```
using Meditech.App.CommonService.Services;
using Microsoft.EntityFrameworkCore;
using Microsoft.OpenApi.Models;
using Meditech.Core.Auth.Authorization;
using Meditech.Core.Auth.DbModels;
using Meditech.Core.Auth.Helpers;
using Meditech.Core.Auth.Services;
using Meditech.Core.Infrastructure;
using Meditech.Core.Infrastructure.Cache;
using Meditech.Core.Infrastructure.Database;
using Meditech.Core.Infrastructure.InternalMiddleware;
using NLog;
using NLog.Web;
using Microsoft.AspNetCore.Localization;
using System.Globalization;
// using Microsoft.AspNetCore.Mvc.Versioning;
using Meditech.Lib.Share.CommonClass;
using Meditech.Core.Common.DBModels;
using Meditech.Core.Common.Repositories;
using Meditech.Lib.MediaStorage;
using Meditech.Core.Common.Services;
using DevExpress.AspNetCore;
using DevExpress.AspNetCore.Reporting;
using System.Runtime.InteropServices;
using Meditech.Core.Common.DbContexts.Ksk;

var logger = NLog.LogManager.Setup().LoadConfigurationFromAppSettings().GetCurrentClassLogger();
logger.Debug("Init Service Crm");

try
{
    var builder = WebApplication.CreateBuilder(args);

    // Add Dbcontext
    builder.Services.AddDbContext<CommonDatabaseContext>(opt =>
            opt.UseNpgsql(builder.Configuration.GetConnectionString("MasterDatabase")));
    builder.Services.AddScoped<DbContext, CommonDatabaseContext>();
    builder.Services.AddTransient<CommonUnitOfWork>();

    builder.Services.AddDbContext<KskDbContext>(opt => opt.UseNpgsql(builder.Configuration.GetConnectionString("KskDatabase")));
    builder.Services.AddScoped<DbContext, KskDbContext>();

    builder.Services.AddTransient(typeof(IEntityRepository<>), typeof(EntityRepository<>));

    // Repository
    builder.Services.AddTransient<IDocumentTypeRepository, DocumentTypeRepository>();
    builder.Services.AddTransient<ISignPositionRepository, SignPositionRepository>();
    builder.Services.AddTransient<ISignatureProcessRepository, SignatureProcessRepository>();
    builder.Services.AddTransient<IAttachmentRepository, AttachmentRepository>();
    builder.Services.AddTransient<ISignedDocumentRepository, SignedDocumentRepository>();
    builder.Services.AddTransient<IErpNotificationRepository, ErpNotificationRepository>();
    builder.Services.AddTransient<IErpNotificationReceiveRepository, ErpNotificationReceiveRepository>();
    builder.Services.AddTransient<IHistoryChangeRepository, HistoryChangeRepository>();
    builder.Services.AddTransient<IPrintRequestRepository, PrintRequestRepository>();
    builder.Services.AddTransient<IDonHangKhamSucKhoeRepository, DonHangKhamSucKhoeRepository>();
    builder.Services.AddTransient<IPhuongAnKinhDoanhKskRepository, PhuongAnKinhDoanhKskRepository>();
    builder.Services.AddTransient<IDonKSKData, DonKSKData>();
    builder.Services.AddTransient<IDoanKhamSucKhoeRepository, DoanKhamSucKhoeRepository>();
    builder.Services.AddTransient<IDanhSachCongTyRepository, DanhSachCongTyRepository>();
    builder.Services.AddTransient<IDonHangKhamSucKhoeKhacRepository, DonHangKhamSucKhoeKhacRepository>();

    // Service
    builder.Services.AddTransient<ISignatureProcessService, SignatureProcessService>();
    builder.Services.AddTransient<IAttachmentService, AttachmentService>();
    builder.Services.AddTransient<IMediaService, MediaService>();
    builder.Services.AddTransient<IErpNotificationService, ErpNotificationService>();
    builder.Services.AddTransient<IHistoryChangeService, HistoryChangeService>();
    builder.Services.AddTransient<ICommonService, CommonService>();
    builder.Services.AddTransient<IDBConnectServices, DBConnectServices>();
    builder.Services.AddTransient<IReportService, ReportService>();

    builder.Services.AddTransient<IDataInitalizeService, DataInitalizeService>();
    // Authentication
    builder.Services.Configure<JwtSettings>(builder.Configuration.GetSection("JwtSettings"));
    // configure DI for application services
    builder.Services.AddScoped<IJwtUtils, JwtUtils>();
    builder.Services.AddDbContext<UserAuthDbContext>(opt =>
            opt.UseNpgsql(builder.Configuration.GetConnectionString("AuthenDatabase")));
    builder.Services.AddTransient<Meditech.Core.Auth.Entities.AuthUnitOfWork>();
    builder.Services.AddTransient<Meditech.Core.Auth.Entities.IEmployeeRepository, Meditech.Core.Auth.Entities.EmployeeRepository>();
    builder.Services.AddTransient<Meditech.Core.Auth.Entities.IDepartmentCategoryRepository, Meditech.Core.Auth.Entities.DepartmentCategoryRepository>();
    builder.Services.AddTransient<Meditech.Core.Auth.Entities.IUserRepository, Meditech.Core.Auth.Entities.UserRepository>();
    builder.Services.AddTransient<Meditech.Core.Auth.Entities.IUserRoleRepository, Meditech.Core.Auth.Entities.UserRoleRepository>();
    builder.Services.AddTransient<Meditech.Core.Auth.Entities.IRolePermitRepository, Meditech.Core.Auth.Entities.RolePermitRepository>();
    builder.Services.AddTransient<Meditech.Core.Auth.Entities.IUserAuthRepository, Meditech.Core.Auth.Entities.UserAuthRepository>();
    builder.Services.AddTransient<Meditech.Core.Auth.Entities.IUserFirebaseTokenCrmRepository, Meditech.Core.Auth.Entities.UserFirebaseTokenCrmRepository>();
    builder.Services.AddTransient<Meditech.Core.Auth.Entities.IHisInfoRepository, Meditech.Core.Auth.Entities.HisInfoRepository>();
    builder.Services.AddScoped<IAuthenUserService, AuthenUserService>();

    var redisSection = builder.Configuration.GetSection("redis");
    string redisConnection = redisSection.GetValue<string>("connectionString");
    string instanceName = redisSection.GetValue<string>("instanceName");
    builder.Services.AddSingleton<ICacheService, CacheService>();
    builder.Services.AddStackExchangeRedisCache(option =>
    {
        option.Configuration = string.IsNullOrEmpty(redisConnection) ? "127.0.0.1:6379" : redisConnection;
        option.InstanceName = instanceName ?? "erp-";
    });

    // Bind host config
    var hostSetting = builder.Configuration.GetSection("HostConfig");
    builder.Services.Configure<HostConfig>(hostSetting);
```

```
var secureSetting = builder.Configuration.GetSection("InternalSecure");
builder.Services.Configure<InternalSecure>(secureSetting);
// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddSwaggerGen(options =>
{
    options.SwaggerDoc("v2", new OpenApiInfo { Title = "Common API", Version = "v2" });
    options.AddSecurityDefinition(name: "Bearer", securityScheme: new OpenApiSecurityScheme
    {
        Name = "Authorization",
        Description = "Enter the Bearer Authorization string as following: `Bearer Generated-JWT-Token`",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.ApiKey,
        Scheme = "Bearer"
    });
    options.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        {
            new OpenApiSecurityScheme
            {
                Name = "Bearer",
                In = ParameterLocation.Header,
                Reference = new OpenApiReference
                {
                    Id = "Bearer",
                    Type = ReferenceType.SecurityScheme
                }
            },
            new List<string>()
        }
    });
});

//builder.Services.AddApiVersioning(opt =>
//{
//    opt.DefaultApiVersion = new Microsoft.AspNetCore.Mvc.ApiVersion(1, 0);
//    opt.AssumeDefaultVersionWhenUnspecified = true;
//    opt.ReportApiVersions = true;
//    opt.ApiVersionReader = ApiVersionReader.Combine(new UrlSegmentApiVersionReader(),
//                                                    new HeaderApiVersionReader("x-api-version"),
//                                                    new MediaTypeApiVersionReader("x-api-version"));
//});
builder.Services.Configure<RequestLocalizationOptions>(
            options =>
            {
                var supportedCultures = new List<CultureInfo>
                {
                            new CultureInfo("en-US"),
                            new CultureInfo("vi-VN")
                };

                options.DefaultRequestCulture = new RequestCulture(culture: "vi-VN", uiCulture: "vi-VN");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
                options.RequestCultureProviders = new[] { new RouteDataRequestCultureProvider { IndexOfCulture = 1, IndexofUICulture = 1 } };
            });
builder.Services.Configure<RouteOptions>(options =>
{
    options.ConstraintMap.Add("culture", typeof(LanguageRouteConstraint));
});
builder.Host.UseNLog();
builder.WebHost.ConfigureKestrel(serverOptions =>
{
    serverOptions.Limits.MaxRequestBodySize = 200 * 1024 * 1024;
});

builder.Services
.AddControllersWithViews();
//.AddJsonOptions(options => options.JsonSerializerOptions.PropertyNamingPolicy = null);

builder.Services.AddDevExpressControls();
builder.Services.AddMvc();
builder.Services.ConfigureReportingServices(configurator => {
    configurator.ConfigureWebDocumentViewer(viewerConfigurator => {
        viewerConfigurator.UseCachedReportSourceBuilder();
    });
});

var app = builder.Build();
app.UseDevExpressControls();
// global cors policy
app.UseCors(x => x
    .AllowAnyOrigin()
    .AllowAnyMethod()
    .AllowAnyHeader());
// app.UseInternalSystemRequestValidate();

app.UseMiddleware<ErrorHandlerMiddleware>();

// custom jwt auth middleware
app.UseMiddleware<JwtMiddleware>();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
}
if (!RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
{
    DevExpress.Drawing.Internal.DXDrawingEngine.ForceSkia();
}
app.UseSwagger();

// Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),
// specifying the Swagger JSON endpoint.
```

```
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v2/swagger.json", "Common API");
    });
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    //app.MapControllerRoute(
    //    name: "default",
    //    pattern: "{controller=Home}/{action=Index}/{id?}");
    //app.MapControllers();

    app.MapControllerRoute(
       name: "default",
       pattern: "{controller=Home}/{action=PrintPreview}/{id?}");

    // Seed data
    using (var scope = app.Services.CreateScope())
    {
        try
        {
            var dataInitializeService = scope.ServiceProvider.GetRequiredService<IDataInitalizeService>();
            dataInitializeService.SystemInitialize();
        }
        catch (Exception ex)
        {
            logger.Error(ex, "seeding datav exception");
        }
    }

    app.MapControllerRoute(
        name: "design",
        pattern: "{controller=DesignReport}/{action=EditReport}/{id?}");
    app.Run();
}
catch (Exception exception)
{
    // NLog: catch setup errors
    logger.Error(exception, "Stopped program because of exception");
    throw;
}
finally
{
    // Ensure to flush and stop internal timers/threads before application-exit (Avoid segmentation fault on Linux)
    NLog.LogManager.Shutdown();
}
```