Identity是微软提供的认证包，其提供的功能有用户管理，用户登录。总结，安全又好用，对于中小项目，推荐直接使用。

这里演示前后端分离如何使用

**安装包**

我们安装如下nuget包

Microsoft.AspNetCore.ApiAuthorization.IdentityServer

**上下文修改**

我们的上下文需要继承IPersistedGrantDbContext接口，如下

[ConnectionStringName("Default")]

public class AuthorizationDbContext : AbpDbContext<AuthorizationDbContext>, IPersistedGrantDbContext

{

    /\* Add DbSet properties for your Aggregate Roots / Entities here. \*/

    #region Entities from the modules

    public DbSet<IceUser> IceUsers { get; set; }

    public DbSet<PersistedGrant> PersistedGrants { get; set; }

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

    public DbSet<Key> Keys { get; set; }

    #endregion

    protected override void OnModelCreating(ModelBuilder builder)

    {

        base.OnModelCreating(builder);

        // 配置 PersistedGrantDbContext 上下文的实体

        builder.ConfigurePersistedGrantContext(new OperationalStoreOptions() {

            DefaultSchema = AuthorizationDbProperties.DbSchema

        });

}

    public async Task<int> SaveChangesAsync()

    {

        return base.SaveChanges();

    }

}

**添加依赖服务**

接着我们需要再DI中注册如下服务

// 设置 Identity 使用的上下文

context.Services.AddDefaultIdentity<IceUser>(options => {

}).AddEntityFrameworkStores<AuthorizationDbContext>();

// IdentityService 的配置，不了解需要去查看其文档

context.Services.AddIdentityServer()

    .AddSigningCredentials()

    // 添加身份资源

    .AddInMemoryIdentityResources(Config.GetIdentityResources())

    // 添加Api资源

    .AddInMemoryApiResources(Config.GetApis())

    // 添加客户端

    .AddInMemoryClients(Config.GetClients())

    // 资源密码验证器

.AddResourceOwnerValidator<IceResourceOwnerPasswordValidator>();

// 添加 IdentityService Token 的认证

context.Services.AddAuthentication()

.AddIdentityServerJwt();

// 自定义 JWT 的认证信息

context.Services.Configure<JwtBearerOptions>(

    IdentityServerJwtConstants.IdentityServerJwtBearerScheme,

    options =>

    {

        // options.Authority = "http://IceAuthorization:11080";

        options.Audience = "Authorization";

        options.RequireHttpsMetadata = false;

});

// 定义 Identity 的规则设置

context.Services.Configure<IdentityOptions>(options =>

{

    // 密码规则

    options.Password.RequireDigit = true;

    options.Password.RequireLowercase = true;

    options.Password.RequireNonAlphanumeric = true;

    options.Password.RequireUppercase = true;

    options.Password.RequiredLength = 6;

    options.Password.RequiredUniqueChars = 1;

    // 锁定设置，多次登录失败后用户会被锁定

    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);

    options.Lockout.MaxFailedAccessAttempts = 5;

    options.Lockout.AllowedForNewUsers = true;

    // 用户设置

    options.User.AllowedUserNameCharacters =

    "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-.\_@+";

    options.User.RequireUniqueEmail = false;

});

// 添加权限

context.Services.AddAuthentication();

**注册中间件**

然后我们需要注册如下中间件

app.UseAuthentication();

app.UseIdentityServer();

app.UseAuthorization();

**用户管理**

UserManager提供了一套用户管理的方法，你可以通过它来进行用户管理

[IceRoleAuthorize(IceRoleAuthorizeType.Host)]

public class IceUserAppService : AuthorizationAppService

{

    protected UserManager<IceUser> UserManager { get; }

    public async Task UpdateAsync(Guid id, UpdateInput input) {

        ...

        // 更新密码

        if (!string.IsNullOrWhiteSpace(input.Password)) {

            await UserManager.RemovePasswordAsync(user);

            await UserManager.AddPasswordAsync(user, input.Password);

        }

    }

    public async Task CreateAsync(CreateInput input) {

        ...

        // 创建用户

        var result = await UserManager.CreateAsync(user, input.Password);

    }

    public virtual async Task DeleteAsync(Guid id)

    {

        ...

    }

}

**登录管理**

public class IceResourceOwnerPasswordValidator : IResourceOwnerPasswordValidator

{

    protected SignInManager<IceUser> SignInManager { get; }

    public async Task ValidateAsync(ResourceOwnerPasswordValidationContext context)

    {

        IceUser user = xxxx.FirstOrDefault(e => e.UserName == context.UserName && e.TenantId == tenantId);

        // 验证密码是否正确

        var result = await SignInManager.CheckPasswordSignInAsync(user, context.Password, true);

        if (result.Succeeded)

        {

            // 密码正确，登录成功

        }

        else

        {

            if (result.IsLockedOut)

            {

                errorDescription = "由于无效的登录尝试，用户帐户已被锁定。请稍等片刻后重试";

            }

            else if (result.IsNotAllowed)

            {

                errorDescription = "你已被禁止登录系统，请联系管理员";

            }

            else

            {

                errorDescription = "用户名或密码错误";

            }

        }

    }

}