Module Zero安装

**从模板创建（自动方式）**

**手动安装**

核心（域）层

域类（实体）

经理（域名服务）

权限检查

**基础设施层**

EntityFramework

DbContext

数据库迁移

初始化数据

**表示层**

NuGet包

Owin启动类

账户控制器

登录查看

保护应用程序

从模板创建（自动方式）

使用ABP和module-zero开始一个新项目的最简单方法是在templates page创建模板。在这个页面，进入“ABP + module zero”选项卡，并创建项目。目前，该模板只支持EntityFramework 作为ORM（还不支持NHibernate）。

创建和下载项目后，

* 打开你的Visual Studio解决方案。
* 选择“Web”项目作为启动项目。
* 打开包管理器控制台，选择“Entity Framework”项目为默认项目并运行的Entity Framework的“Update-Database”命令。这将创建数据库。
* 运行该应用程序。用户名是“管理员”，密码默认值为“123qwe”。

在此模板，多租户默认情况下启用。如果你不需要，您可以在Core工程的模块文件中禁用它。

手动安装

如果你有一个预先创建的应用程序，稍后再安装module-zero，您可以按照本部分的说明。

在本文中，我将假定您的解决方案具有以下项目：

AbpZeroSample.Core

AbpZeroSample.Application

AbpZeroSample.EntityFramework

AbpZeroSample.Web

AbpZeroSample.WebApi

核心（域）层

安装Abp.Zero的NuGet包到时Core工程。然后进入到核心模块类（在此示例中为AbpZeroSampleCoreModule类），并添加DependsOn属性AbpZeroCoreModule如下图所示：

**[DependsOn(typeof(AbpZeroCoreModule))]**

public class AbpZeroSampleCoreModule : AbpModule

{

public override void Initialize()

{

IocManager.RegisterAssemblyByConvention(Assembly.GetExecutingAssembly());

}

}

领域类（实体）

Module-zero提供了用户，角色和租客类抽象。所以，我们应该实现它们，如下所示：

public class User : AbpUser<Tenant, User>

{

}

public class Role : AbpRole<Tenant, User>

{

}

public class Tenant : AbpTenant<Tenant, User>

{

}

你可以在这里添加自定义属性。通过这种方式，我们可以根据我们的需要扩展基用户，角色和租户类。

管理器（领域服务）

我们应该实现基用户，角色和租户管理器和存储类，因为他们也是抽象的。

从启动用户存储和用户管理器：

public class UserStore : AbpUserStore<Tenant, Role, User>

{

public UserStore(

IRepository<User, long> userRepository,

IRepository<UserLogin, long> userLoginRepository,

IRepository<UserRole, long> userRoleRepository,

IRepository<Role> roleRepository,

IRepository<UserPermissionSetting, long> userPermissionSettingRepository,

IAbpSession session,

IUnitOfWorkManager unitOfWorkManager)

: base(

userRepository,

userLoginRepository,

userRoleRepository,

roleRepository,

userPermissionSettingRepository,

session,

unitOfWorkManager)

{

}

}

public class UserManager : AbpUserManager<Tenant, Role, User>

{

public UserManager(

UserStore store,

RoleManager roleManager,

IRepository<Tenant> tenantRepository,

IMultiTenancyConfig multiTenancyConfig,

IPermissionManager permissionManager,

IUnitOfWorkManager unitOfWorkManager,

ISettingManager settingManager)

: base(

store,

roleManager,

tenantRepository,

multiTenancyConfig,

permissionManager,

unitOfWorkManager,

settingManager)

{

}

}

别担心依赖列表。他们可能会在下一个版本改变。如果需要的话只是准备构造函数。角色存储和角色管理器也类似：

public class RoleStore : AbpRoleStore<Tenant, Role, User>

{

public RoleStore(

IRepository<Role> roleRepository,

IRepository<UserRole, long> userRoleRepository,

IRepository<RolePermissionSetting, long> rolePermissionSettingRepository)

: base(

roleRepository,

userRoleRepository,

rolePermissionSettingRepository)

{

}

}

public class RoleManager : AbpRoleManager<Tenant, Role, User>

{

public RoleManager(

RoleStore store,

IPermissionManager permissionManager,

IRoleManagementConfig roleManagementConfig,

IUnitOfWorkManager unitOfWorkManager)

: base(

store,

permissionManager,

roleManagementConfig,

unitOfWorkManager)

{

}

}

最后，我们要创建一个类租户管理器（这里没有租户存储）：

public class TenantManager : AbpTenantManager<Tenant, Role, User>

{

public TenantManager(IRepository<Tenant> tenantRepository)

: base(tenantRepository)

{

}

}

权限检查器

为了使授权系统工作，我们应该实现权限检查器：

public class PermissionChecker : PermissionChecker<Tenant, Role, User>

{

public PermissionChecker(UserManager userManager)

: base(userManager)

{

}

}

基础设施层

Entity Framework

如果您选择Entity Framework，我们应该配置它以便使用module-zero。安装Abp.Zero.EntityFramework的NuGet包到EntityFramework项目。然后进入到模块文件（此示例中为AbpZeroSampleDataModule）和改变AbpEntityFrameworkModule依赖到AbpZeroEntityFrameworkModule如下图所示：

[**DependsOn(typeof(AbpZeroEntityFrameworkModule)**, typeof(AbpZeroSampleCoreModule))]

public class AbpZeroSampleDataModule : AbpModule

{

//...

}

DbContext

转到您的DbContext类，并改变基类AbpDbContext为AbpZeroDbContext如下图所示：

public class AbpZeroSampleDbContext : **AbpZeroDbContext<Tenant, Role, User>**

{

//...

}

从而，module-zero里的基础实体被添加到你的数据库环境中。

数据库迁移

现在，我们应该创建数据库迁移，因为我们的数据库上下文被更改了。打开包管理器控制台，然后键入以下命令：

Add-Migration "AbpZero\_Installed"

当然，你可以选择不同的迁移名称。不要忘了在包管理器控制台中选择默认工程为AbpZeroSample.EntityFramework（对于你的例子AbpZeroSample将是不同的）。执行这个命令之后，一个新的迁移文件被添加到工程中。检查它，如果你需要可以改变它。然后键入以下命令来更新数据库模式：

Update-Database

您可以检查EntityFramework的代码优先迁移文档以获取更多信息。

初始化数据

如果你检查你的数据库，你会看到表已经被创建，但它们是空的。您可以使用EntityFramework的播种以填补初始数据。您可以使用这样的类作为初始数据生成器：

public class DefaultTenantRoleAndUserBuilder

{

private readonly AbpZeroSampleDbContext \_context;

public DefaultTenantRoleAndUserBuilder(AbpZeroSampleDbContext context)

{

\_context = context;

}

public void Build()

{

CreateUserAndRoles();

}

private void CreateUserAndRoles()

{

//Admin role for tenancy owner

var adminRoleForTenancyOwner = \_context.Roles.FirstOrDefault(r => r.TenantId == null && r.Name == "Admin");

if (adminRoleForTenancyOwner == null)

{

adminRoleForTenancyOwner = \_context.Roles.Add(new Role {Name = "Admin", DisplayName = "Admin"});

\_context.SaveChanges();

}

//Admin user for tenancy owner

var adminUserForTenancyOwner = \_context.Users.FirstOrDefault(u => u.TenantId == null && u.UserName == "admin");

if (adminUserForTenancyOwner == null)

{

adminUserForTenancyOwner = \_context.Users.Add(

new User

{

TenantId = null,

UserName = "admin",

Name = "System",

Surname = "Administrator",

EmailAddress = "admin@aspnetboilerplate.com",

IsEmailConfirmed = true,

Password = "AM4OLBpptxBYmM79lGOX9egzZk3vIQU3d/gFCJzaBjAPXzYIK3tQ2N7X4fcrHtElTw==" //123qwe

});

\_context.SaveChanges();

\_context.UserRoles.Add(new UserRole(adminUserForTenancyOwner.Id, adminRoleForTenancyOwner.Id));

\_context.SaveChanges();

}

//Default tenant

var defaultTenant = \_context.Tenants.FirstOrDefault(t => t.TenancyName == "Default");

if (defaultTenant == null)

{

defaultTenant = \_context.Tenants.Add(new Tenant {TenancyName = "Default", Name = "Default"});

\_context.SaveChanges();

}

//Admin role for 'Default' tenant

var adminRoleForDefaultTenant = \_context.Roles.FirstOrDefault(r => r.TenantId == defaultTenant.Id && r.Name == "Admin");

if (adminRoleForDefaultTenant == null)

{

adminRoleForDefaultTenant = \_context.Roles.Add(new Role { TenantId = defaultTenant.Id, Name = "Admin", DisplayName = "Admin" });

\_context.SaveChanges();

}

//Admin for 'Default' tenant

var adminUserForDefaultTenant = \_context.Users.FirstOrDefault(u => u.TenantId == defaultTenant.Id && u.UserName == "admin");

if (adminUserForDefaultTenant == null)

{

adminUserForDefaultTenant = \_context.Users.Add(

new User

{

TenantId = defaultTenant.Id,

UserName = "admin",

Name = "System",

Surname = "Administrator",

EmailAddress = "admin@aspnetboilerplate.com",

IsEmailConfirmed = true,

Password = "AM4OLBpptxBYmM79lGOX9egzZk3vIQU3d/gFCJzaBjAPXzYIK3tQ2N7X4fcrHtElTw==" //123qwe

});

\_context.SaveChanges();

\_context.UserRoles.Add(new UserRole(adminUserForDefaultTenant.Id, adminRoleForDefaultTenant.Id));

\_context.SaveChanges();

}

}

}

该类创建默认租户，角色和用户。我们可以用它在EF的配置类播种我们的数据库：

internal sealed class Configuration : DbMigrationsConfiguration<AbpZeroSample.EntityFramework.AbpZeroSampleDbContext>

{

public Configuration()

{

AutomaticMigrationsEnabled = false;

ContextKey = "AbpZeroSample";

}

**protected override void Seed(AbpZeroSample.EntityFramework.AbpZeroSampleDbContext context)**

**{**

**context.DisableAllFilters();**

**new DefaultTenantRoleAndUserBuilder(context).Build();**

**}**

}

在这里，我们禁用数据过滤器（所以我们可以自由地操纵数据库），并使用的初始数据生成器类。

表示层

NuGet包

添加以下的NuGet包到.Web工程：

* Abp.Zero.EntityFramework（这也将增加Abp.Zero和所有的依赖）
* Microsoft.AspNet.Identity.Owin
* Microsoft.Owin.Host.SystemWeb

Owin启动类

加入这样一个Owin启动类：

using AbpZeroSample.Web;

using Microsoft.AspNet.Identity;

using Microsoft.Owin;

using Microsoft.Owin.Security.Cookies;

using Owin;

**[assembly: OwinStartup(typeof(Startup))]**

namespace AbpZeroSample.Web

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

// Enable the application to use a cookie to store information for the signed in user

app.UseCookieAuthentication(new CookieAuthenticationOptions

{

AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,

LoginPath = new PathString("/Account/Login")

});

// Use a cookie to temporarily store information about a user logging in with a third party login provider

app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

}

}

}

账户控制器

我们可以创建一个控制器用于登录/注销，如下图所示：

public class AccountController : AbpZeroSampleControllerBase

{

private readonly UserManager \_userManager;

private IAuthenticationManager AuthenticationManager

{

get

{

return HttpContext.GetOwinContext().Authentication;

}

}

public AccountController(**UserManager userManager**)

{

\_userManager = userManager;

}

public ActionResult Login(string returnUrl = "")

{

if (string.IsNullOrWhiteSpace(returnUrl))

{

returnUrl = Request.ApplicationPath;

}

ViewBag.ReturnUrl = returnUrl;

return View();

}

[HttpPost]

public async Task<JsonResult> Login(LoginViewModel loginModel, string returnUrl = "")

{

if (!ModelState.IsValid)

{

throw new UserFriendlyException("Your form is invalid!");

}

var loginResult = await \_userManager.LoginAsync(

loginModel.UsernameOrEmailAddress,

loginModel.Password,

loginModel.TenancyName

);

switch (loginResult.Result)

{

case AbpLoginResultType.Success:

break;

case AbpLoginResultType.InvalidUserNameOrEmailAddress:

case AbpLoginResultType.InvalidPassword:

throw new UserFriendlyException("Invalid user name or password!");

case AbpLoginResultType.InvalidTenancyName:

throw new UserFriendlyException("No tenant with name: " + loginModel.TenancyName);

case AbpLoginResultType.TenantIsNotActive:

throw new UserFriendlyException("Tenant is not active: " + loginModel.TenancyName);

case AbpLoginResultType.UserIsNotActive:

throw new UserFriendlyException("User is not active: " + loginModel.UsernameOrEmailAddress);

case AbpLoginResultType.UserEmailIsNotConfirmed:

throw new UserFriendlyException("Your email address is not confirmed!");

default: //Can not fall to default for now. But other result types can be added in the future and we may forget to handle it

throw new UserFriendlyException("Unknown problem with login: " + loginResult.Result);

}

AuthenticationManager.SignOut(DefaultAuthenticationTypes.ExternalCookie);

AuthenticationManager.SignIn(new AuthenticationProperties { IsPersistent = loginModel.RememberMe }, loginResult.Identity);

if (string.IsNullOrWhiteSpace(returnUrl))

{

returnUrl = Request.ApplicationPath;

}

return Json(new MvcAjaxResponse { TargetUrl = returnUrl });

}

public ActionResult Logout()

{

AuthenticationManager.SignOut();

return RedirectToAction("Login");

}

}

附带一个简单的LoginViewModel：

public class LoginViewModel

{

public string TenancyName { get; set; }

[Required]

public string UsernameOrEmailAddress { get; set; }

[Required]

public string Password { get; set; }

public bool RememberMe { get; set; }

}

登录视图

为了能够使用的AccountController，我们应该创建一个登录页面。这由你决定创建一个登录表单。只需通过AJAX使用适当的参数调用AccountController.Login。

保护应用程序

现在，我们可以添加一个AbpAuthorize特性到HomeController，以确保只有合法的用户才能进入到页面：

**[AbpMvcAuthorize]**

public class HomeController : AbpZeroSampleControllerBase

{

public ActionResult Index()

{

return View("~/App/Main/views/layout/layout.cshtml"); //Layout of the angular application.

}

}