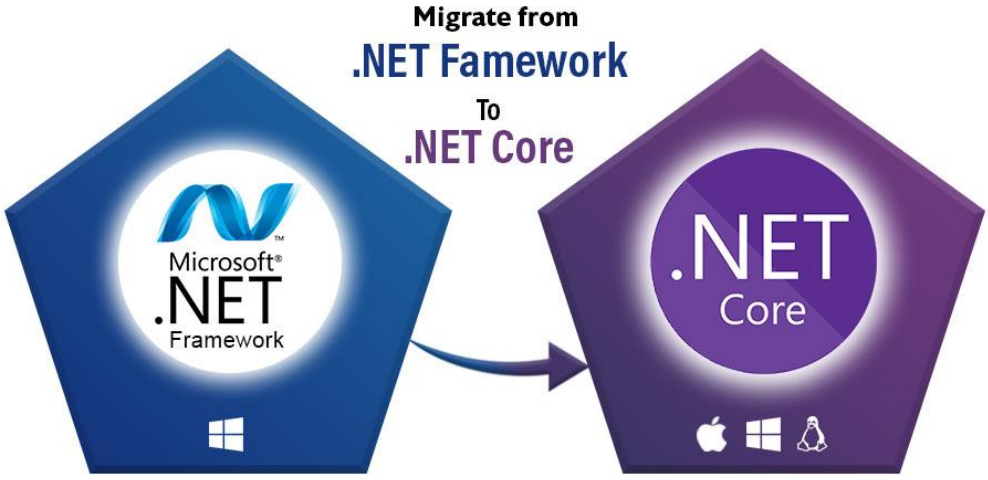

MIGRATION

Documentation

Abstract

In this module I learn how to migrate any .NET framework application in .NET Core

Dhruvil Dobariya
dhruvildobariya21@gmail.com

INDEX

ASP.NET Web API 2 To ASP.NET Core Web API

1.1 Introduction

- We have different steps to migrate ASP.NET Web API 3 to ASP.NET Core Web API project.
  - Create new project
  - Remove unnecessary files
  - Configure ASP.NET Core application based on ASP.NET Web API 2 application
  - Migrate Models
  - Migrate Controllers

1.2 Create new project

- We have different steps to create new project of ASP.NET Core Web API.
- From the File menu, select **New > Project**
- Enter **Web API** in the search box.
- Select the **ASP.NET Core Web API** template and click on Next.
- In the Configure your new project dialog, give name the project like "ProductAPI" and click on Next.
- In the Additional information dialog,
  - Choose the version of .NET (.NET 6 or above)
  - Confirm the checkbox for Use controllers(uncheck to use minimal APIs) is checked
  - Check Enable OpenAPI support (if we want to use of swagger)
  - Click on Create
- Now we have ASP.NET Core Web API project.

1.3 Remove unnecessary files

- Microsoft give build in created file in project to run as example.
- So, first we must delete those file.
- For this, we have different steps,
- Remove the **WeatherForecast.cs** and **Controllers/WeatherForecastController.cs** example files from the new ProductsCore project.
- Change **launchUrl** properties from "weatherforcast" to "product"(if we don't enable OpenAPI).

1.4 Configure ASP.NET Core application based on ASP.NET Web API 2 application

- Now ASP.NET Core Web API project doesn't contains **Global.asax**, **WebConfig.cs** and **Web.config**.
- First we want to migrate all custom configuration from **Web.config** to **appsettings.json** based on our requirement.
- Now, in ASP.NET Core Web API (.NET 6 or above) project all global level event and methods and dependencies are managed and register in **"Program.cs"**.
- So we must configure the **Program.cs** based on **Global.asax** and **WebConfig.cs** and other child config file of **WebConfig.cs**.

1.5 Migrate Models

- If we use any architecture, like 3 tier architecture, N tier architecture, Onion architecture, Clean architecture and other which separate Models and place in other layer then we don't want to migrate models, because we have separate class library for Models.
- So just we detach form framework application and attach it in core application.
- But If our .NET framework project contains folder of Models the we have different steps to migrate it.
- In Solution Explorer, right-click the project. Select **Add > New Folder**. Name the folder Models.
- Now migrate all the models form ASP.NET Web API 2 application to ASP.NET Core Web API application.
- Now change namespace of all the models based on .NET Core application and our requirement.

1.6 Migrate Controller

- First we should move all the controller file from ASP.NET Web API 2 application to ASP.NET Core Web API application.
- Now we should change all the namespace based on our application.
- In ASP.NET Core Web API 2 all the controller inherit by **ApiController** class when in ASP.NET Core Web API all the controller inherit by **ControllerBase** class.
- Second thing if we want to use controller as API controller in ASP.NET Core Web API then we must give attribute **[ApiController]** on controller.
- Third thing is in ASP.NET Web API 2 we configure route using conventional based routing and attribute based routing where in ASP.NET Core Web API we must use attribute based routing for API controllers.

Migration

- Fourth ting is in ASP.NET Web API 2 we use return type IHttpActionResult and HttpResponceMessage where in ASP.NET Core Web API we use IActionResult or ActionResult.

Example:

```
using Microsoft.AspNetCore.Mvc;
using ProductsAPI.Models;

namespace ProductsAPI.Controllers;

[Route("api/[controller]")]
[ApiController]
public class ProductsController : ControllerBase
{
    Product[] products = new Product[]
    {
            new Product
            {
                Id = 1, Name = "Tomato Soup", Category = "Groceries", Price = 1
            },
            new Product
            {
                Id = 2, Name = "Yo-yo", Category = "Toys", Price = 3.75M
            },
            new Product
            {
                Id = 3, Name = "Hammer", Category = "Hardware", Price = 16.99M
            }
    };

    [HttpGet]
    public ActionResult<IEnumerable<Product>> GetAllProducts()
    {
        return Ok(products);
    }

    [HttpGet("{id}")]
    public ActionResult<Product> GetProduct(int id)
    {
        var product = products.FirstOrDefault((p) => p.Id == id);
        if (product == null)
        {
            return NotFound();
        }
        return Ok(product);
    }
}
```

Migration

- Now we have new .NET Core application which we are successfully done migration form ASP.NET Web API 2 to ASP.NET Core Web API.

MIGRATE WHOLE APPLICATION WITH CLASS LIBRARY

2.1 INTRODUCTION

- In ASP.NET Web API 2 we don't have built in feature of dependency injection.
- So we create object when we want to use any thing form BL or DAL.
- This method take lot of memory compare to dependency injection.
- So if we want to migrate whole application from ASP.NET Web API 2 to ASP.NET Core Web API, then we must manage dependency injection.

2.2 HOW WE DO

- We have different steps to migrate whole application.
  - Create new project
  - Remove unnecessary files
  - Configure ASP.NET Core application based on ASP.NET Web API 2 application
  - Migrate Models
- Above things already we se in last chapter.
- Now here we first manage different layers.
  - Migrate DAL
  - Migrate BL
  - Migrate Controllers
  - Register Dependencies

2.3 MIGRATE DAL

- In ASP.NET Web API 2 we use just class based pattern, but in ASP.NET Core Web API we must use interface based pattern so we can easily use dependency injection.

Before Migration:

- We have ASP.NET Web API 2 DAL like,

```
public class DBProductContext
{
    private readonly string _connectionString = "Server = localhost; Database =
ProductDB; Username = Admin; Password=gs@123;";

    #region GetProducts
```

Migration

```
    public List<Product> GetProducts()
    {
        try
        {
            List<Product> products = new List<Product>();
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                products = connection.Query<Product>("Select * From
Product").ToList();
            }
            return products;
        }
        catch { throw; }
    }

    #endregion GetProducts

    #region GetProductById

    public Product GetProductById(int id)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                Product product = connection.QuerySingleOrDefault<Product>("Select *
From Product Where Id = " + id);
                return product;
            }

        }
        catch { throw; }
    }

    #endregion GetProductById

    #region AddProduct

    public int AddProduct(Product product)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                int result = connection.Execute("Insert into Product (Name,
Quantity, Price) values (@Name, @Quantity, @Price)", product);
                return result;
            }
        }
        catch { throw; }
    }
```

Migration

```
    #endregion AddProduct

    #region UpdateProduct

    public int UpdateProduct(Product product)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                int result = connection.Execute("Update Product set Name = @Name,
Quantity = @Quantity, Price = @Price Where Id = @Id", product);
                return result;
            }

        }
        catch { throw; }
    }

    #endregion UpdateProduct

    #region DeleteProduct

    public int DeleteProduct(int id)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                int result = connection.Execute("Delete From Product Where Id = " +
id);
                return result;
            }
        }
        catch { throw; }
    }

        #endregion DeleteProductpublic class DBProductContext
{
    private readonly string _connectionString = "Server = localhost; Database =
ProductDB; Username = Admin; Password=gs@123;";

    #region GetProducts

    public List<Product> GetProducts()
    {
        try
        {
            List<Product> products = new List<Product>();
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
```

Migration

```
                connection.Open();
                products = connection.Query<Product>("Select * From
Product").ToList();
            }
            return products;
        }
        catch { throw; }
    }

    #endregion GetProducts

    #region GetProductById

    public Product GetProductById(int id)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                Product product = connection.QuerySingleOrDefault<Product>("Select *
From Product Where Id = " + id);
                return product;
            }

        }
        catch { throw; }
    }

    #endregion GetProductById

    #region AddProduct

    public int AddProduct(Product product)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                int result = connection.Execute("Insert into Product (Name,
Quantity, Price) values (@Name, @Quantity, @Price)", product);
                return result;
            }
        }
        catch { throw; }
    }

    #endregion AddProduct

    #region UpdateProduct

    public int UpdateProduct(Product product)
    {
        try
```

```
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                int result = connection.Execute("Update Product set Name = @Name,
Quantity = @Quantity, Price = @Price Where Id = @Id", product);
                return result;
            }

        }
        catch { throw; }
    }

    #endregion UpdateProduct

    #region DeleteProduct

    public int DeleteProduct(int id)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                int result = connection.Execute("Delete From Product Where Id = " +
id);
                return result;
            }
        }
        catch { throw; }
    }

    #endregion DeleteProduct
}
```

After Migration:

- Now we should just add one interface “IDBProductContext” in DAL which is implement by “DBProductContext”.

```
public interface IDbProductContext
{
    List<Product> GetProducts();
    Product GetProductById(int id);
    int AddProduct(Product product);
    int UpdateProduct(Product product);
    int DeleteProduct(int id);
}
```

- Now we just implement it in DBProductContext which is already implement we just inherit it.

Migration

```
public class DBProductContext : IDbProductContext
{
    private readonly string _connectionString = "Server = localhost; Database =
ProductDB; Username = Admin; Password=gs@123;";

    #region GetProducts

    public List<Product> GetProducts()
    {
        try
        {
            List<Product> products = new List<Product>();
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                products = connection.Query<Product>("Select * From
Product").ToList();
            }
            return products;
        }
        catch { throw; }
    }

    #endregion GetProducts

    #region GetProductById

    public Product GetProductById(int id)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                Product product = connection.QuerySingleOrDefault<Product>("Select *
From Product Where Id = " + id);
                return product;
            }

        }
        catch { throw; }
    }

    #endregion GetProductById

    #region AddProduct

    public int AddProduct(Product product)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
```

```
                int result = connection.Execute("Insert into Product (Name,
Quantity, Price) values (@Name, @Quantity, @Price)", product);
                return result;
            }
        }
        catch { throw; }
    }

    #endregion AddProduct

    #region UpdateProduct

    public int UpdateProduct(Product product)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                int result = connection.Execute("Update Product set Name = @Name,
Quantity = @Quantity, Price = @Price Where Id = @Id", product);
                return result;
            }

        }
        catch { throw; }
    }

    #endregion UpdateProduct

    #region DeleteProduct

    public int DeleteProduct(int id)
    {
        try
        {
            using (MySqlConnection connection = new
MySqlConnection(_connectionString))
            {
                connection.Open();
                int result = connection.Execute("Delete From Product Where Id = " +
id);
                return result;
            }
        }
        catch { throw; }
    }

    #endregion DeleteProduct
}
```

- In DAL we are using Dapper micro ORM, we can use any other ORM or pure ADO.NET provider.

2.4 MIGRATE BL

- Now we migrate BL, it same as migration of DAL.
- Just create interface and inherit it in already implemented class.
- But the thing is now we don't define object of DAL in every method of BL.
- We just inject dependency of DAL and use in every method of BAL.

Before Migration:

- Let say we have "BLProductHandler" in ASP.NET Web API 2.

```
public class BLProductHandler
{
    #region GetProducts

    public List<Product> GetProducts()
    {
        try
        {
            return new DBProductContext().GetProducts();
        }
        catch
        {
            throw;
        }
    }

    #endregion GetProducts

    #region GetProductById

    public Product GetProductById(int id)
    {
        try
        {
            return new DBProductContext().GetProductById(id);
        }
        catch { throw; }
    }

    #endregion GetProductById

    #region AddProduct

    public bool AddProduct(Product product)
    {
        try
        {
            int result = new DBProductContext().AddProduct(product);

            if (result > 0)
            {
                return true;
            }
```

```
            return false;
        }
        catch { throw; }
    }

    #endregion AddProduct

    #region UpdateProduct

    public bool UpdateProduct(Product product)
    {
        try
        {
            int result = new DBProductContext().UpdateProduct(product);

            if (result > 0)
            {
                return true;
            }
            return false;
        }
        catch { throw; }
    }

    #endregion UpdateProduct

    #region DeleteProduct

    public bool DeleteProduct(int id)
    {
        try
        {
            int result = new DBProductContext().DeleteProduct(id);

            if (result > 0)
            {
                return true;
            }
            return false;
        }
        catch { throw; }
    }

    #endregion DeleteProduct
}
```

After Migration:

- We should just create interface “IBLProductHandler” based on “BLProductHandler” and that interface inherit in “BLProductHandler” which is already implemented.

```
public interface IBLProductHandler
{
    List<Product> GetProducts();
    Product GetProductById(int id);
```

```
    bool AddProduct(Product product);
    bool UpdateProduct(Product product);
    bool DeleteProduct(int id);
}
```

- Now we should just inject dependency of “IDALProductContext” using constructor injection and replace all the instance of “DALProductContext” with “IDALProductContext” instance.

```
public class BLProductHandler : IBLProductHandler
{
    private readonly IDbProductContext _context;

    public BLProductHandler(IDbProductContext context)
    {
        _context = context;
    }

    #region GetProducts

    public List<Product> GetProducts()
    {
        try
        {
            return _context.GetProducts();
        }
        catch
        {
            throw;
        }
    }

    #endregion GetProducts

    #region GetProductById

    public Product GetProductById(int id)
    {
        try
        {
            return _context.GetProductById(id);
        }
        catch { throw; }
    }

    #endregion GetProductById

    #region AddProduct

    public bool AddProduct(Product product)
    {
        try
        {
            int result = _context.AddProduct(product);
```

```
            if (result > 0)
            {
                return true;
            }
            return false;
        }
        catch { throw; }
    }

    #endregion AddProduct

    #region UpdateProduct

    public bool UpdateProduct(Product product)
    {
        try
        {
            int result = _context.UpdateProduct(product);

            if (result > 0)
            {
                return true;
            }
            return false;
        }
        catch { throw; }
    }

    #endregion UpdateProduct

    #region DeleteProduct

    public bool DeleteProduct(int id)
    {
        try
        {
            int result = _context.DeleteProduct(id);

            if (result > 0)
            {
                return true;
            }
            return false;
        }
        catch { throw; }
    }

    #endregion DeleteProduct
}
```

2.5 Migrate Controllers

- Now we migrate Controllers, now we just use dependency injection of “IBLProductHandler” in every controller instead of create object “BLProductHandler” in every method.

Migration

Before Migration:

- We have “ProductController” like,

```
namespace ProductAPI.Controllers
{
    public class ProductController : ApiController
    {
        [HttpGet]
        public IHttpActionResult GetUsers()
        {
            return Ok(new BLProductHandler().GetProducts());
        }

        [HttpGet]
        public IHttpActionResult GetUser(int id)
        {
            return Ok(new BLProductHandler().GetProductById(id));
        }

        [HttpPost]
        public IHttpActionResult AddProduct(Product product)
        {
            if (new BLProductHandler().AddProduct(product))
            {
                return Created("", "Product added successfully");
            }
            return BadRequest();
        }

        [HttpPut]
        public IHttpActionResult Update(Product product)
        {
            if (new BLProductHandler().UpdateProduct(product))
            {
                return Ok("Product updated successfully");
            }
            return BadRequest();
        }

        [HttpDelete]
        public IHttpActionResult AddProduct(int id)
        {
            if (new BLProductHandler().DeleteProduct(id))
            {
                return Ok("Product deleted successfully");
            }
            return NotFound();
        }
    }
}
```

After Migration:

Migration

- Now as I mentation in early part, we just inject dependency of “IBLProductHandler” and use it instead of creating object of “BLProductHandler” in every time.
- Also as we learn in last chapter in ASP.NET Web API 2 support attribute based routing as well as conventional based routing where in ASP.NET Core Web API we should just use attribute based routing, we can’t use conventional based routing.
- So we also define routing in controller.
- We also change base class of controller **ControllerBase** instead of **ApiController**.
- We also one more change in return type of action method which is we replace **IHttpActionResult** with **IActionResult**.

```
namespace ProductAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]s")]
    public class ProductController : ControllerBase
    {
        private readonly IBLProductHandler _bLProductHandler;

        public ProductController(IBLProductHandler bLProductHandler)
        {
            _bLProductHandler = bLProductHandler;
        }

        [HttpGet]
        public IActionResult GetUsers()
        {
            return Ok(_bLProductHandler.GetProducts());
        }

        [HttpGet("{id:int}")]
        public IActionResult GetUser(int id)
        {
            return Ok(_bLProductHandler.GetProductById(id));
        }

        [HttpPost]
        public IActionResult AddProduct(Product product)
        {
            if (_bLProductHandler.AddProduct(product))
            {
                return Created("", "Product added successfully");
            }
            return BadRequest();
        }

        [HttpPut]
        public IActionResult Update(Product product)
        {
            if (_bLProductHandler.UpdateProduct(product))
            {
                return Ok("Product updated successfully");
            }
```

```
            return BadRequest();
        }

        [HttpDelete("{id:int}")]
        public IActionResult AddProduct(int id)
        {
            if (_bLProductHandler.DeleteProduct(id))
            {
                return Ok("Product deleted successfully");
            }
            return NotFound();
        }
    }
}
```

2.6 Register Dependencies

- Now we just register all the dependencies which we are use in our application in **Program.cs**.

```
using Newtonsoft.Json.Serialization;
using ProductAPI.BAL;
using ProductAPI.BAL.Services;
using ProductAPI.DAL;
using ProductAPI.DAL.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers().AddNewtonsoftJson(options =>
{
    options.SerializerSettings.ContractResolver = new DefaultContractResolver();
});

// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddScoped<IBLProductHandler, BLProductHandler>();
builder.Services.AddScoped<IDbProductContext, DBProductContext>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();
```

Migration

```
app.MapControllers();

app.Run();
```

- Now we have migrated application which use .NET Core.