Lab 3_PRN231

Apply security, Ajax request with RESTful API and Web Application

Note: You have to change something when you create project and fullfil if need because you use .NET 6.0

Introduction

Imagine you're an employee of a product retailer named Product Store. Your manager has asked you to develop an application for simple product management. The relationship between Category and Product is One-to-Many, one product is belong to only one Category, one category will have zero or many products. The Product includes these properties: ProductId, ProductName, Category, UnitsInStock, UnitPrice. The Category includes properties: such as CategoryId, CategoryName. The application has to support adding, viewing, modifying, and removing products - a standardized usage action verbs better known as Create, Read, Update, Delete (CRUD).

This lab explores creating an application using ASP.NET Core Web API to create RESTful API, and ASP.NET Core Web App MVC and Identity and AJAX. An **SQL Server Database** will be created to persist the product data that will be used for reading and managing product data by **Entity Framework Core.**

Lab Objectives

In this lab, you will:

- Use the Visual Studio.NET to create ASP.NET Core Web Web API Project.
- Develop Web application using MVC Pattern combination with Identity and AJAX techniques.
- Use Entity Framework Core to create a SQL Server database (Forward Engineering Approach).
- Develop Entity classes, DBContext class, DAO class to perform CRUD actions using Entity Framework Core.
- Apply Repository pattern to develop application.
- Run the project and test the application actions.

Guidelines

Activity 01: Create a Blank Solution

Step 01. Create a Solution named **Lab03_IdetityAjax ASP.NETCoreWebAPI**.
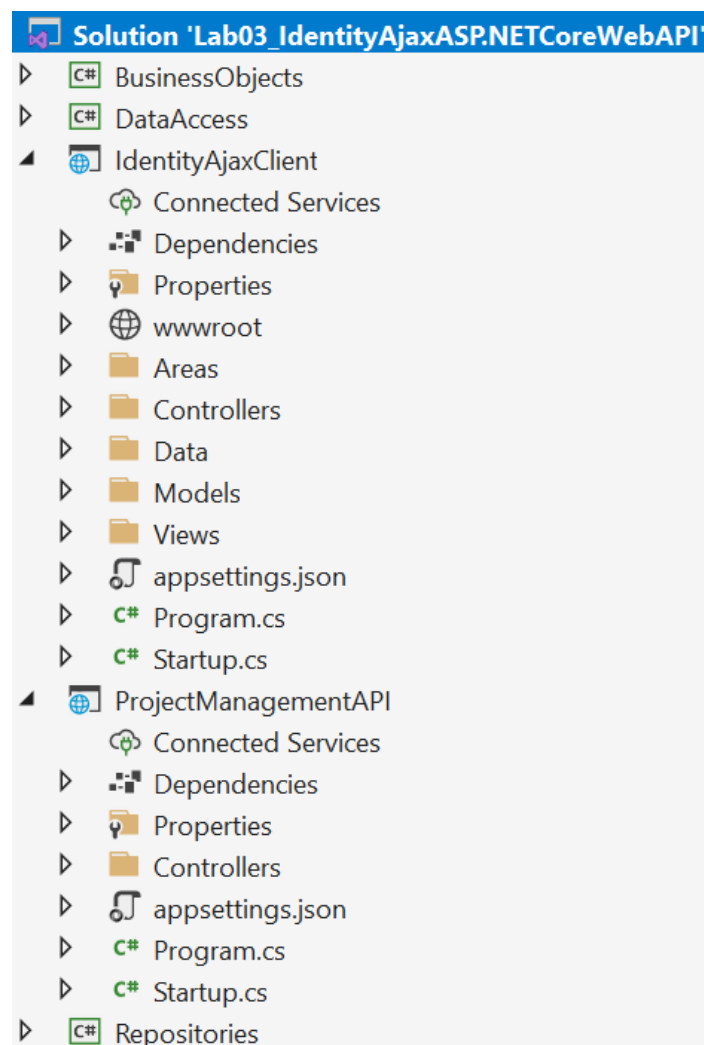
Step 02. Create Class Library Project: BusinessObjects.

Step 03. Create Class Library Project: Repositories.

Step 04. Create Class Library Project: DataAccess.

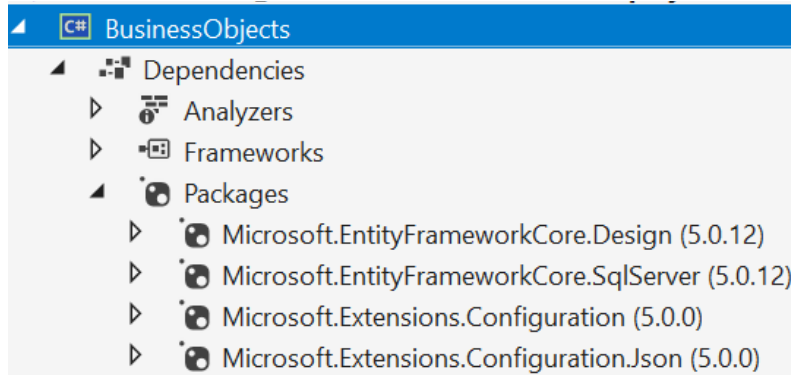**Step 05**. Create ASP.NET Core Web Web API Project.

Step 06. Create ASP.NET Core Web Application (Model-View-Controller) Project using Security with Identity.

Activity 02: BusinessObjects Project - Work with Entity Framework

Step 01. Create Class Library Project named BusinessObjects

Step 02. Install the following packages from NuGet:

Step 03. Add Connection string (also add JSON appsettings.json file)

```json
{
  "ConnectionStrings": {
    "MyStoreDB": "Server=(local);Uid=sa;Pwd=1234567890;Database=MyStoreDB"
  }
}
```

```xml
<ItemGroup>
  <None Update="appsettings.json">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </None>
</ItemGroup>
```

Step 04. Add "Products.cs", "Category.cs" entities, and the context class "ApplicationDBContext.cs"

```csharp
public class Category
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    8 references
    public int CategoryId { get; set; }
    [Required]
    [StringLength(40)]
    8 references
    public string CategoryName { get; set; }
    0 references
    public virtual ICollection<Product> Products { get; set; }
}
```

```csharp
public class Product
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    8 references
    public int ProductId { get; set; }
    [Required]
    [StringLength(40)]
    5 references
    public string ProductName { get; set; }
    [Required]
    0 references
    public int CategoryId { get; set; }
    [Required]
    3 references
    public int UnitsInStock { get; set; }
    [Required]
    5 references
    public decimal UnitPrice { get; set; }
    0 references
    public virtual Category Category { get; set; }
}

public class MyDbContext : DbContext
{
    6 references
    public MyDbContext() { }
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        var builder = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true);
        IConfigurationRoot configuration = builder.Build();
        optionsBuilder.UseSqlServer(configuration.GetConnectionString("MyStoreDB"));
    }
    1 reference
    public virtual DbSet<Category> Categories { get; set; }
    5 references
    public virtual DbSet<Product> Products { get; set; }

    0 references
    protected override void OnModelCreating(ModelBuilder optionsBuilder)
    {
        optionsBuilder.Entity<Category>().HasData(
            new Category { CategoryId = 1, CategoryName = "Beverages" },
            new Category { CategoryId = 2, CategoryName = "Condiments" },
            new Category { CategoryId = 3, CategoryName = "Confections" },
            new Category { CategoryId = 4, CategoryName = "Dairy Products" },
            new Category { CategoryId = 5, CategoryName = "Grains/Cereals" },
            new Category { CategoryId = 6, CategoryName = "Meat/Poultry" },
            new Category { CategoryId = 7, CategoryName = "Produce" },
            new Category { CategoryId = 8, CategoryName = "Seafood" }
        );
    }
}
```

Step 05. Add-Migration and Update-Database

> *dotnet ef migrations add "InitialDB"*

> *dotnet ef database update*

Activity 03: DataAccess Project - contain methods for accessing the underlying database

Step 01. Create Class Library Project named DataAccess

Step 02. Add Project reference: BusinessObjects Project

Step 03. Add data access classes for Product and Category

```csharp
public class CategoryDAO
{
    1 reference
    public static List<Category> GetCategories()
    {
        var listCategories = new List<Category>();
        try
        {
            using (var context = new MyDbContext())
            {
                listCategories = context.Categories.ToList();
            }
        }
        catch (Exception e)
        {
            throw new Exception(e.Message);
        }
        return listCategories;
    }
}

public class ProductDAO
{
    1 reference
    public static List<Product> GetProducts()...
    1 reference
    public static Product FindProductById(int prodId)...
    1 reference
    public static void SaveProduct(Product p)...
    1 reference
    public static void UpdateProduct(Product p)...
    1 reference
    public static void DeleteProduct(Product p)...
}
```

The detail functions for ProductDAO.cs

```csharp
public static List<Product> GetProducts()
{
    var listProducts = new List<Product>();
    try
    {
        using (var context = new MyDbContext())
        {
            listProducts = context.Products.ToList();
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
    return listProducts;
}

public static Product FindProductById(int prodId)
{
    Product p = new Product();
    try
    {
        using (var context = new MyDbContext())
        {
            p = context.Products.SingleOrDefault(x=> x.ProductId==prodId);
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
    return p;
}

public static void SaveProduct(Product p)
{
    try
    {
        using (var context = new MyDbContext())
        {
            context.Products.Add(p);
            context.SaveChanges();
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
}
```

```csharp
public static void UpdateProduct(Product p)
{
    try
    {
        using (var context = new MyDbContext())
        {
            context.Entry<Product>(p).State =
                Microsoft.EntityFrameworkCore.EntityState.Modified;
            context.SaveChanges();
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
}

public static void DeleteProduct(Product p)
{
    try
    {
        using (var context = new MyDbContext())
        {
            var p1 = context.Products.SingleOrDefault(
                            c => c.ProductId == p.ProductId);
            context.Products.Remove(p1);

            context.SaveChanges();
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
}
```

Activity 04: Class Library Repositories Project - create an abstraction layer between the Data Access Layer and the Business Logic Layer of the application

Step 01. Create Class Library Project named Repositories

Step 02. Add Project reference: BusinessObjects, DataAccess Projects

Step 03. Create IProductRepository Interface

```csharp
public interface IProductRepository
{
    2 references
    void SaveProduct(Product p);
    3 references
    Product GetProductById(int id);
    2 references
    void DeleteProduct(Product p);
    2 references
    void UpdateProduct(Product p);
    1 reference
    List<Category> GetCategories();
    2 references
    List<Product> GetProducts();
}
```

Step 04. Create ProductRepository class implements IProductRepository Interface

```csharp
public class ProductRepository : IProductRepository
{
    2 references
    public void DeleteProduct(Product p) => ProductDAO.DeleteProduct(p);
    2 references
    public void SaveProduct(Product p) => ProductDAO.SaveProduct(p);
    2 references
    public void UpdateProduct(Product p) => ProductDAO.UpdateProduct(p);
    1 reference
    public List<Category> GetCategories() => CategoryDAO.GetCategories();
    2 references
    public List<Product> GetProducts() => ProductDAO.GetProducts();
    3 references
    public Product GetProductById(int id) => ProductDAO.FindProductById(id);
}
```

Activity 05: Create ProductManagementAPI Project (Work with ASP.NET Core Web API template)

Step 01. Create ASP.NET Core Web API Project named ProductManagementAPI

Step 02. Add Project reference: Repository Project

Step 03. Add ApiController named ProductsControllers.cs

```csharp
namespace ProjectManagementAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    0 references
    public class ProductsController : ControllerBase
    {
        private IProductRepository repository = new ProductRepository();

        //GET: api/Products
        [HttpGet]
        0 references
        public ActionResult<IEnumerable<Product>> GetProducts() => repository.GetProducts();


        // POST: ProductsController/Products
        [HttpPost]
        0 references
        public IActionResult PostProduct(Product p)...

        // GET: ProductsController/Delete/5
        [HttpDelete("id")]
        0 references
        public IActionResult DeleteProduct(int id)...

        [HttpPut("id")]
        0 references
        public IActionResult UpdateProduct(int id, Product p)...
    }
}
```

The detail of functions

```csharp
// POST: ProductsController/Products
[HttpPost]
0 references
public IActionResult PostProduct(Product p)
{
    repository.SaveProduct(p);
    return NoContent();
}
```

```csharp
// GET: ProductsController/Delete/5
[HttpDelete("id")]
0 references
public IActionResult DeleteProduct(int id)
{
    var p = repository.GetProductById(id);
    if (p == null)
        return NotFound();
    repository.DeleteProduct(p);
    return NoContent();
}


[HttpPut("id")]
0 references
public IActionResult UpdateProduct(int id, Product p)
{
    var pTmp = repository.GetProductById(id);
    if (p == null)
        return NotFound();
    repository.UpdateProduct(p);
    return NoContent();
}
```

Step 04. Create Web API Settings to allow Ajax request with Startup.cs

Configure with ConfigureServices() function

```csharp
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors();
    services.AddControllers();
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new OpenApiInfo {
            Title = "ProjectManagementAPI",
            Version = "v1" });
    });
}
```

Configure with Configure () function

```csharp
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())...
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this
        app.UseHsts();
    }

    app.UseCors(builder =>
    {
        builder
        .AllowAnyOrigin()
        .AllowAnyMethod()
        .AllowAnyHeader();
    });

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>...);
}
```

Step 05. Test API project with OpenAPI or Postman

Activity 06: Implement Identity and Ajax in ASP.NET Core Web Application with Model-View-Controller Project

Step 01. Create ASP.NET Core Web App (Model-View-Controller) named IdentityAjaxClient

Additional information

ASP.NET Core Web App (Model-View-Controller)   C#   Linux   macOS   Windows   Cloud   Service   Web

Target Framework ⓘ

.NET 5.0 (Current)

Authentication Type ⓘ

Individual Accounts

☐ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ

Linux

☐ Enable Razor runtime compilation ⓘ

Step 02. Add Project reference: BusinessObjects Project (or create new DTO classes)

Step 03. Create Controller to connect to ProductManagementAPI

Simple create ProductController with [Authorize] and action methods return View. All actions will work with Ajax requests in View

```csharp
[Authorize]
0 references
public class ProductController : Controller
{
    // GET: ProductController
    3 references
    public ActionResult Index()...

    // GET: ProductController/Details/5
    0 references
    public ActionResult Details(int id)...

    // GET: ProductController/Create
    0 references
    public ActionResult Create()...

    // POST: ProductController/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public ActionResult Create(IFormCollection collection)...

    // GET: ProductController/Edit/5
    0 references
    public ActionResult Edit(int id)...

    // POST: ProductController/Edit/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public ActionResult Edit(int id, IFormCollection collection)...

    // GET: ProductController/Delete/5
    0 references
    public ActionResult Delete(int id)
    {
        return View();
    }

    // POST: ProductController/Delete/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public ActionResult Delete(int id, IFormCollection collection)...
}
```

Step 04. Create View with Ajax

```html
<div class="container-fluid">
    <h2>Product List</h2>
    <table class="table table-sm table-striped table-bordered m-2">
        <thead>
            <tr>
                <th>ProductID</th>
                <th>Product Name</th>
                <th>Quatity</th>
                <th>Unit Price</th>
                <th>Update</th>
                <th>Delete</th>
            </tr>
        </thead>
        <tbody></tbody>
    </table>
</div>

<script type="text/javascript">
    $(document).ready(function () {
        ShowAllProducts();

        function ShowAllProducts() {
            $("table tbody").html("");
            $.ajax({
                url: "http://localhost:53633/api/Products",
                type: "get",
                contentType: "application/json; charset=utf-8",
                dataType: "json",
                success: function (result, status, xhr) {
                    $.each(result, function (index, value) {
                        $("tbody").append($("<tr>"));
                        appendElement = $("tbody tr").last();
                        appendElement.append($("<td>").html(value["productId"]));
                        appendElement.append($("<td>").html(value["productName"]));
                        appendElement.append($("<td>").html(value["unitsInStock"]));
                        appendElement.append($("<td>").html(value["unitPrice"]));
                        appendElement.append($("<td>").html("<a href=\"?id=" +
                            value["productId"] + "\"><img src=\"icon/edit.png\" /></a>"));
                        appendElement.append($("<td>").html("<img class=\"delete\" src=\"icon/close.png\" />"));
                    });
                },
                error: function (xhr, status, error) {
                    console.log(xhr)
                }
            });
        }
```

> URL get from ASP.NET Core Web API

```javascript
        $("table").on("click", "img.delete", function () {
            var productId = $(this).parents("tr").find("td:nth-child(1)").text();

            $.ajax({
                url: "http://localhost:53633/api/Products/" + productId,
                type: "delete",
                contentType: "application/json",
                success: function (result, status, xhr) {
                    ShowAllProducts();
                },
                error: function (xhr, status, error) {
                    console.log(xhr)
                }
            });
        });

    });
</script>
```

Step 05. Test the function of Web Client

Registration UI

IdentityAjaxClient   Home   Privacy   Register   Login

Register
Create a new account.

Email

Password

Confirm password

Register

Login UI

Log in

Use a local account to log in.

Email

Password

☐ Remember me?

Log in

List Product UI

IdentityAjaxClient   Home   Privacy   Hello admin@fstore.com!   Logout

Product List

ProductID	Product Name	Quatity	Unit Price	Update	Delete
2	Chai	12	20	✎	⊗
3	Chang	30	40	✎	⊗
4	Aniseed Syrup	15	60	✎	⊗
5	Chef Anton's Cajun Seasoning	40	10	✎	⊗

© 2022 - IdentityAjaxClient - Privacy

Activity 07: Build and run Project. Test all CRUD actions

Note: Choose the option for multiple startup projects.