Contents

[Create ASP.NET Core Web Application- webapi - security selected 2](#_Toc43853409)

[Add Logging 2](#_Toc43853410)

[Create Models (separate Class Library or Folder?) - you decide (better in class library for production I'd say) - add Microsoft.EntityFrameworkCore packages first 3](#_Toc43853411)

[Create a RepositoryContext.cs (inherits from DbContext) for mapping between EF Core and Database. We can also seed some of the tables with data (which will run first time only) 7](#_Toc43853412)

[Create a new Database (called 'TestDatabase') in SSMS where tables will be created 8](#_Toc43853413)

[Add the Database ConnectionString to appsettings.json 8](#_Toc43853414)

[Associate the ConnectionString with the RepositoryContext in ConfigureServices method of Startup.cs 9](#_Toc43853415)

[Create the database tables (using EF Migrations) via the Package Manager Console 11](#_Toc43853416)

[Create Generic Repository for layer between business objects and EntityFrameworkCore 15](#_Toc43853417)

[Create Model specific Repositories for layer between business objects and EntityFrameworkCore 17](#_Toc43853418)

[Create IUnitOfWork and UnitOfWork 18](#_Toc43853419)

[Add the DI setup for the UnitOfWork in ConfigureServices method of Startup.cs 19](#_Toc43853420)

[Create Web API Controller - CustomerController 19](#_Toc43853421)

[Test Web API with Postman 21](#_Toc43853422)

[Add Swagger 24](#_Toc43853423)

[Add Unit Tests (using xUnit and MOQ) 25](#_Toc43853424)

# Create ASP.NET Core Web Application- webapi - security selected

(build and run the project to make sure everything ok)

iis version ==> https://localhost:44391/weatherforecast

dotnet cli version (Kestrel) ==> http://localhost:5000/weatherforecast

- Properties/launchSettings.json (turn off browser)

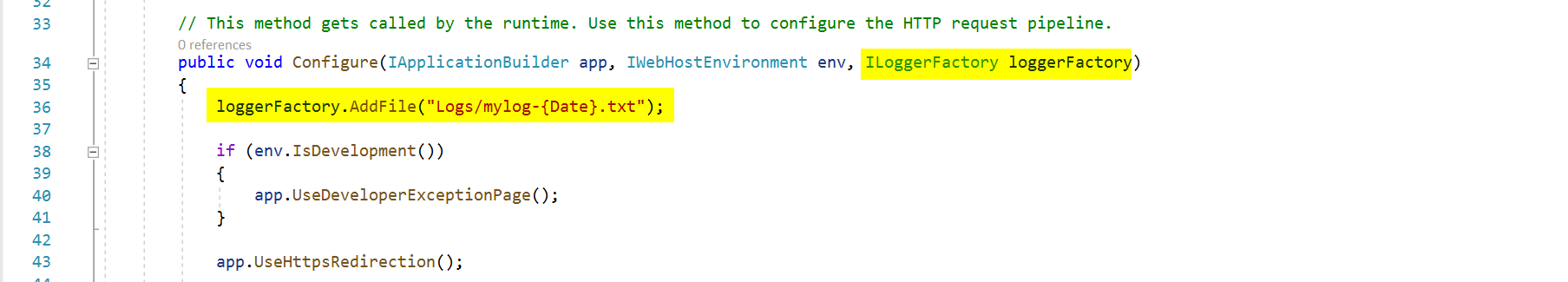
- PROJECT - PROPERTIES - Debug -

Environment Variables: ASPNETCORE\_ENVIRONMENT DEVELOPMENT

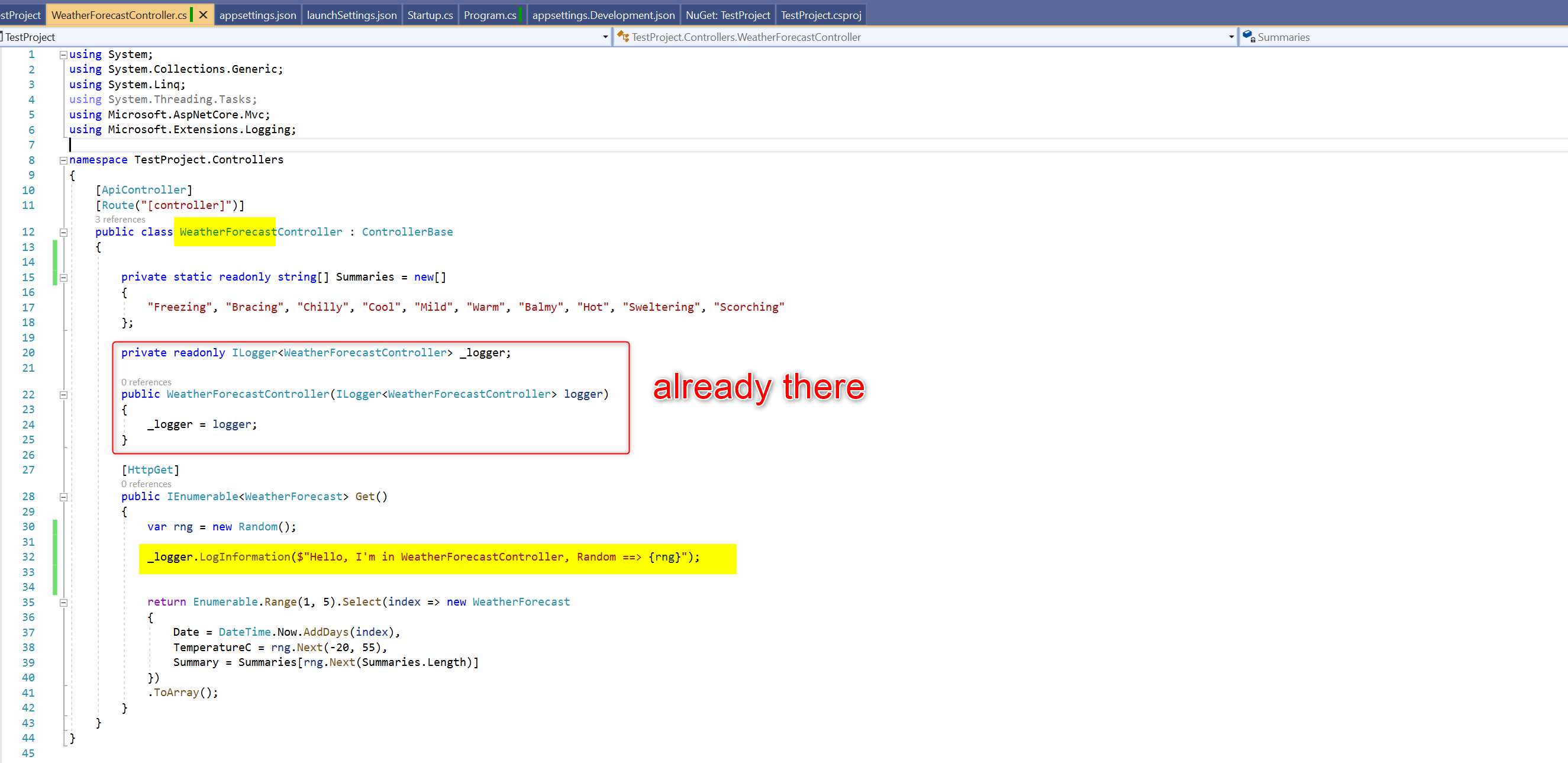
# Add Logging

simply add **Serilog.Extensions.Logging.File** nuget package (and amend Startup.cs - 'configure' method)

<https://github.com/serilog/serilog-extensions-logging-file>

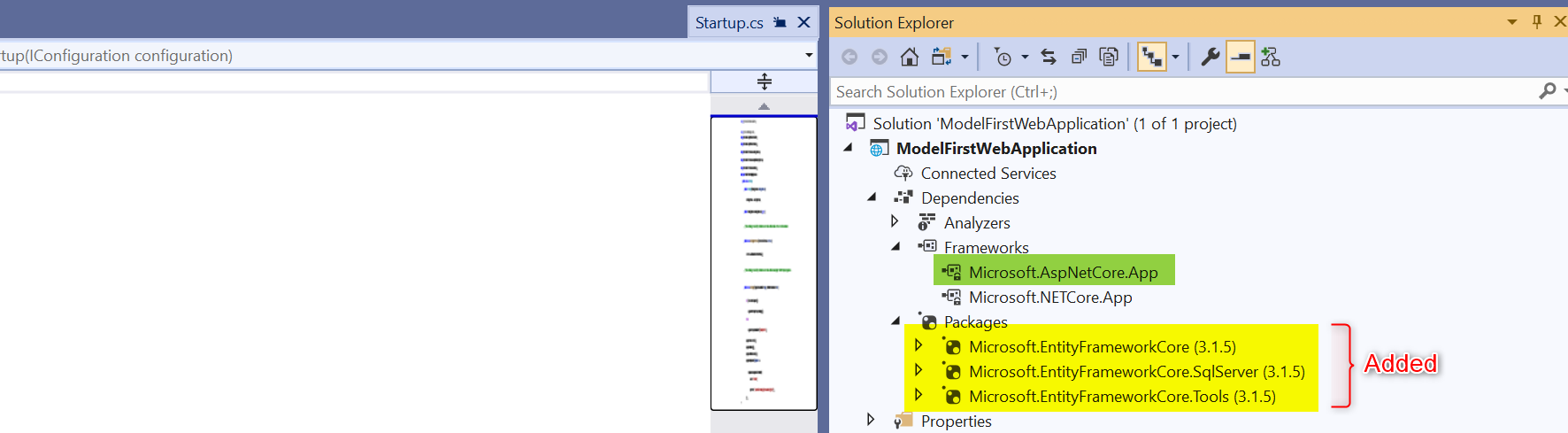


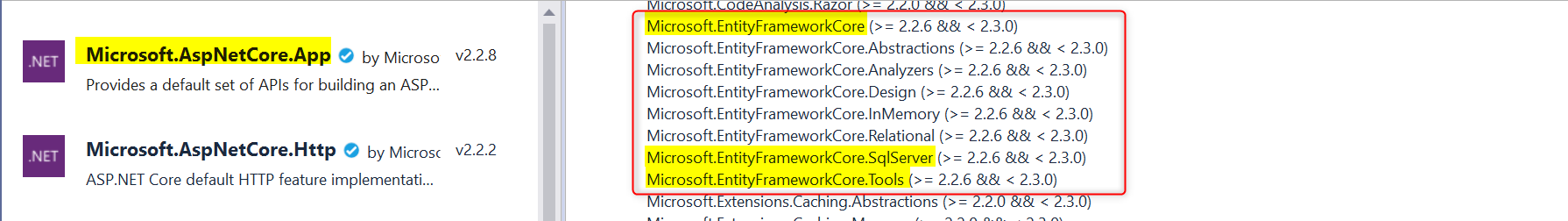
As an example, in the controller, you already have the Dependency Injection for the logger service, so you only need to add line(s) to write to logger file.



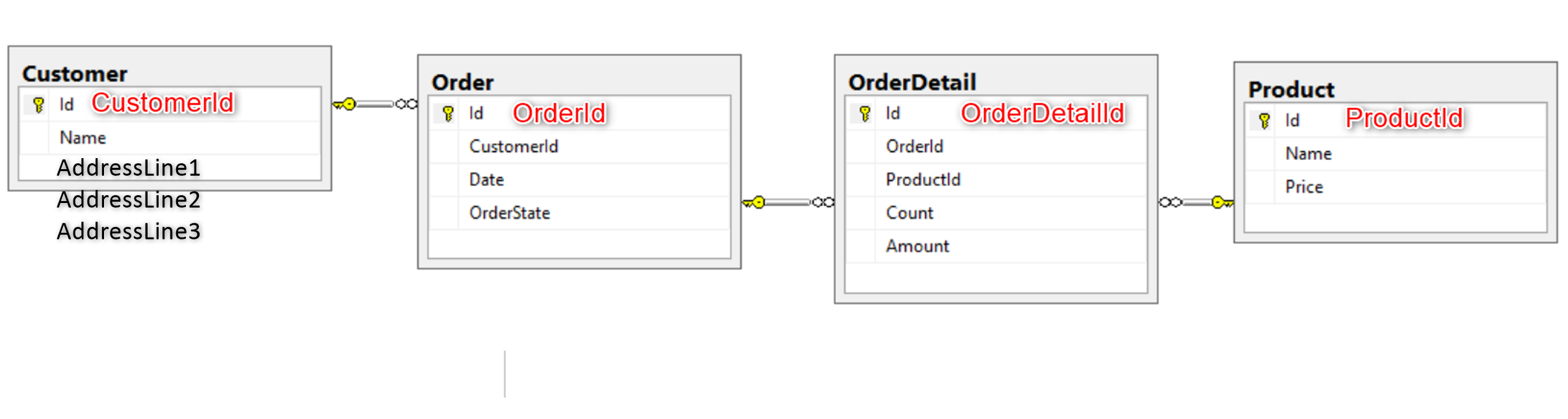
# Create Models (separate Class Library or Folder?) - you decide (better in class library for production I'd say) - add Microsoft.EntityFrameworkCore packages first

n.b. We will add Microsoft EntityFrameworkCore packages to application since we'll need this for **'data annotations'** in the models and for **DbContext** and for EntityFrameworkCore **Migrations**; (even though some of these assemblys come as part of already installed Microsoft.AspNetCore.App





(2) We'll base our database on the following database model (n.b. columns renamed for better understanding)



**Customer.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

namespace TestProject.Models

{

    [Table("Customer")]

    public class Customer

    {

        [Key]

        public int CustomerId { get; set; }

        [Required]

        public string Name { get; set; }

        [Required]

        public string AddressLine1 { get; set; }

        [Required]

        public string AddressLine2 { get; set; }

        [Required]

        public string AddressLine3 { get; set; }

        public ICollection<Order> Orders { get; set; }

    }

}

**Order.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

namespace TestProject.Models

{

    [Table("Order")]

    public class Order

    {

        [Key]

        public int OrderId { get; set; }

        [Required]

        [DataType(DataType.Date)]

        public DateTime Date { get; set; }

        [Required]

        public string OrderState { get; set; }

        [Required]

        [ForeignKey("Customer")]

        public int CustomerId { get; set; }

        public Customer Customer { get; set; }

        public ICollection<OrderDetail> OrderDetails { get; set; }

    }

}

**OrderDetail.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Net.Http.Headers;

using System.Threading.Tasks;

namespace TestProject.Models

{

    [Table("OrderDetail")]

    public class OrderDetail

    {

        [Key]

        public int OrderDetailId { get; set; }

        [Required]

        public int Count { get; set; }

        [Required]

        [Column(TypeName = "decimal(18,2)")]

        public decimal Amount { get; set; }

        [Required]

        [ForeignKey("Product")]

        public int OrderId { get; set; }

        public Order Order { get; set; }

        [Required]

        [ForeignKey("Product")]

        public int ProductId { get; set; }

        public Product Product { get; set; }

    }

}

**Product.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

namespace TestProject.Models

{

    [Table("Product")]

    public class Product

    {

        [Key]

        public int ProductId { get; set; }

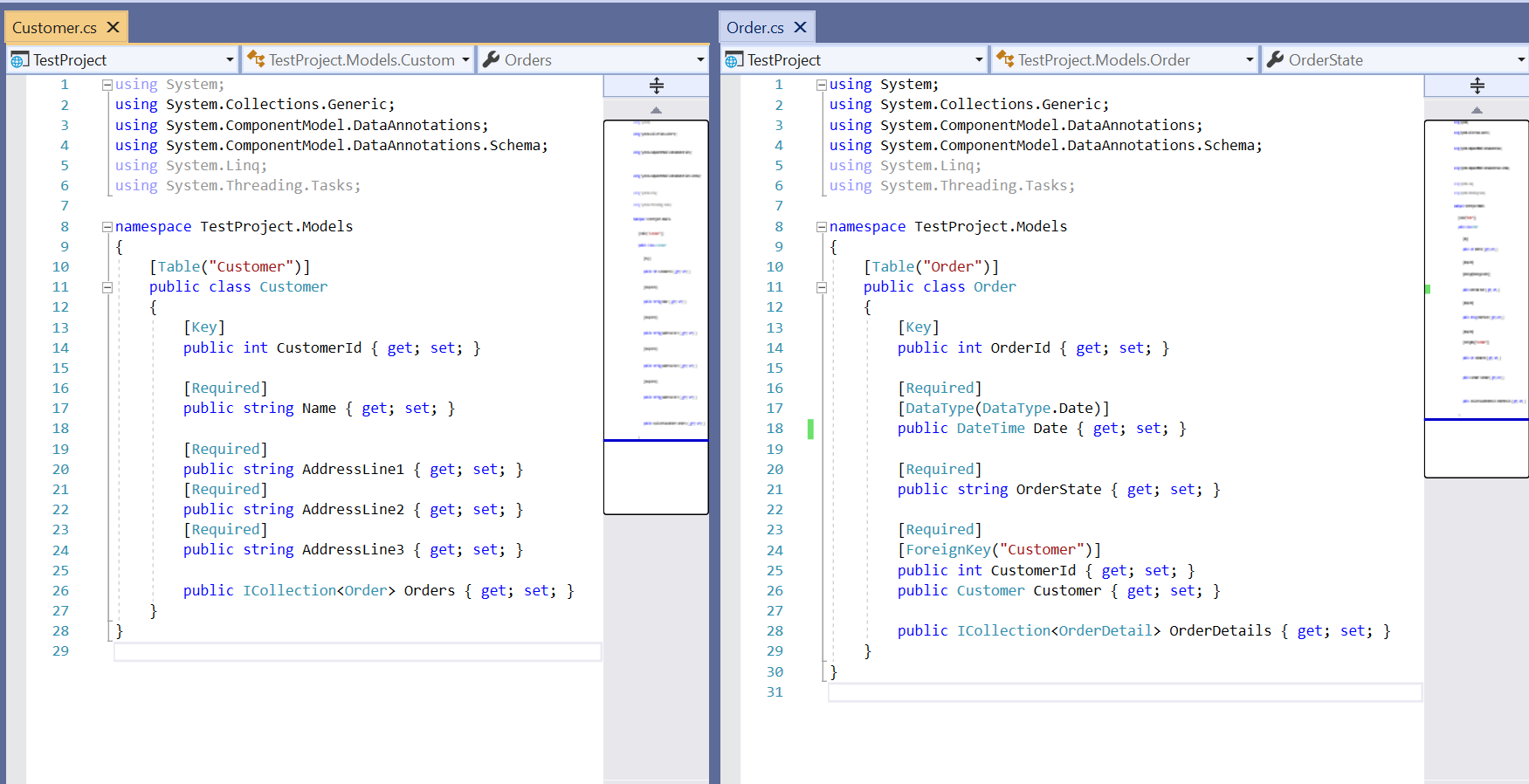
        [Required]

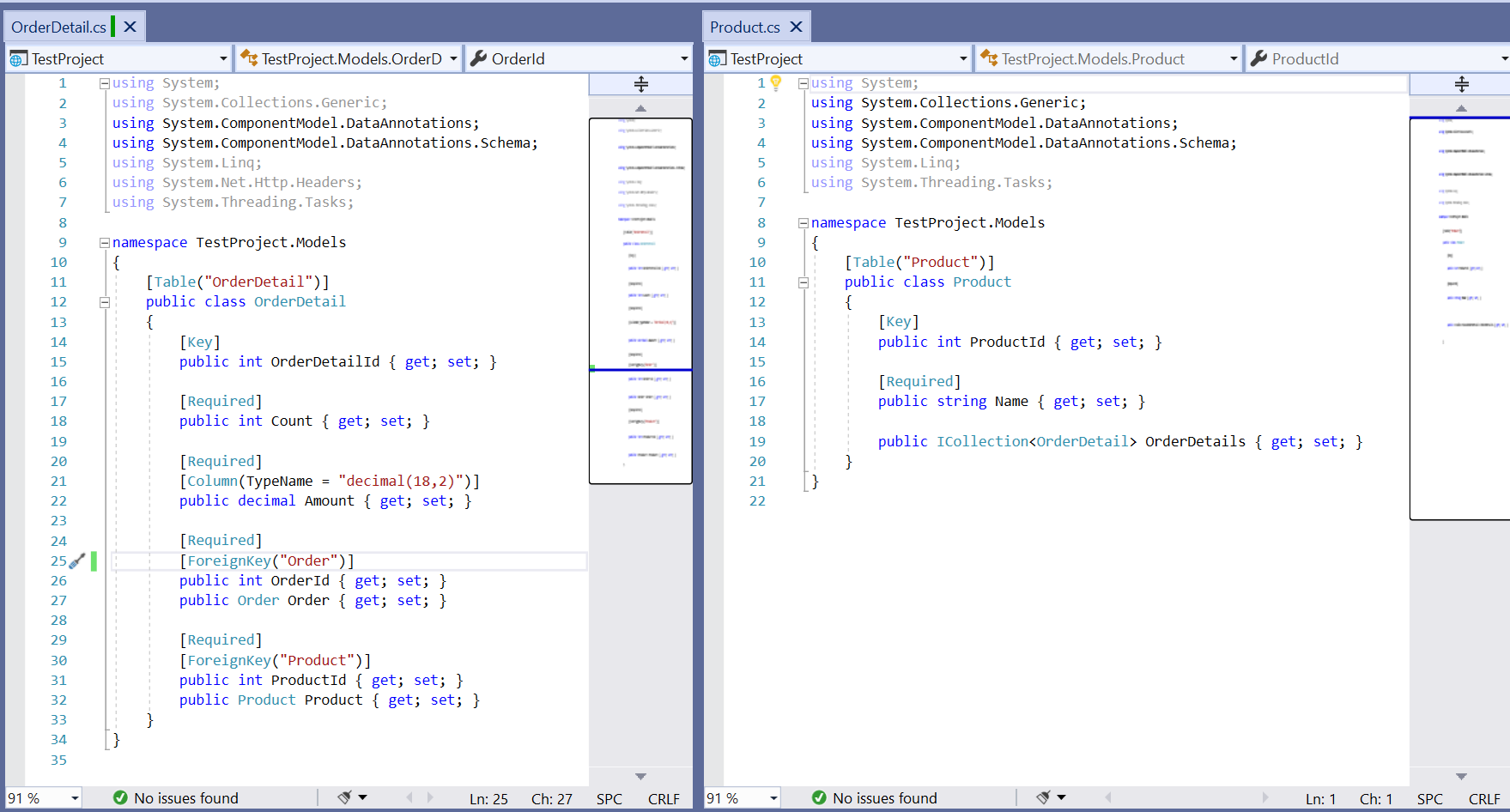
        public string Name { get; set; }

        public ICollection<OrderDetail> OrderDetails { get; set; }

    }

}





# Create a RepositoryContext.cs (inherits from DbContext) for mapping between EF Core and Database. We can also seed some of the tables with data (which will run first time only)

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using TestProject.Models;

namespace TestProject.Repository

{

    public class RepositoryContext : DbContext

    {

        public RepositoryContext(DbContextOptions options) : base(options) { }

        public DbSet<Customer> Customer { get; set; }

        public DbSet<Order> Order { get; set; }

        public DbSet<OrderDetail> OrderDetail { get; set; }

        public DbSet<Product> Product { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            modelBuilder.Entity<Customer>().HasData(

                new Customer { CustomerId = 1, Name = "John", AddressLine1 = "1 High St", AddressLine2 = "Burton", AddressLine3 = "UK" },

                new Customer { CustomerId = 2, Name = "Michael", AddressLine1 = "2 High St", AddressLine2 = "Burton", AddressLine3 = "UK" },

                new Customer { CustomerId = 3, Name = "Steven", AddressLine1 = "3 High St", AddressLine2 = "Burton", AddressLine3 = "UK" }

            );

            // n.b. the use of anonymous type

            modelBuilder.Entity<Order>().HasData(

                new Order { OrderId = 1, Date = new DateTime(2018, 12, 29), OrderState = "Pending", CustomerId = 1 },

                new Order { OrderId = 2, Date = new DateTime(2019, 12, 30), OrderState = "Approved", CustomerId = 1 },

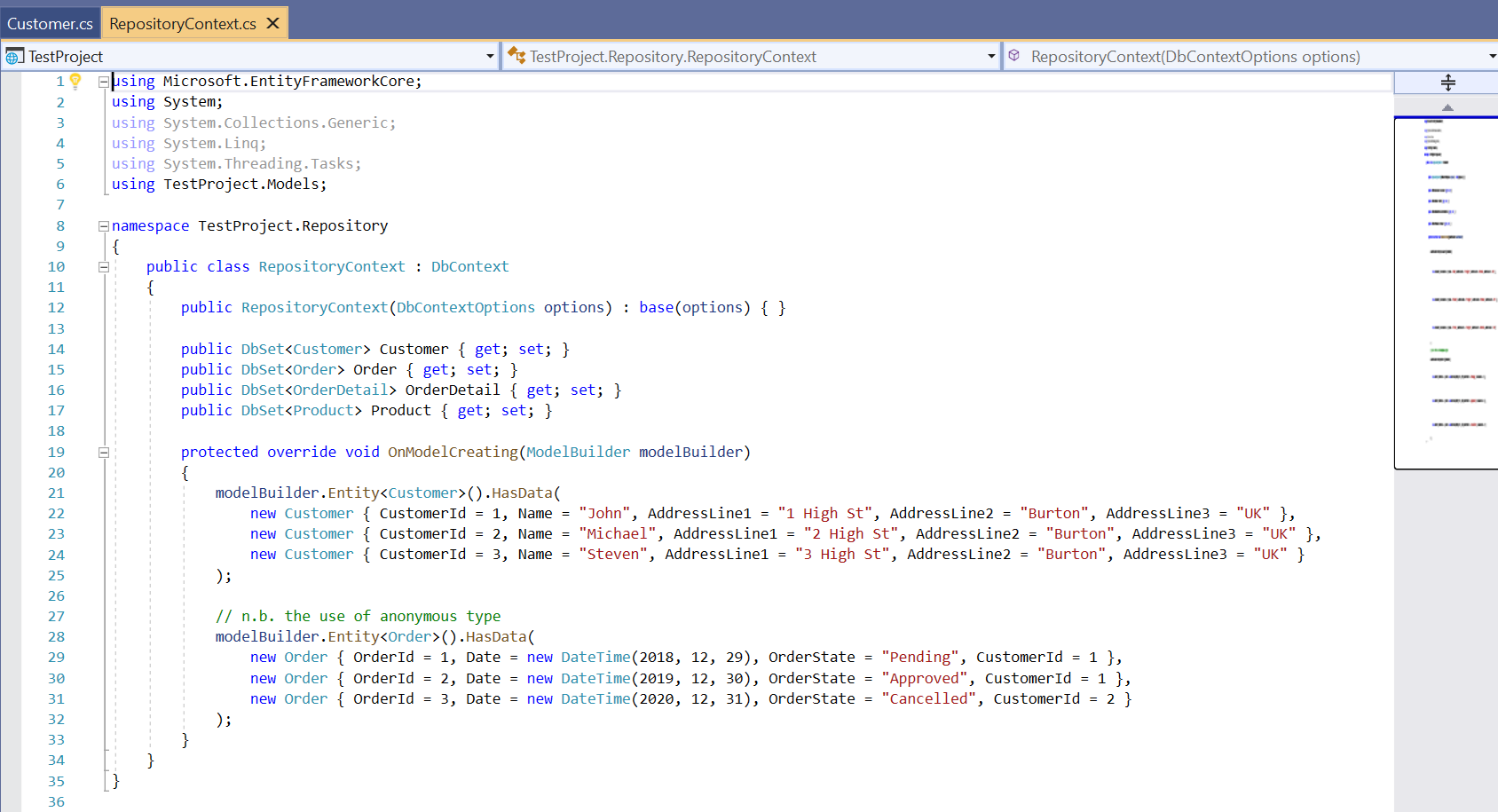
                new Order { OrderId = 3, Date = new DateTime(2020, 12, 31), OrderState = "Cancelled", CustomerId = 2 }

            );

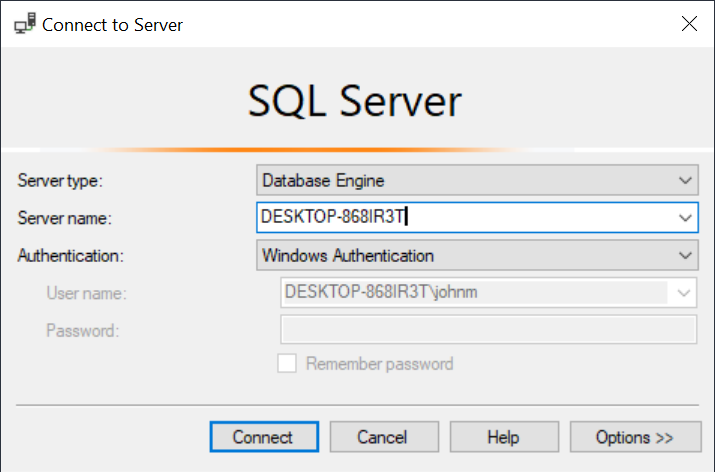
        }

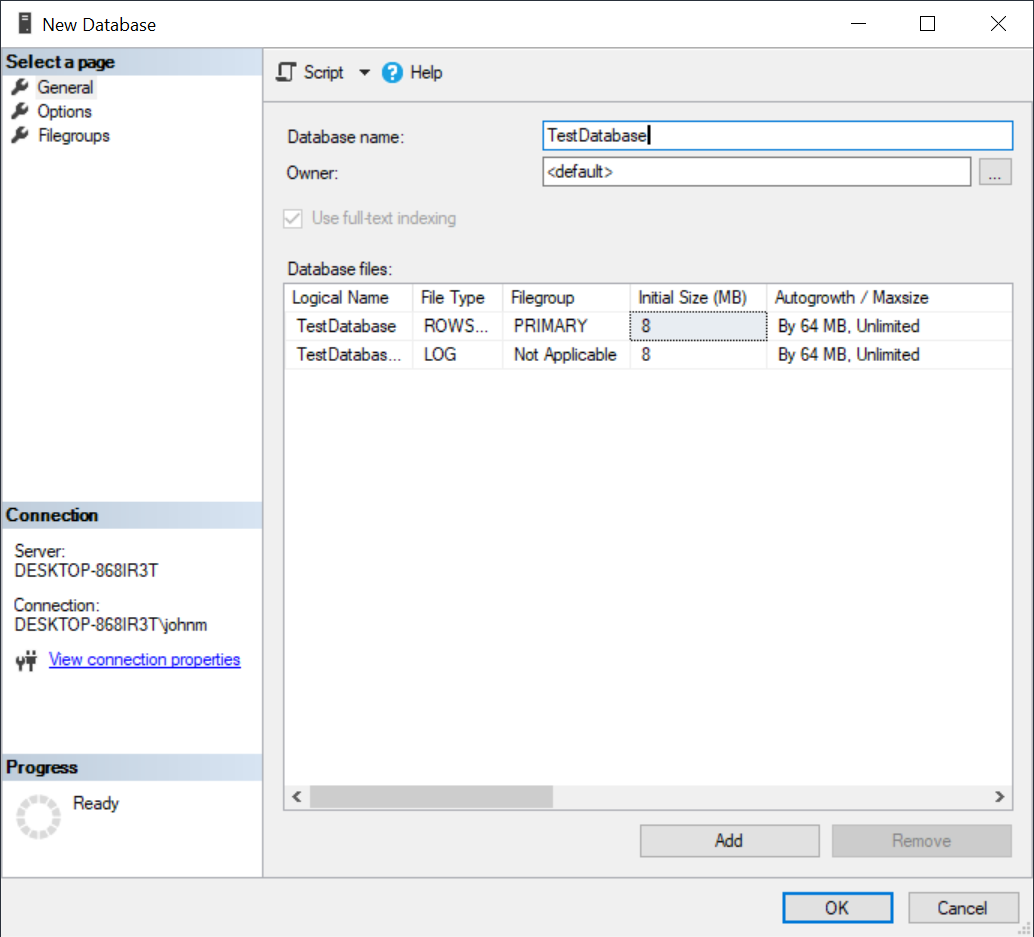
    }

}



# Create a new Database (called 'TestDatabase') in SSMS where tables will be created





# Add the Database ConnectionString to appsettings.json

"ConnectionStrings": {

    "sqlConnection": "server=DESKTOP-868IR3T; database=TestDatabase; Integrated Security=true"

  },



# Associate the ConnectionString with the RepositoryContext in ConfigureServices method of Startup.cs

n.b. only highlighted lines need adding

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.HttpsPolicy;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using TestProject.Repository;

namespace TestProject

{

    public class Startup

    {

        public Startup(IConfiguration configuration)

        {

            Configuration = configuration;

        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.

        public void ConfigureServices(IServiceCollection services)

        {

            services.AddDbContext<RepositoryContext>(opts => opts.UseSqlServer(Configuration.GetConnectionString("sqlConnection")));

            services.AddControllers();

        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env, ILoggerFactory loggerFactory)

        {

            loggerFactory.AddFile("Logs/mylog-{Date}.txt");

            if (env.IsDevelopment())

            {

                app.UseDeveloperExceptionPage();

            }

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>

            {

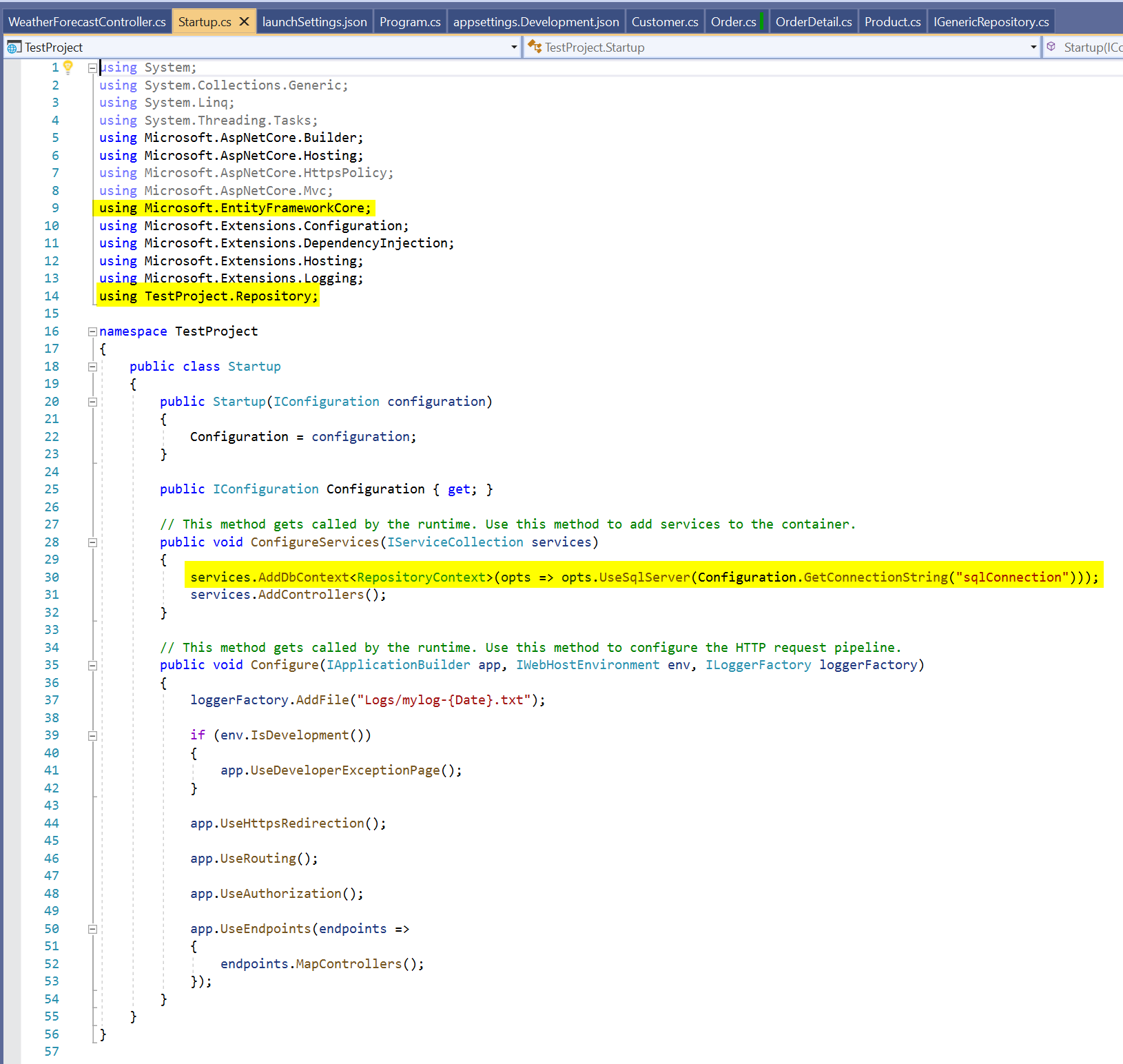
                endpoints.MapControllers();

            });

        }

    }

}



# Create the database tables (using EF Migrations) via the Package Manager Console

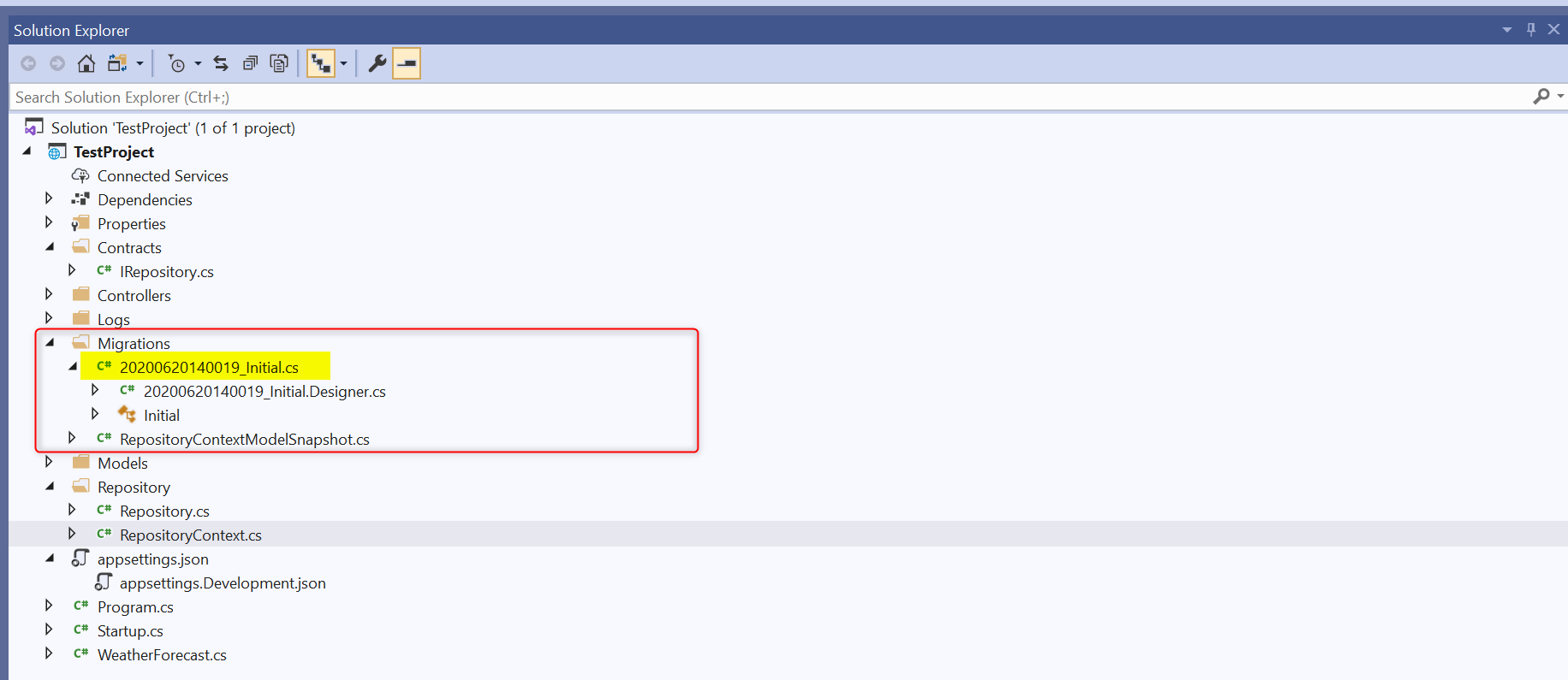
BEFORE



n.b. You do NOT need to enable-migrations; just do the following:

PM> Add-Migration Initial -verbose







using System;

using Microsoft.EntityFrameworkCore.Migrations;

namespace TestProject.Migrations

{

    public partial class Initial : Migration

    {

        protected override void Up(MigrationBuilder migrationBuilder)

        {

            migrationBuilder.CreateTable(

                name: "Customer",

                columns: table => new

                {

                    CustomerId = table.Column<int>(nullable: false)

                        .Annotation("SqlServer:Identity", "1, 1"),

                    Name = table.Column<string>(nullable: false),

                    AddressLine1 = table.Column<string>(nullable: false),

                    AddressLine2 = table.Column<string>(nullable: false),

                    AddressLine3 = table.Column<string>(nullable: false)

                },

                constraints: table =>

                {

                    table.PrimaryKey("PK\_Customer", x => x.CustomerId);

                });

            migrationBuilder.CreateTable(

                name: "Product",

                columns: table => new

                {

                    ProductId = table.Column<int>(nullable: false)

                        .Annotation("SqlServer:Identity", "1, 1"),

                    Name = table.Column<string>(nullable: false)

                },

                constraints: table =>

                {

                    table.PrimaryKey("PK\_Product", x => x.ProductId);

                });

            migrationBuilder.CreateTable(

                name: "Order",

                columns: table => new

                {

                    OrderId = table.Column<int>(nullable: false)

                        .Annotation("SqlServer:Identity", "1, 1"),

                    Date = table.Column<DateTime>(nullable: false),

                    OrderState = table.Column<string>(nullable: false),

                    CustomerId = table.Column<int>(nullable: false)

                },

                constraints: table =>

                {

                    table.PrimaryKey("PK\_Order", x => x.OrderId);

                    table.ForeignKey(

                        name: "FK\_Order\_Customer\_CustomerId",

                        column: x => x.CustomerId,

                        principalTable: "Customer",

                        principalColumn: "CustomerId",

                        onDelete: ReferentialAction.Cascade);

                });

            migrationBuilder.CreateTable(

                name: "OrderDetail",

                columns: table => new

                {

                    OrderDetailId = table.Column<int>(nullable: false)

                        .Annotation("SqlServer:Identity", "1, 1"),

                    Count = table.Column<int>(nullable: false),

                    Amount = table.Column<decimal>(type: "decimal(18,2)", nullable: false),

                    OrderId = table.Column<int>(nullable: false),

                    ProductId = table.Column<int>(nullable: false)

                },

                constraints: table =>

                {

                    table.PrimaryKey("PK\_OrderDetail", x => x.OrderDetailId);

                    table.ForeignKey(

                        name: "FK\_OrderDetail\_Order\_OrderId",

                        column: x => x.OrderId,

                        principalTable: "Order",

                        principalColumn: "OrderId",

                        onDelete: ReferentialAction.Cascade);

                    table.ForeignKey(

                        name: "FK\_OrderDetail\_Product\_ProductId",

                        column: x => x.ProductId,

                        principalTable: "Product",

                        principalColumn: "ProductId",

                        onDelete: ReferentialAction.Cascade);

                });

            migrationBuilder.InsertData(

                table: "Customer",

                columns: new[] { "CustomerId", "AddressLine1", "AddressLine2", "AddressLine3", "Name" },

                values: new object[] { 1, "1 High St", "Burton", "UK", "John" });

            migrationBuilder.InsertData(

                table: "Customer",

                columns: new[] { "CustomerId", "AddressLine1", "AddressLine2", "AddressLine3", "Name" },

                values: new object[] { 2, "2 High St", "Burton", "UK", "Michael" });

            migrationBuilder.InsertData(

                table: "Customer",

                columns: new[] { "CustomerId", "AddressLine1", "AddressLine2", "AddressLine3", "Name" },

                values: new object[] { 3, "3 High St", "Burton", "UK", "Steven" });

            migrationBuilder.InsertData(

                table: "Order",

                columns: new[] { "OrderId", "CustomerId", "Date", "OrderState" },

                values: new object[] { 1, 1, new DateTime(2018, 12, 29, 0, 0, 0, 0, DateTimeKind.Unspecified), "Pending" });

            migrationBuilder.InsertData(

                table: "Order",

                columns: new[] { "OrderId", "CustomerId", "Date", "OrderState" },

                values: new object[] { 2, 1, new DateTime(2019, 12, 30, 0, 0, 0, 0, DateTimeKind.Unspecified), "Approved" });

            migrationBuilder.InsertData(

                table: "Order",

                columns: new[] { "OrderId", "CustomerId", "Date", "OrderState" },

                values: new object[] { 3, 2, new DateTime(2020, 12, 31, 0, 0, 0, 0, DateTimeKind.Unspecified), "Cancelled" });

            migrationBuilder.CreateIndex(

                name: "IX\_Order\_CustomerId",

                table: "Order",

                column: "CustomerId");

            migrationBuilder.CreateIndex(

                name: "IX\_OrderDetail\_OrderId",

                table: "OrderDetail",

                column: "OrderId");

            migrationBuilder.CreateIndex(

                name: "IX\_OrderDetail\_ProductId",

                table: "OrderDetail",

                column: "ProductId");

        }

        protected override void Down(MigrationBuilder migrationBuilder)

        {

            migrationBuilder.DropTable(

                name: "OrderDetail");

            migrationBuilder.DropTable(

                name: "Order");

            migrationBuilder.DropTable(

                name: "Product");

            migrationBuilder.DropTable(

                name: "Customer");

        }

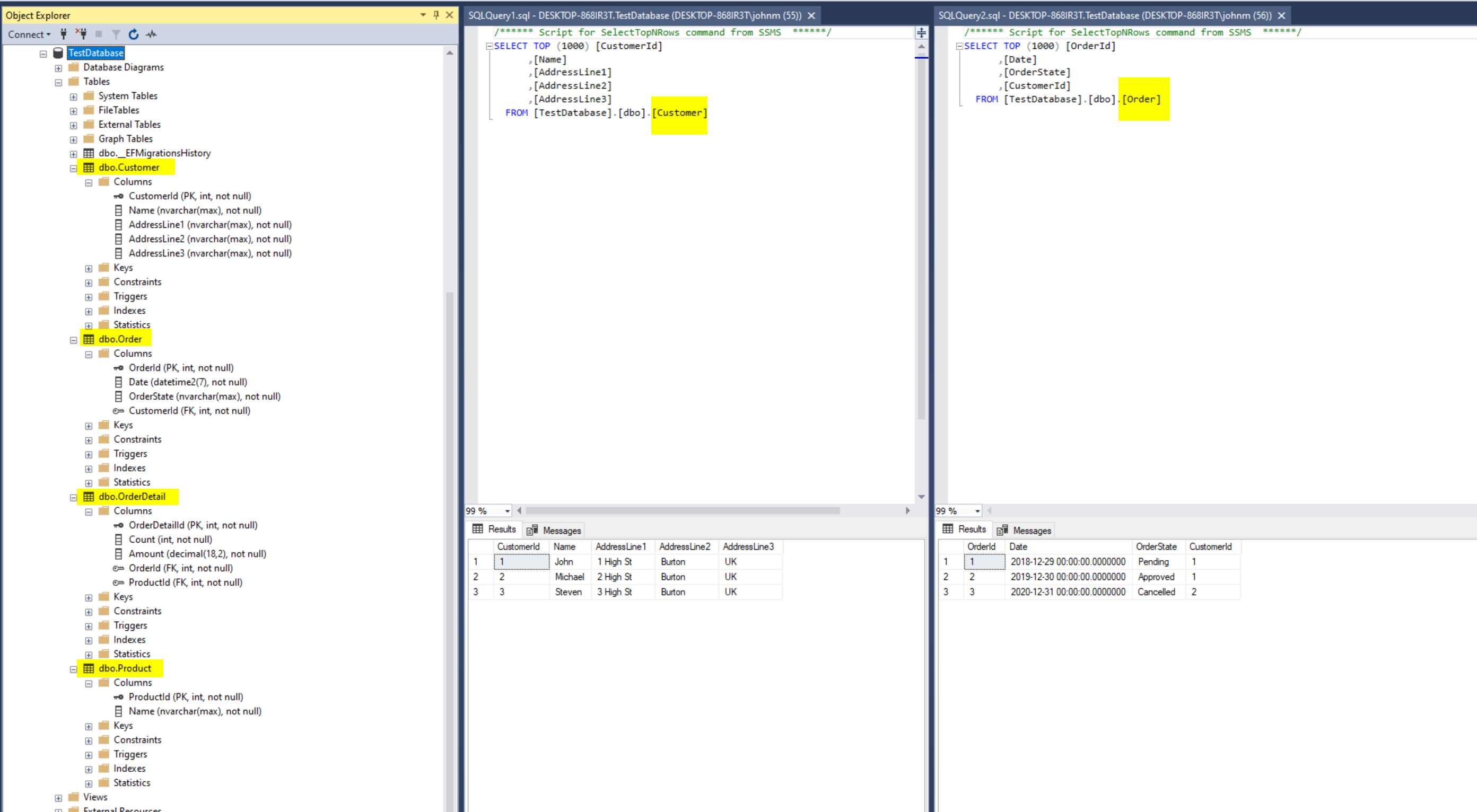
    }

}

PM> update-database -verbose



AFTER



# Create Generic Repository for layer between business objects and EntityFrameworkCore

i.e. without the repository and unit of work patterns we would have had:

controller ==> EF Core ==> SQL Server

but now we will have

controller ==> **Repository (and UnitOfWork)** ==> EF Core ==> SQL Server

**IRepository.cs (Generic Repository Interface)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace TestProject.Contracts

{

    public interface IRepository<T> where T: class

    {

        void Create(T obj);

        void Delete(object id);

        IEnumerable<T> GetAll();

        T Get(object id);

        void Update(T obj);

// n.b. Save not required since in UnitOfWork

    }

}

**Repository.cs (Generic Repository Implementation)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using TestProject.Contracts;

namespace TestProject.Repository

{

    public class Repository<T> : IRepository<T> where T : class

    {

        protected RepositoryContext repositoryContext;

        public Repository(RepositoryContext repositoryContext)

        {

            this.repositoryContext = repositoryContext;

        }

        public void Create(T obj)

        {

            repositoryContext.Set<T>().Add(obj);

        }

        public void Delete(object id)

        {

            T t = repositoryContext.Set<T>().Find(id);

            repositoryContext.Remove(t);

        }

        public IEnumerable<T> GetAll()

        {

            return repositoryContext.Set<T>().ToList();

        }

        public T Get(object id)

        {

            return repositoryContext.Set<T>().Find(id);

        }

        public void Update(T obj)

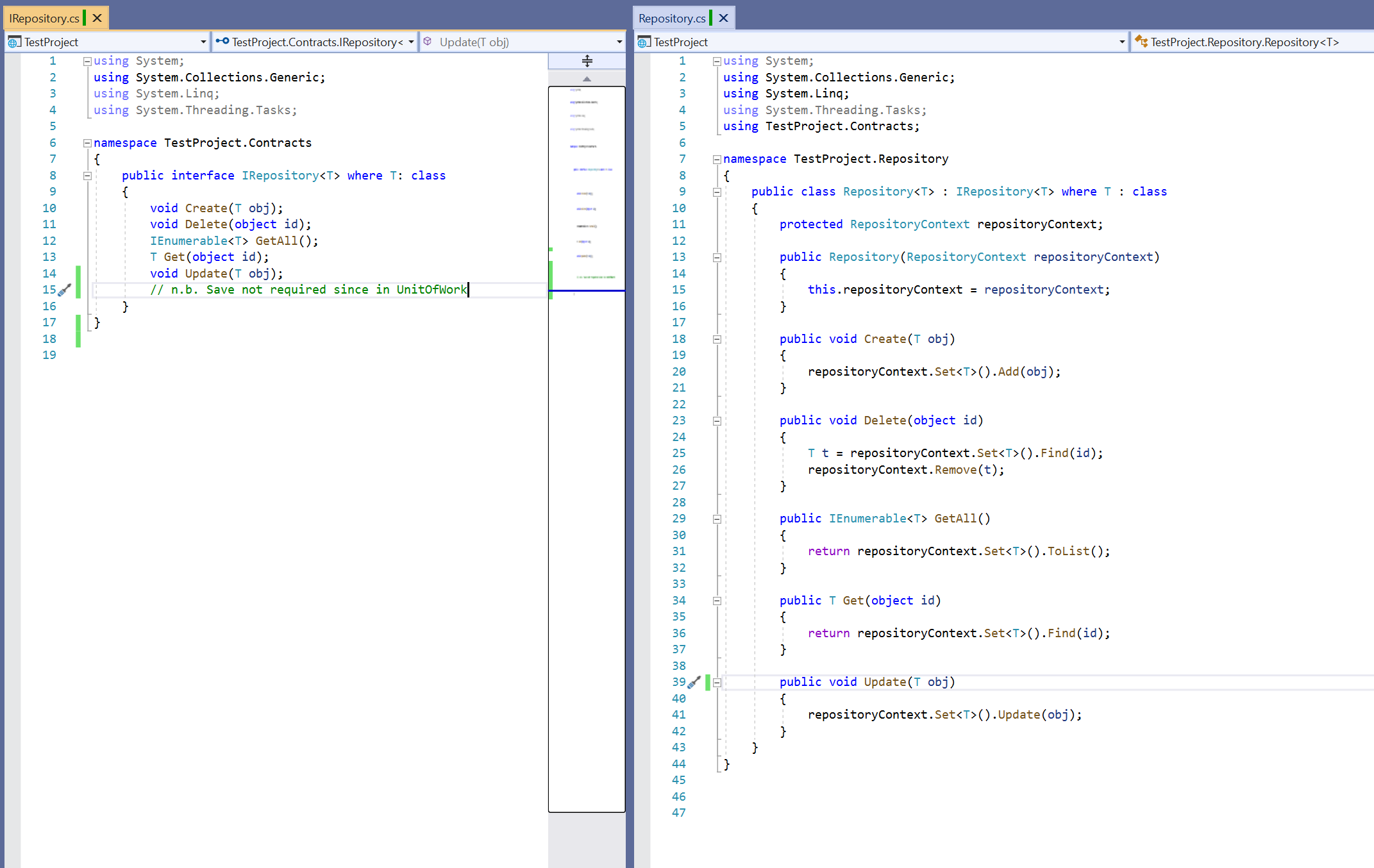
        {

            repositoryContext.Set<T>().Update(obj);

        }

    }

}



# Create Model specific Repositories for layer between business objects and EntityFrameworkCore

(I know we should have interfaces for each model repository, but we're not going to add any extra logic so not required for this application)

**CustomerRepository.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using TestProject.Models;

namespace TestProject.Repository

{

    public class CustomerRepository : Repository<Customer>

    {

        public CustomerRepository(RepositoryContext repositoryContext) : base(repositoryContext) { }

    }

}

**OrderRepository.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using TestProject.Models;

namespace TestProject.Repository

{

    public class OrderRepository : Repository<Order>

    {

        public OrderRepository(RepositoryContext repositoryContext) : base(repositoryContext) { }

    }

}

**OrderDetailRepository.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using TestProject.Models;

namespace TestProject.Repository

{

    public class OrderDetailRepository : Repository<OrderDetail>

    {

        public OrderDetailRepository(RepositoryContext repositoryContext) : base(repositoryContext) { }

    }

}

**ProductRepository.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using TestProject.Models;

namespace TestProject.Repository

{

    public class ProductRepository : Repository<Product>

    {

        public ProductRepository(RepositoryContext repositoryContext) : base(repositoryContext) { }

    }

}

# Create IUnitOfWork and UnitOfWork

**IUnitOfWork.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using TestProject.Repository;

namespace TestProject.Contracts

{

    public interface IUnitOfWork

    {

        CustomerRepository Customer { get; }

        OrderRepository Order { get; }

        OrderDetailRepository OrderDetail { get; }

        ProductRepository Product { get; }

        void Save();

    }

}

**UnitOfWork.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using TestProject.Contracts;

namespace TestProject.Repository

{

    public class UnitOfWork : IUnitOfWork

    {

        private RepositoryContext \_repositoryContext;

        private CustomerRepository \_customerRepository;

        private OrderRepository \_orderRepository;

        private OrderDetailRepository \_orderDetailRepository;

        private ProductRepository \_productRepository;

        public UnitOfWork(RepositoryContext repositoryContext)

        {

            \_repositoryContext = repositoryContext;

        }

        public CustomerRepository Customer

        {

            get

            {

                if (\_customerRepository == null) \_customerRepository = new CustomerRepository(\_repositoryContext);

                return \_customerRepository;

            }

        }

        public OrderRepository Order

        {

            get

            {

                if (\_orderRepository == null) \_orderRepository = new OrderRepository(\_repositoryContext);

                return \_orderRepository;

             }

        }

        public OrderDetailRepository OrderDetail

        {

            get

            {

                if (\_orderDetailRepository == null) \_orderDetailRepository = new OrderDetailRepository(\_repositoryContext);

                return \_orderDetailRepository;

            }

        }

        public ProductRepository Product

        {

            get

            {

                if (\_productRepository == null) \_productRepository = new ProductRepository(\_repositoryContext);

                return \_productRepository;

            }

        }

        public void Save()

        {

            \_repositoryContext.SaveChanges();

        }

    }

}

# Add the DI setup for the UnitOfWork in ConfigureServices method of Startup.cs

public void ConfigureServices(IServiceCollection services)

        {

            services.AddDbContext<RepositoryContext>(opts => opts.UseSqlServer(Configuration.GetConnectionString("sqlConnection")));

            services.AddControllers();

            services.AddScoped<IUnitOfWork, UnitOfWork>();

        }

# Create Web API Controller - CustomerController

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using TestProject.Contracts;

using TestProject.Models;

namespace TestProject.Controllers

{

    [Route("[controller]")]

    [ApiController]

    // https://localhost:5001/customer

    public class CustomerController : ControllerBase

    {

        private readonly IUnitOfWork \_unitOfWork;

        private readonly ILogger<CustomerController> \_logger;

        public CustomerController(IUnitOfWork unitOfWork, ILogger<CustomerController> logger)

        {

            \_unitOfWork = unitOfWork;

            \_logger = logger;

        }

        [HttpGet]

        public IActionResult GetCustomers()

        {

            try

            {

                var customers = \_unitOfWork.Customer.GetAll();

                return Ok(customers);

            }

            catch (Exception)

            {

                \_logger.LogInformation($"Error getting Customers data");

                return StatusCode(500, "Internal Server Error");

            }

        }

        // https://localhost:5001/customer/1

        [HttpGet("{id}", Name = "CustomerById")]

        public IActionResult Get(int id)

        {

            try

            {

                var customer = \_unitOfWork.Customer.Get(id);

                if (customer == null)

                {

                    return NotFound();

                }

                return Ok(customer);

            }

            catch (Exception)

            {

                \_logger.LogInformation($"Error getting Customer {id}");

                return StatusCode(500, "Internal Server Error");

            }

        }

        // https://localhost:5001/customer

        /\* n.b. example post data; no need for customerId since it's an Identity column

         {

        "name": "Max",

        "addressLine1": "6 High St",

        "addressLine2": "Burton",

        "addressLine3": "UK",

        "orders": null

        }

        \*/

        [HttpPost]

        public IActionResult CreateCustomer([FromBody] Customer customer)

        {

            try

            {

                if (customer == null)

                {

                    \_logger.LogInformation($"Error creating Customer");

                    return BadRequest("customer object is null");

                }

                \_unitOfWork.Customer.Create(customer);

                \_unitOfWork.Save();

                return CreatedAtRoute("CustomerById", new { id =  customer.CustomerId }, customer);

            }

            catch (Exception)

            {

                \_logger.LogInformation($"Error creating Customer");

                return StatusCode(500, "Internal Server Error");

            }

        }

        // https://localhost:5001/customer/1

        [HttpPut("{id}")]

        public IActionResult UpdateCustomer(int id, [FromBody] Customer customer)

        {

            // this can either use id (i.e. customerId) from URL or from request body

            // I decied to use it from URL

            try

            {

                if (customer == null)

                {

                    \_logger.LogInformation($"Error creating Customer");

                    return BadRequest("customer object is null");

                }

                // this didn't work - clash of transactions for same ID

                //var cus = \_unitOfWork.Customer.Get(id);

                //if (cus == null)

                //{

                //    \_logger.LogInformation($"Customer {id} does not exist");

                //    return BadRequest("customer does not exist");

                //}

                customer.CustomerId = id;

                \_unitOfWork.Customer.Update(customer);

                \_unitOfWork.Save();

                return NoContent();

            }

            catch (Exception)

            {

                \_logger.LogInformation($"Error updating Customer");

                return StatusCode(500, "Internal Server Error");

            }

        }

        // https://localhost:5001/customer/1

        [HttpDelete("{id}")]

        public IActionResult DeleteCustomer(int id)

        {

            // this can either use id (i.e. customerId) from URL or from request body

            // I decied to use it from URL

            try

            {

                \_unitOfWork.Customer.Delete(id);

                \_unitOfWork.Save();

                return NoContent();

            }

            catch (Exception)

            {

                \_logger.LogInformation($"Error updating Customer");

                return StatusCode(500, "Internal Server Error");

            }

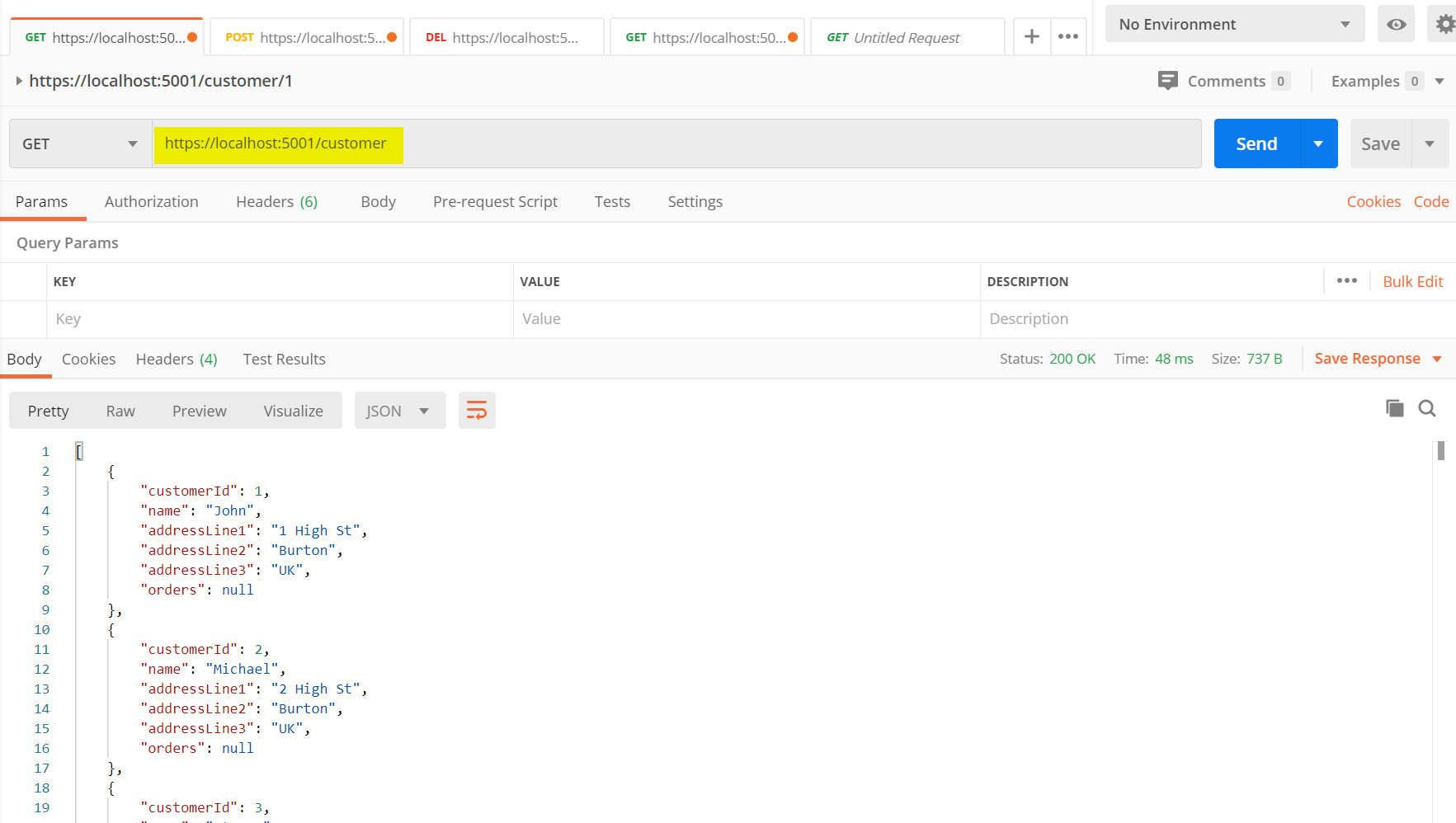
        }

    }

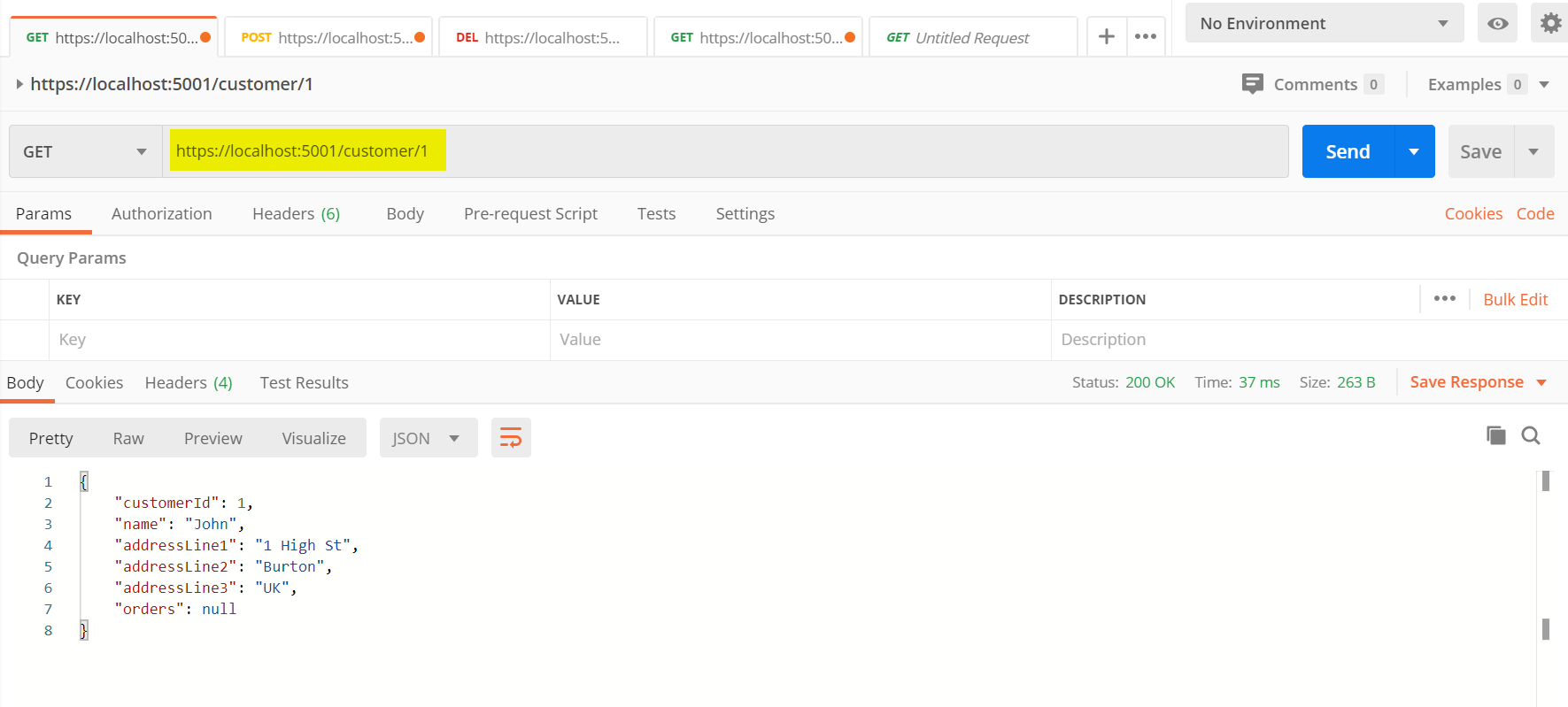
}

# Test Web API with Postman

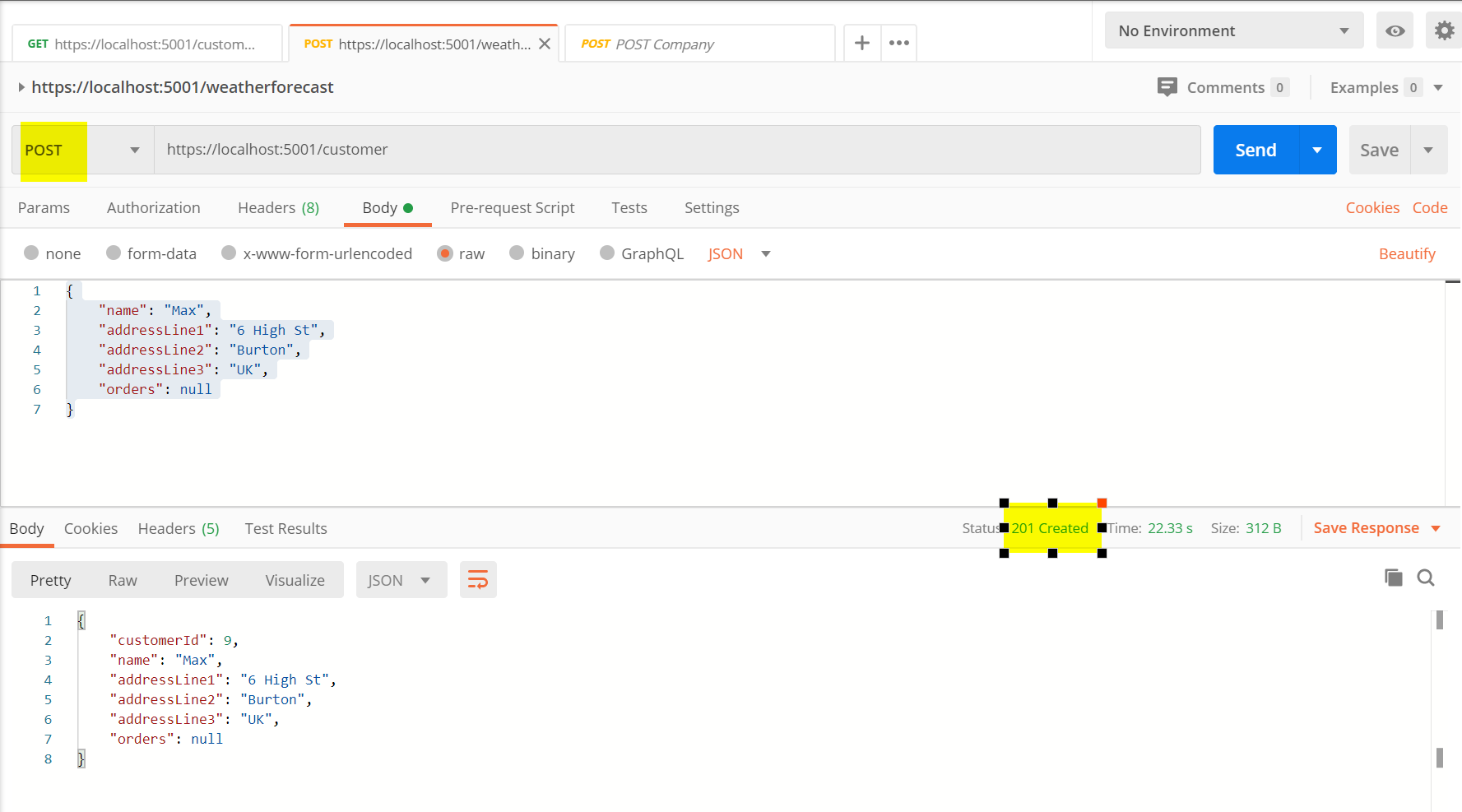
Get All Customers



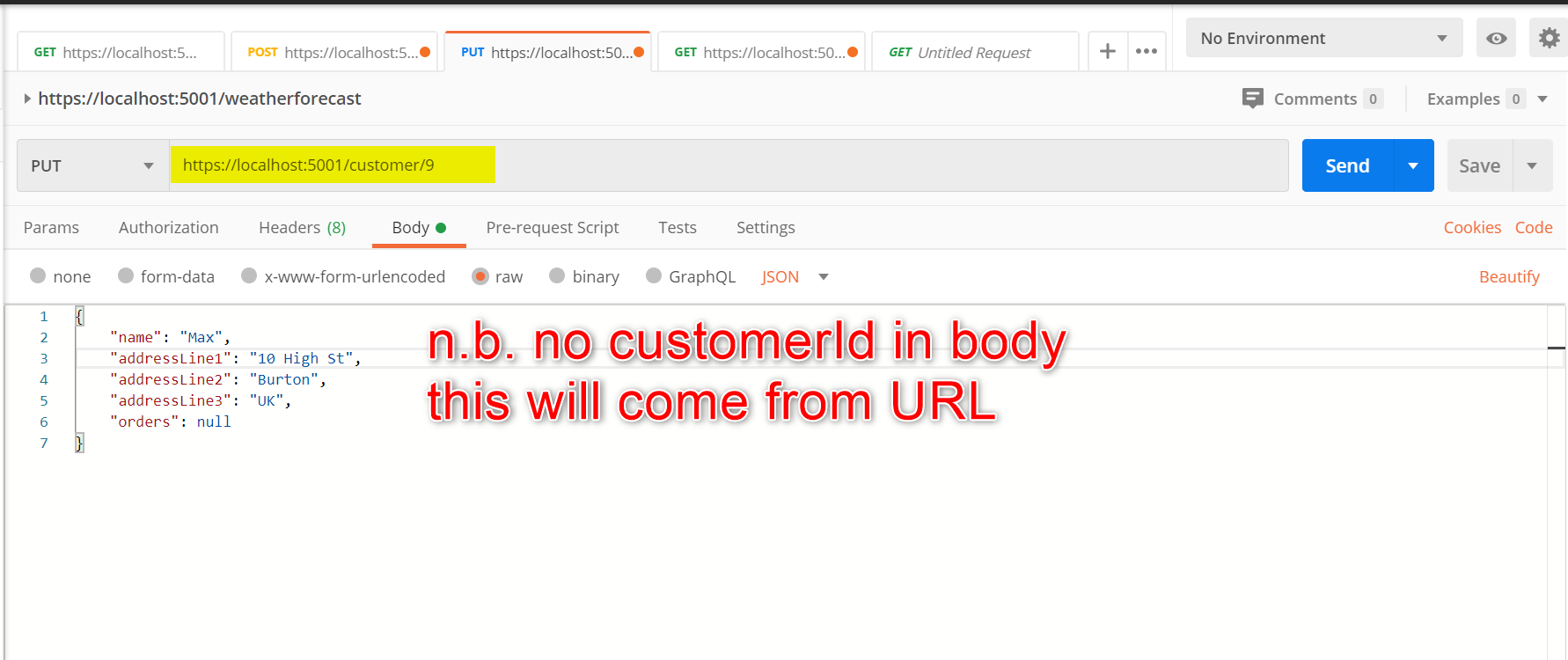
Get specific customer



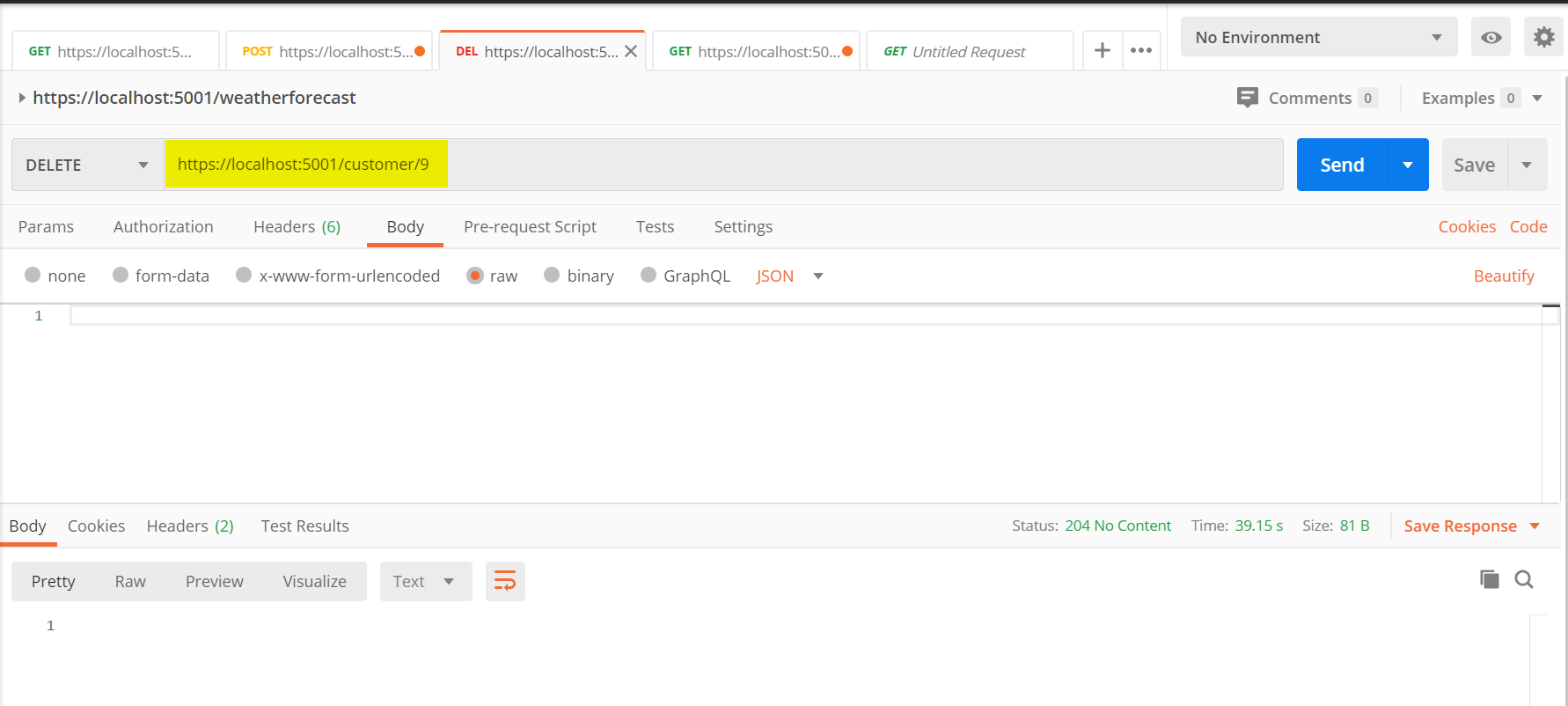
Create new customer (POST)



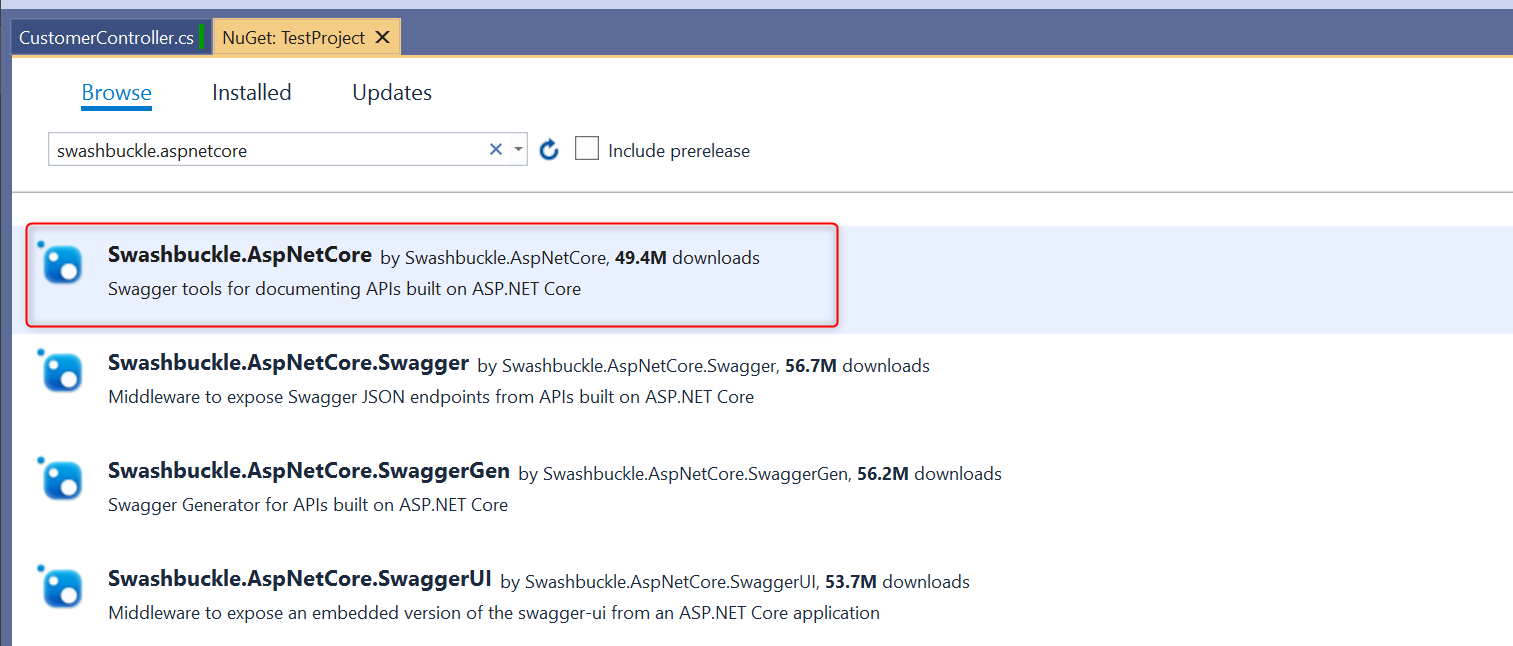
Update Customer

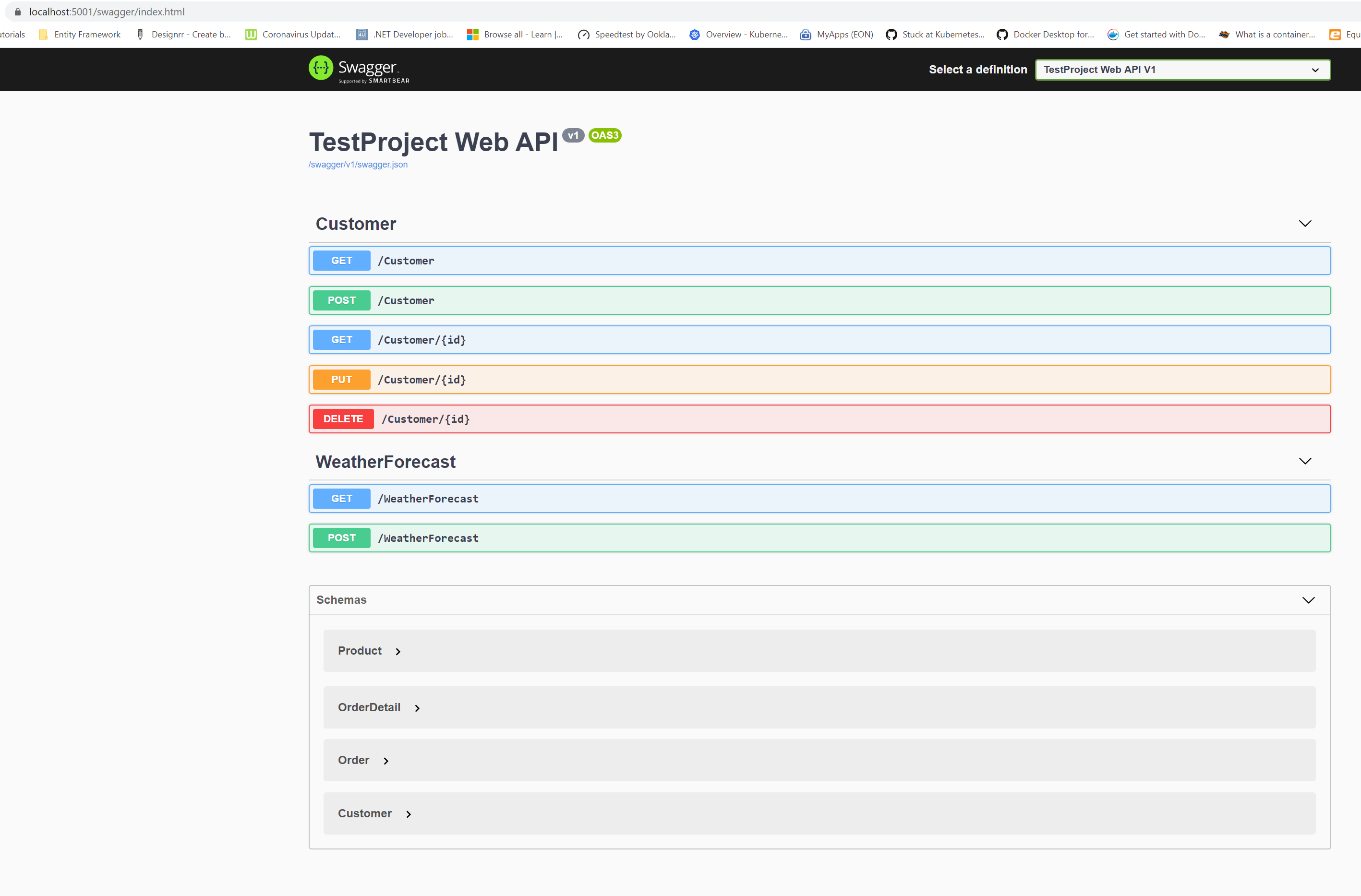


Delete Customer



# Add Swagger





# Add Unit Tests (using xUnit and MOQ)

Create an xUnit Project called TestProject.UnitTests

Create a CustomerControllerTests.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using Moq;

using System;

using System.Collections.Generic;

using System.Linq;

using TestProject.Contracts;

using TestProject.Controllers;

using TestProject.Models;

using TestProject.Repository;

using Xunit;

namespace TestProject.UnitTests

{

    public class CustomerControllerTest

    {

        Mock<ILogger<CustomerController>> \_mockLogger;

        Mock<IRepository<Customer>> \_mockCustomerRepository;

        Mock<IUnitOfWork> \_mockUnitOfWork;

        List<Customer> listCustomer = new List<Customer>

        {

                new Customer { CustomerId = 1, Name = "John", AddressLine1 = "1 High St", AddressLine2 = "Burton", AddressLine3 = "UK" },

                new Customer { CustomerId = 2, Name = "Michael", AddressLine1 = "2 High St", AddressLine2 = "Burton", AddressLine3 = "UK" },

                new Customer { CustomerId = 3, Name = "Steven", AddressLine1 = "3 High St", AddressLine2 = "Burton", AddressLine3 = "UK" }

        };

        [Fact]

        public void Get\_WhenCalled\_ReturnsCustomers()

        {

            // Arrange

            \_mockLogger = new Mock<ILogger<CustomerController>>();

            \_mockCustomerRepository = new Mock<IRepository<Customer>>();

            \_mockCustomerRepository.Setup(m => m.GetAll()).Returns(listCustomer);

            \_mockUnitOfWork = new Mock<IUnitOfWork>();

            \_mockUnitOfWork.Setup(x => x.customerRepository).Returns(\_mockCustomerRepository.Object);

            ICustomerController sut = new CustomerController(\_mockUnitOfWork.Object,

                                                             \_mockLogger.Object);

            // Act

            var result = sut.GetCustomers();

            var okResult = result as OkObjectResult;

            // Assert

            Assert.IsType<OkObjectResult>(okResult);

            Assert.Equal(3, ((System.Collections.Generic.List<TestProject.Models.Customer>)okResult.Value).Count);

        }

        [Theory]

        [InlineData(1)]

        [InlineData(2)]

        [InlineData(3)]

        public void Get\_WhenCalled\_ReturnsCustomer(int value)

        {

            // Arrange

            \_mockLogger = new Mock<ILogger<CustomerController>>();

            \_mockCustomerRepository = new Mock<IRepository<Customer>>();

            \_mockCustomerRepository.Setup(m => m.Get(value)).Returns(listCustomer.Where(c => c.CustomerId == value).FirstOrDefault);

            \_mockUnitOfWork = new Mock<IUnitOfWork>();

            \_mockUnitOfWork.Setup(x => x.customerRepository).Returns(\_mockCustomerRepository.Object);

            ICustomerController sut = new CustomerController(\_mockUnitOfWork.Object,

                                                             \_mockLogger.Object);

            // Act

            var result = sut.Get(value);

            var okResult = result as OkObjectResult;

            // Assert

            Assert.IsType<OkObjectResult>(okResult);

            Assert.Equal(value, ((TestProject.Models.Customer)((Microsoft.AspNetCore.Mvc.ObjectResult)result).Value).CustomerId);

        }

        [Fact]

        public void Create\_ValidObjectPassed\_ReturnsCreatedResponse()

        {

            // Arrange

            Customer testCustomer = new Customer

            {

                CustomerId = 10,

                Name = "John",

                AddressLine1 = "1 High St",

                AddressLine2 = "Burton",

                AddressLine3 = "UK"

            };

            \_mockLogger = new Mock<ILogger<CustomerController>>();

            \_mockCustomerRepository = new Mock<IRepository<Customer>>();

            \_mockCustomerRepository.Setup(m => m.Create(testCustomer));

            \_mockUnitOfWork = new Mock<IUnitOfWork>();

            \_mockUnitOfWork.Setup(x => x.customerRepository).Returns(\_mockCustomerRepository.Object);

            ICustomerController sut = new CustomerController(\_mockUnitOfWork.Object,

                                                             \_mockLogger.Object);

            // Act

            var result = sut.CreateCustomer(testCustomer);

            var createdAtRoute = result as CreatedAtRouteResult;

            // Assert

            Assert.IsType<CreatedAtRouteResult>(createdAtRoute);

            Assert.Equal("CustomerById", createdAtRoute.RouteName);

            Assert.Equal(testCustomer.CustomerId, ((TestProject.Models.Customer)((Microsoft.AspNetCore.Mvc.ObjectResult)result).Value).CustomerId);

        }

        [Theory]

        [InlineData(1)]

        public void Delete\_WhenCalled\_ReturnsOkResult(int value)

        {

            // Arrange

            Customer testCustomer = new Customer

            {

                CustomerId = 10,

                Name = "John",

                AddressLine1 = "1 High St",

                AddressLine2 = "Burton",

                AddressLine3 = "UK"

            };

            \_mockLogger = new Mock<ILogger<CustomerController>>();

            \_mockCustomerRepository = new Mock<IRepository<Customer>>();

            \_mockCustomerRepository.Setup(m => m.Delete(value));

            \_mockUnitOfWork = new Mock<IUnitOfWork>();

            \_mockUnitOfWork.Setup(x => x.customerRepository).Returns(\_mockCustomerRepository.Object);

            ICustomerController sut = new CustomerController(\_mockUnitOfWork.Object,

                                                             \_mockLogger.Object);

            // Act

            var result = sut.DeleteCustomer(value);

            var noContent = result as NoContentResult;

            // Assert

            Assert.IsType<NoContentResult>(noContent);

        }

    }

}