**Lab 02. Building an ASP.NET Core Web API with OData**

# 1. Introduction

Imagine you're an employee of a store named **CosmeticsStore**. Your manager has asked you to develop a Web application for product management. The application has to support adding, viewing, modifying, and removing products—a standardized usage action verbs better known as Create, Read, Update, Delete (CRUD).

* Cosmetic Name: The name of the cosmetic product.
* Ingredients: List of ingredients used in the cosmetic product.
* Skin Type: Suitable skin types (e.g., oily, dry, sensitive, combination).
* Size: Size or volume of the product (e.g., 30ml, 15g).
* Price: Retail price of the product.
* ExpirationDate: Shelf life or expiration date.
* *Category Name: The name of the category (e.g., skincare, makeup, haircare).*
* Subcategories: Specific types within the main category (e.g., under makeup: foundation, lipstick, eyeshadow).
* Usage Purpose: Intended use or function (e.g., moisturizing, coverage, enhancing).
* Formulation Type: How the product is formulated (e.g., liquid, cream, powder, gel).

This lab explores creating an application using ASP.NET Core Web App (Razor Pages). An **SQL Server** **Database** will be created to persist the car's data that will be used for reading and managing product data by **Entity Framework Core**.

Authentication/Authorization - Note that: *Role: Administrator=1; Manager = 2; Staff=3; Member= 4.*

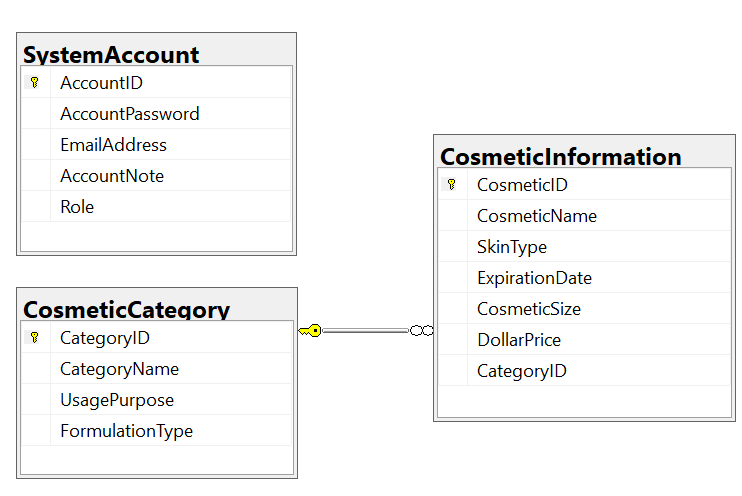
* *Administrator role will have a permission to all actions (CRUD and search).*
* *Staff or Member role will have only permission to search information.*
* *Other roles do not have any permissions.*

# 2. Lab Objectives

In this lab, you will:

* Use the Visual Studio.NET to create **ASP.NET Core Web API with OData** and Class Library (.dll) projects.
* Create a SQL Server database named CosmeticsDB that has a CosmeticInformation, CosmeticCategory, SystemAccount tables.
* Apply Repository pattern in a project.
* Work with Realtime Communication Web application
* Run the project and test the application actions.

# 3. Database Design (CosmeticsDB)

****

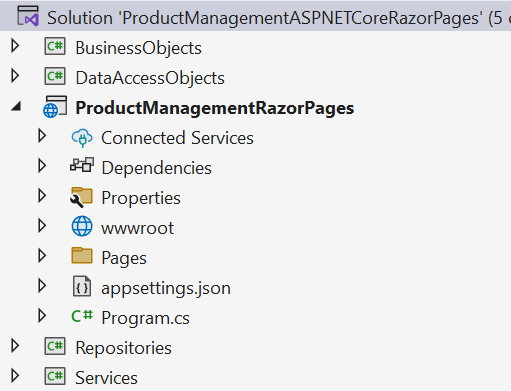
# Activity 01: Build a solution by Visual Studio.NET

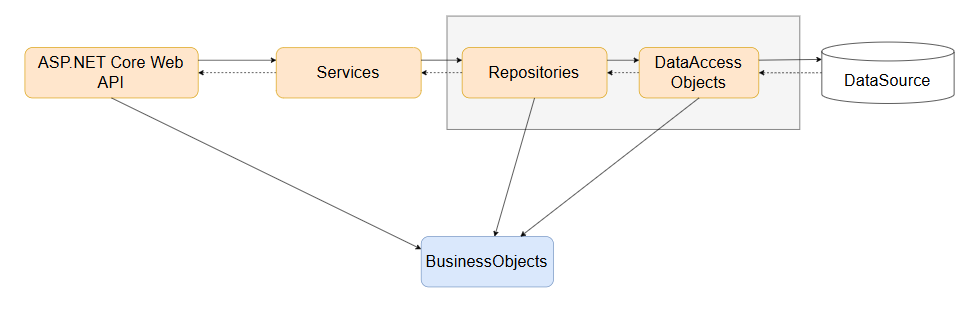
Create a Blank Solution named **Lab01.sln** then add new a **Class Library** project named **BusinessObjects, DataAccessObjects, Repositories, Services** and an ASP.NET Core Web App (Razor Pages) project named **ProductManagementRazorPages**

## **Step 01**. Create a Blank solution.

## **Step 02.** Create 4 **Class Library** projects.

## **Step 03.** Create a project (ASP.NET Core Web App (Razor Pages).



****

Note:

* **Data Source** in this case is the SQL Server Database
* **Services Project** – This project represents a layer or component responsible for implementing the business logic of an application.
* **Repository Project** – This project provides an abstraction layer between the application’s business logic and the underlying data source.
* **Data Access Layer Project** – This project used to abstract and encapsulate the logic for accessing data from a data source, such as a database.

# Activity 02: Write codes for the BusinessObjects project

## **Step 01**. Install the following packages from NuGet:

* Microsoft.EntityFrameworkCore.SqlServer --version 8.0.2
* Microsoft.EntityFrameworkCore.Tools --version 8.0.2
* Microsoft.Extensions.Configuration.Json --version 8.0.0

Check the tool for EFCore (install/uninstall tool if needed) (dotnet SDK 8.0.202)

dotnet tool install --global dotnet-ef --version 8.0.2

dotnet tool uninstall --global dotnet-ef

## **Step 02**. Right-click on project , select **Open In Terminal.** On **Developer PowerShell** dialog execute the following commands to generate model:

* Implement ORM

dotnet ef dbcontext scaffold "Server=(local);uid=sa;pwd=12345;database= **CosmeticsDB; TrustServerCertificate=True" Microsoft.EntityFrameworkCore.SqlServer --output-dir Entities**

* Change the connection string in OnConfiguring() function of MyStoreContext.cs

using System.IO;

         using Microsoft.Extensions.Configuration.Json;

*private string GetConnectionString()*

*{*

*IConfiguration configuration = new ConfigurationBuilder()*

*.SetBasePath(Directory.GetCurrentDirectory())*

*.AddJsonFile("appsettings.json", true, true).Build();*

*return configuration["ConnectionStrings:DefaultConnectionString"];*

*}*

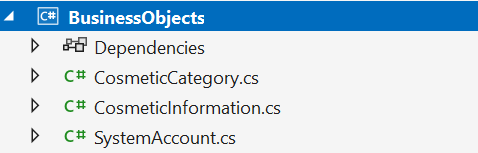
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

        {

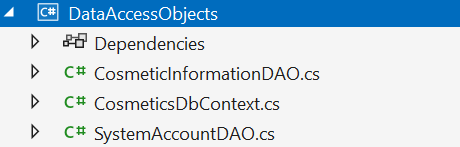
***optionsBuilder.UseSqlServer(GetConnectionString());***

         }

* Move the *CosmeticsDbContext*.cs to DataAccessLayer Project



# Activity 03: Write codes for the DataAccessLayer project



## **Step 01.** On the **DataAccessObjects** project, add a class named **CosmeticInformationDAO.cs** and write codes as follows:

using BusinessObjects;

using Microsoft.EntityFrameworkCore;

namespace DataAccessObjects;

public class CosmeticInformationDAO

{

private static CosmeticInformationDAO? instance = null;

private CosmeticInformationDAO() { }

public static CosmeticInformationDAO Instance

{

get

{

if (instance == null)

{

instance = new CosmeticInformationDAO();

}

return instance;

}

}

public async Task<List<CosmeticInformation>> GetAllCosmetics()

{

try

{

using (var context = new CosmeticsDbContext())

{

var listComestics = await context.CosmeticInformations.Include(x => x.Category).ToListAsync();

return listComestics;

}

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<List<CosmeticCategory>> GetAllCategories()

{

using (var context = new CosmeticsDbContext())

{

var listCategories = await context.CosmeticCategories.ToListAsync();

return listCategories;

}

}

public async Task<CosmeticInformation> AddCosmeticInformation(CosmeticInformation cosmeticInformation)

{

using (var context = new CosmeticsDbContext())

{

var categoryObject = await context.CosmeticCategories.FirstOrDefaultAsync(x => x.CategoryId.Equals(cosmeticInformation.CategoryId));

if (categoryObject == null)

{

throw new Exception("Category is not found");

}

cosmeticInformation.CosmeticId = GenerateId();

await context.CosmeticInformations.AddAsync(cosmeticInformation);

await context.SaveChangesAsync();

return cosmeticInformation;

}

}

public async Task<CosmeticInformation> GetById(string id)

{

using (var context = new CosmeticsDbContext())

{

var resultObject = await context.CosmeticInformations.Include(x => x.Category)

.FirstOrDefaultAsync(x => x.CosmeticId.Equals(id));

return resultObject;

}

}

public async Task<CosmeticInformation> Update(CosmeticInformation cosmeticInformation)

{

using (var context = new CosmeticsDbContext())

{

var updateObject = await context.CosmeticInformations.FirstOrDefaultAsync(x => x.CosmeticId == cosmeticInformation.CosmeticId);

if (updateObject == null)

{

throw new Exception("CosmeticInformations not found");

}

var cate = await context.CosmeticCategories.FirstOrDefaultAsync(x => x.CategoryId.Equals(cosmeticInformation.CategoryId));

if (cate == null)

{

throw new Exception("Cate not found");

}

updateObject.CosmeticName = cosmeticInformation.CosmeticName;

updateObject.SkinType = cosmeticInformation.SkinType;

updateObject.ExpirationDate = cosmeticInformation.ExpirationDate;

updateObject.CosmeticSize = cosmeticInformation.CosmeticSize;

updateObject.DollarPrice = cosmeticInformation.DollarPrice;

updateObject.CategoryId = cosmeticInformation.CategoryId;

context.CosmeticInformations.Update(updateObject);

await context.SaveChangesAsync();

return updateObject;

}

}

public async Task<CosmeticInformation> Delete(string id)

{

using (var context = new CosmeticsDbContext())

{

var deleteObject = await context.CosmeticInformations.FirstOrDefaultAsync(p => p.CosmeticId.Equals(id));

if (deleteObject == null)

{

throw new Exception("CosmeticInformations not found");

}

context.CosmeticInformations.Remove(deleteObject);

await context.SaveChangesAsync();

return deleteObject;

}

}

private string GenerateId()

{

var random = new Random();

var id = random.Next(100000, 999999);

return "PL" + id.ToString();

}

}

## **Step 02**. On the **DataAccessObjects** project, add a class named **SystemAccounrDAO.cs** and write codes as follows:

using BusinessObjects;

using Microsoft.EntityFrameworkCore;

namespace DataAccessObjects

{

public class SystemAccountDAO

{

private static SystemAccountDAO? instance = null;

private SystemAccountDAO() { }

public static SystemAccountDAO Instance

{

get

{

if (instance == null)

{

instance = new SystemAccountDAO();

}

return instance;

}

}

public async Task<SystemAccount> Login(string email, string password)

{

using (var context = new CosmeticsDbContext())

{

var account = await context.SystemAccounts.FirstOrDefaultAsync(account => account.EmailAddress == email && account.AccountPassword == password);

return account;

}

}

}

}

## **Step 03**. The codes for **CosmeticsDbContext.cs**:

using System;

using System.Collections.Generic;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using BusinessObjects;

namespace DataAccessObjects;

public partial class CosmeticsDbContext : DbContext

{

public CosmeticsDbContext()

{

}

public CosmeticsDbContext(DbContextOptions<CosmeticsDbContext> options)

: base(options)

{

}

public virtual DbSet<CosmeticCategory> CosmeticCategories { get; set; }

public virtual DbSet<CosmeticInformation> CosmeticInformations { get; set; }

public virtual DbSet<SystemAccount> SystemAccounts { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

=> optionsBuilder.UseSqlServer(GetConnectionString());

private string GetConnectionString()

{

IConfiguration configuration = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json", true, true).Build();

return configuration["ConnectionStrings:DefaultConnectionString"];

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<CosmeticCategory>(entity =>

{

entity.HasKey(e => e.CategoryId).HasName("PK\_\_Cosmetic\_\_19093A2BA2750A62");

entity.ToTable("CosmeticCategory");

entity.Property(e => e.CategoryId)

.HasMaxLength(30)

.HasColumnName("CategoryID");

entity.Property(e => e.CategoryName).HasMaxLength(120);

entity.Property(e => e.FormulationType).HasMaxLength(250);

entity.Property(e => e.UsagePurpose).HasMaxLength(250);

});

modelBuilder.Entity<CosmeticInformation>(entity =>

{

entity.HasKey(e => e.CosmeticId).HasName("PK\_\_Cosmetic\_\_98ED527E67429B69");

entity.ToTable("CosmeticInformation");

entity.Property(e => e.CosmeticId)

.HasMaxLength(30)

.HasColumnName("CosmeticID");

entity.Property(e => e.CategoryId)

.HasMaxLength(30)

.HasColumnName("CategoryID");

entity.Property(e => e.CosmeticName).HasMaxLength(160);

entity.Property(e => e.CosmeticSize).HasMaxLength(400);

entity.Property(e => e.DollarPrice).HasColumnType("decimal(18, 0)");

entity.Property(e => e.ExpirationDate).HasMaxLength(160);

entity.Property(e => e.SkinType).HasMaxLength(200);

entity.HasOne(d => d.Category).WithMany(p => p.CosmeticInformations)

.HasForeignKey(d => d.CategoryId)

.OnDelete(DeleteBehavior.Cascade)

.HasConstraintName("FK\_\_CosmeticI\_\_Categ\_\_3C69FB99");

});

modelBuilder.Entity<SystemAccount>(entity =>

{

entity.HasKey(e => e.AccountId).HasName("PK\_\_SystemAc\_\_349DA58646609DFA");

entity.ToTable("SystemAccount");

entity.HasIndex(e => e.EmailAddress, "UQ\_\_SystemAc\_\_49A14740597553D5").IsUnique();

entity.Property(e => e.AccountId)

.ValueGeneratedNever()

.HasColumnName("AccountID");

entity.Property(e => e.AccountNote).HasMaxLength(240);

entity.Property(e => e.AccountPassword).HasMaxLength(100);

entity.Property(e => e.EmailAddress).HasMaxLength(100);

});

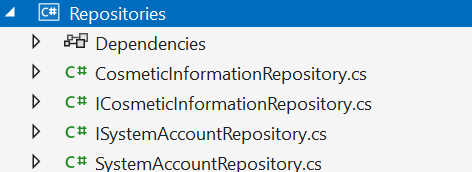
OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

# Activity 04: Write codes for the Repositories project



## **Step 01.** On the **Repositories** project, add an interface named **ICosmeticInformationRepository.cs** and write codes as follows:

## **Step 02**. On the **Repositories** project, add an interface named **IProductRepository.cs** and write codes as follows:

using BusinessObjects;

namespace Repositories

{

public interface ICosmeticInformationRepository

{

Task<List<CosmeticInformation>> GetAllCosmetics();

Task<CosmeticInformation> GetOne(string id);

Task<CosmeticInformation> Add(CosmeticInformation cosmeticInformation);

Task<CosmeticInformation> Update(CosmeticInformation cosmeticInformation);

Task<CosmeticInformation> Delete(string id);

Task<List<CosmeticCategory>> GetAllCategories();

}

}

## **Step 03**. On the **Repositories** project, add an interface named **ISystemAccountRepository.cs** and write codes as follows:

using BusinessObjects;

namespace Repositories

{

public interface ISystemAccountRepository

{

Task<SystemAccount> Login(string email, string password);

}

}

## **Step 04**. Write codes for class **CosmeticInformationRepository.cs** as follows:

using BusinessObjects;

using DataAccessObjects;

namespace Repositories

{

public class CosmeticInformationRepository : ICosmeticInformationRepository

{

public async Task<CosmeticInformation> Add(CosmeticInformation cosmeticInformation)

{

return await CosmeticInformationDAO.Instance.AddCosmeticInformation(cosmeticInformation);

}

public async Task<CosmeticInformation> Delete(string id)

{

return await CosmeticInformationDAO.Instance.Delete(id);

}

public async Task<List<CosmeticCategory>> GetAllCategories()

{

return await CosmeticInformationDAO.Instance.GetAllCategories();

}

public async Task<List<CosmeticInformation>> GetAllCosmetics()

{

return await CosmeticInformationDAO.Instance.GetAllCosmetics();

}

public async Task<CosmeticInformation> GetOne(string id)

{

return await CosmeticInformationDAO.Instance.GetById(id);

}

public async Task<CosmeticInformation> Update(CosmeticInformation cosmeticInformation)

{

return await CosmeticInformationDAO.Instance.Update(cosmeticInformation);

}

}

}

## **Step 05**. Write codes for class **SystemAccountRepository.cs** as follows:

using BusinessObjects;

using DataAccessObjects;

namespace Repositories

{

public class SystemAccountRepository : ISystemAccountRepository

{

public async Task<SystemAccount> Login(string email, string password)

{

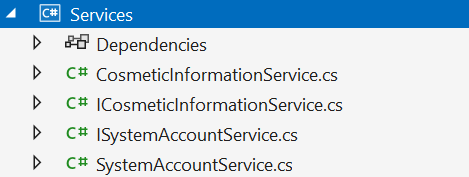
return await SystemAccountDAO.Instance.Login(email, password);

}

}

}

# Activity 05: Write codes for the Services project



## **Step 01.** On the **Services** project, add an interface named **ICosmeticInformationService.cs** and write codes as follows:

using BusinessObjects;

namespace Services

{

public interface ICosmeticInformationService

{

Task<List<CosmeticInformation>> GetAllCosmetics();

Task<CosmeticInformation> GetOne(string id);

Task<CosmeticInformation> Add(CosmeticInformation cosmeticInformation);

Task<CosmeticInformation> Update(CosmeticInformation cosmeticInformation);

Task<CosmeticInformation> Delete(string id);

Task<List<CosmeticCategory>> GetAllCategories();

}

}

## **Step 02**. On the **Services** project, add an interface named **ISystemAccountService.cs** and write codes as follows:

using BusinessObjects;

namespace Services

{

public interface ISystemAccountService

{

Task<SystemAccount> Login(string email, string password);

}

}

## **Step 03**. Write codes for class **CosmeticInformationService.cs** as follows:

using BusinessObjects;

using Repositories;

namespace Services

{

public class CosmeticInformationService : ICosmeticInformationService

{

private readonly ICosmeticInformationRepository \_repository;

public CosmeticInformationService(ICosmeticInformationRepository repository)

{

\_repository = repository;

}

public async Task<CosmeticInformation> Add(CosmeticInformation cosmeticInformation)

{

return await \_repository.Add(cosmeticInformation);

}

public async Task<CosmeticInformation> Delete(string id)

{

return await \_repository.Delete(id);

}

public async Task<List<CosmeticCategory>> GetAllCategories()

{

return await \_repository.GetAllCategories();

}

public async Task<List<CosmeticInformation>> GetAllCosmetics()

{

return await \_repository.GetAllCosmetics();

}

public async Task<CosmeticInformation> GetOne(string id)

{

return await \_repository.GetOne(id);

}

public async Task<CosmeticInformation> Update(CosmeticInformation cosmeticInformation)

{

return await \_repository.Update(cosmeticInformation);

}

}

}

## **Step 04**. Write codes for class **SystemAccountService.cs** as follows:

using BusinessObjects;

using Repositories;

namespace Services

{

public class SystemAccountService : ISystemAccountService

{

private readonly ISystemAccountRepository \_repo;

public SystemAccountService(ISystemAccountRepository repo)

{

\_repo = repo;

}

public async Task<SystemAccount> Login(string email, string password)

{

return await \_repo.Login(email, password);

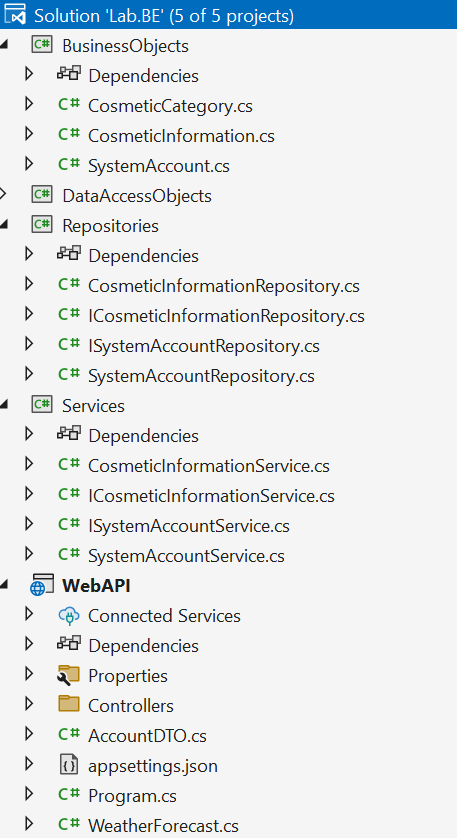
}

}

}

# Activity 06: Work with ASP.NET Core Web App (Razor Pages)

## **Step 01.** Create the **ASP.NET Core Web API** project, the result as the following



## **Step 02**. Add connection string to **appsettings.json**.

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"DefaultConnectionString": "Server=(local); Database= CosmeticsDB; Uid=sa; Pwd=1234567890;TrustServerCertificate=True"

},

"Jwt": {

"SecretKey": "SecretKeySecretKeySecretKeySecretKeySecretKeySecretKeySecretKeySecretKeySecretKeySecretKeySecretKey",

"Issuer": "FU Lab Issuer",

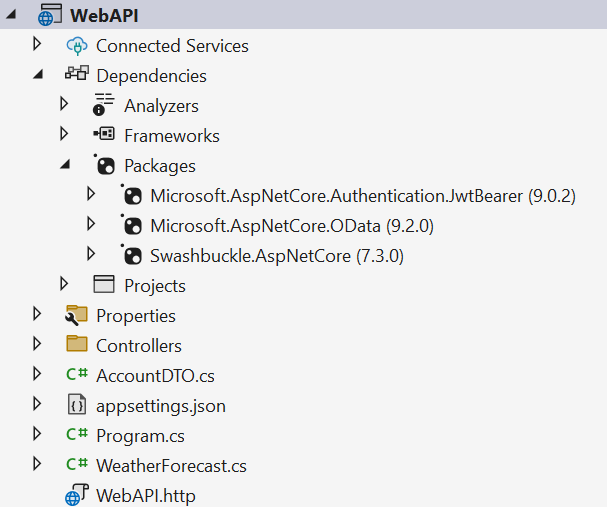
"Audience": "FU Lab Audience"

}

}

## **Step 03.** Add **Business Objects** and **Services** projects as references for the ASP.NET Core Web API

## **Step 04.** Add **packages for** OData and JwtBearer

****

## **Step 05.** Create DTO class for Login processing

AccountDTO.cs (AccountRequestDTO for sending objects đi, AccounResponsetDTO (with Token for responding object to Client)

namespace WebAPI

{

public class AccountRequestDTO

{

public string Email { get; set; }

public string Password { get; set; }

}

public class AccountResponseDTO

{

public string Token { get; set; }

public string Role { get; set; }

public string AccountId { get; set; }

}

}

## **Step 06.** Program.cs – Configure OData, JWT and other services

using BusinessObjects;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.OData;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OData.ModelBuilder;

using Microsoft.OpenApi.Models;

using Repositories;

using Services;

using System.Text;

using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

var modelBuilder = new ODataConventionModelBuilder();

modelBuilder.EntitySet<CosmeticInformation>("CosmeticInformations");

modelBuilder.EntitySet<CosmeticCategory>("CosmeticCategories");

// Add services to the container.

builder.Services.AddControllers()

.AddJsonOptions(options =>

{

options.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles;

options.JsonSerializerOptions.DefaultIgnoreCondition = JsonIgnoreCondition.Never;

})

.AddOData(

options => options.Select().Filter().OrderBy().Expand().Count().SetMaxTop(null).AddRouteComponents(

"odata",

modelBuilder.GetEdmModel()));

// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi

//builder.Services.AddOpenApi();

IConfiguration configuration = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json", true, true).Build();

builder.Services.AddScoped<ISystemAccountRepository, SystemAccountRepository>();

builder.Services.AddScoped<ICosmeticInformationRepository, CosmeticInformationRepository>();

builder.Services.AddScoped<ISystemAccountService, SystemAccountService>();

builder.Services.AddScoped<ICosmeticInformationService, CosmeticInformationService>();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services

.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(x =>

{

x.SaveToken = true;

x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = false,

ValidateAudience = false,

ValidateLifetime = false,

ValidIssuer = configuration["JWT:Issuer"],

ValidAudience = configuration["JWT:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["JWT:SecretKey"]))

};

});

// Add Swagger JWT configuration

builder.Services.AddSwaggerGen(c =>

{

var jwtSecurityScheme = new OpenApiSecurityScheme

{

Name = "JWT Authentication",

Description = "JWT Authentication for Cosmetics Management",

In = ParameterLocation.Header,

Type = SecuritySchemeType.Http,

Scheme = "Bearer",

BearerFormat = "JWT"

};

c.AddSecurityDefinition("Bearer", jwtSecurityScheme);

var securityRequirement = new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] {}

}

};

c.AddSecurityRequirement(securityRequirement);

});

builder.Services.AddAuthorization(options =>

{

options.AddPolicy("AdminOnly",

policyBuilder => policyBuilder.RequireAssertion(

context => context.User.HasClaim(claim => claim.Type == "Role") &&

context.User.FindFirst(claim => claim.Type == "Role").Value == "1"));

options.AddPolicy("AdminOrStaffOrMember",

policyBuilder => policyBuilder.RequireAssertion(

context => context.User.HasClaim(claim => claim.Type == "Role")

&& (context.User.FindFirst(claim => claim.Type == "Role").Value == "1"

|| context.User.FindFirst(claim => claim.Type == "Role").Value == "3"

|| context.User.FindFirst(claim => claim.Type == "Role").Value == "4")));

});

var app = builder.Build();

app.UseRouting();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

//app.MapOpenApi();

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseAuthorization();

app.MapControllers();

app.Run();

*Role: Administrator=1; Manager = 2; Staff=3; Member= 4.*

* *Administrator role will have a permission to all actions (CRUD and search).*
* *Staff or Member role will have only permission to search information.*
* *Other roles do not have any permissions.*

## **Step 07.** Controller for processing login process. This controller has 4 type of action methods: POST, PUT, DELETE, GET

using Microsoft.AspNetCore.Mvc;

using Services;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace WebAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class SystemAccountsController : ControllerBase

{

private readonly ISystemAccountService \_systemAccountService;

public SystemAccountsController(ISystemAccountService systemAccountService)

{

\_systemAccountService = systemAccountService;

}

// POST: api/SystemAccounts

// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754

[HttpPost("Login")]

public async Task<ActionResult> Login([FromBody] AccountRequestDTO loginDTO)

{

try

{

var account = await \_systemAccountService.Login(loginDTO.Email, loginDTO.Password);

if (account == null)

{

return Unauthorized("Invalid email or password.");

}

IConfiguration configuration = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json", true, true).Build();

var claims = new List<Claim>

{

new Claim(ClaimTypes.Email, account.EmailAddress),

new Claim("Role", account.Role.ToString()),

new Claim("AccountId", account.AccountId.ToString()),

};

var symetricKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["JWT:SecretKey"]));

var signCredential = new SigningCredentials(symetricKey, SecurityAlgorithms.HmacSha256);

var preparedToken = new JwtSecurityToken(

issuer: configuration["JWT:Issuer"],

audience: configuration["JWT:Audience"],

claims: claims,

expires: DateTime.Now.AddMinutes(16),

signingCredentials: signCredential);

var generatedToken = new JwtSecurityTokenHandler().WriteToken(preparedToken);

var role = account.Role.ToString();

var accountId = account.AccountId.ToString();

return Ok(new AccountResponseDTO

{

Role = role,

Token = generatedToken,

AccountId = accountId

});

}

catch (Exception e)

{

return BadRequest(e.Message);

}

}

}

}

## **Step 08.** Controller for CRUD cosmetics information.

using Microsoft.AspNetCore.Mvc;

using BusinessObjects;

using Services;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.OData.Query;

using Microsoft.AspNetCore.OData.Routing.Controllers;

namespace WebAPI.Controllers

{

//[Route("api/[controller]")]

[ApiController]

public class CosmeticInformationsController : ODataController

{

private readonly ICosmeticInformationService \_cismeticInformationService;

public CosmeticInformationsController(ICosmeticInformationService cismeticInformationService)

{

\_cismeticInformationService = cismeticInformationService;

}

[EnableQuery]

[Authorize(Policy = "AdminOrStaffOrMember")]

[HttpGet("/api/CosmeticInformations")]

public async Task<ActionResult<IEnumerable<CosmeticInformation>>> GetCosmeticInformations()

{

try

{

var result = await \_cismeticInformationService.GetAllCosmetics();

return Ok(result);

}

catch (Exception ex)

{

return StatusCode(400, $"{ex.Message}");

}

}

[Authorize(Policy = "AdminOrStaffOrMember")]

[HttpGet("/api/CosmeticCategories")]

public async Task<ActionResult<List<CosmeticCategory>>> GetCategories()

{

try

{

var result = await \_cismeticInformationService.GetAllCategories();

return Ok(result);

}

catch (Exception ex)

{

return StatusCode(400, $"{ex.Message}");

}

}

[Authorize(Policy = "AdminOnly")]

[HttpPost("/api/CosmeticInformations")]

public async Task<ActionResult<CosmeticInformation>> AddCosmeticInformation([FromBody] CosmeticInformation cosmeticInformation)

{

try

{

var result = await \_cismeticInformationService.Add(cosmeticInformation);

return Ok(result);

}

catch (Exception ex)

{

return StatusCode(400, $"{ex.Message}");

}

}

[Authorize(Policy = "AdminOnly")]

[HttpPut("/api/CosmeticInformations/{id}")]

public async Task<ActionResult<CosmeticInformation>> UpdateCosmeticInformation(string id, [FromBody] CosmeticInformation cosmeticInformation)

{

try

{

cosmeticInformation.CosmeticId = id;

var result = await \_cismeticInformationService.Update(cosmeticInformation);

return Ok(result);

}

catch (Exception ex)

{

return StatusCode(400, $"{ex.Message}");

}

}

[Authorize(Policy = "AdminOnly")]

[HttpDelete("/api/CosmeticInformations/{id}")]

public async Task<ActionResult<CosmeticInformation>> DeleteCosmeticInformation(string id)

{

try

{

var result = await \_cismeticInformationService.Delete(id);

return Ok(result);

}

catch (Exception ex)

{

return StatusCode(400, $"{ex.Message}");

}

}

[Authorize(Policy = "AdminOrStaffOrMember")]

[HttpGet("/api/CosmeticInformations/{id}")]

public async Task<ActionResult<CosmeticInformation>> AddCosmeticInformation(string id)

{

try

{

var result = await \_cismeticInformationService.GetOne(id);

return Ok(result);

}

catch (Exception ex)

{

return StatusCode(400, $"{ex.Message}");

}

}

}

}

# Activity 07: Run the ASP.NET Core Web API project and test all actions

