Kentico provides an integration package that allows you to work with Kentico membership data on websites presented by a separate MVC application.

You can set up the following scenarios:

- Allow visitors to sign in with Kentico user accounts
- Allow users to register new accounts from the MVC site (the user data is stored in the shared Kentico database)
- Allow users to reset their passwords
- Authorize actions based on Kentico roles
- Use external services for authentication

The membership integration is based on ASP.NET Identity and the OWIN standard. As a result, you can work with user data through the standard approaches that you would use in any ASP.NET MVC application.

Integrating Kentico membership into your project

Before you can start working with Kentico membership data in your MVC application, you need to integrate the required API:

1. Open your MVC project in Visual Studio.
2. Install the **Kentico.Membership** NuGet integration package.
3. Install the **Microsoft.Owin.Host.SystemWeb** NuGet package.
4. Add a **Startup.Auth** class to your project's **App_Start** folder (or modify your existing authentication startup file):

```
 »using System;
using System.Web;
using System.Web.Mvc;

using Microsoft.Owin;
using Microsoft.Owin.Security.Cookies;
using Microsoft.AspNet.Identity;
using Owin;

using CMS.Helpers;
using CMS.SiteProvider;

using Kentico.Membership;

// Assembly attribute that sets the OWIN startup class
// This example sets the Startup class from the 'LearningKit.App_Start'
namespace, not 'LearningKit.App_Start.Basic' used below
// The active Startup class is defined in Startup.Auth.cs and additionally
demonstrates registration of external authentication services
[assembly: OwinStartup(typeof(LearningKit.App_Start.Startup))]

namespace LearningKit.App_Start.Basic
{
    public partial class Startup
    {
        // Cookie name prefix used by OWIN when creating authentication cookies
        private const string OWIN_COOKIE_PREFIX = ".AspNet.";


        public void Configuration(IAppBuilder app)
        {
            // Registers the Kentico.Membership identity implementation
            app.CreatePerOwinContext(() => UserManager.Initialize(app, new
UserManager(new UserStore(SiteContext.CurrentSiteName))));
            app.CreatePerOwinContext<SignInManager>(SignInManager.Create);

            // Configures the authentication cookie
```

```
            UrlHelper urlHelper = new UrlHelper(HttpContext.Current.Request.
RequestContext);
            app.UseCookieAuthentication(new CookieAuthenticationOptions
            {
                AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
                // Fill in the name of your sign-in action and controller
                LoginPath = new PathString(urlHelper.Action("SignIn", "Account")),
                Provider = new CookieAuthenticationProvider
                {
                    // Sets the return URL for the sign-in page redirect (fill in
the name of your sign-in action and controller)
                    OnApplyRedirect = context => context.Response.Redirect
(urlHelper.Action("SignIn", "Account")
                                                    + new Uri(context.RedirectUri).
Query)
                }
            });

            // Registers the authentication cookie with the 'Essential' cookie
level
            // Ensures that the cookie is preserved when changing a visitor's
allowed cookie level below 'Visitor'
            CookieHelper.RegisterCookie(OWIN_COOKIE_PREFIX +
DefaultAuthenticationTypes.ApplicationCookie, CookieLevel.Essential);
        }
    }
}
```

Registering authentication cookies

We strongly recommend registering all authentication cookies used on your website with an appropriate cookie level (typically **Essential** when working with the default cookie level values).

Otherwise you may encounter problems with the cookies being cleared after adjusting the allowed cookie level for visitors (for example when managing tracking consent). Changes of the allowed cookie level automatically remove all cookies above the given level. Any unregistered cookies are processed with the *Visitor* level, which is usually too high for basic authentication cookies.

To register a cookie, call the **CookieHelper.RegisterCookie** method (available in the **CMS.Helpers** namespace of the Kentico API) in your application's startup code. You can access the default level values in the **CookieLevel** enumeration.

The Kentico identity implementation is now registered and you can access the *Kentico.Membership* API in your application's code. Continue by setting up user authentication. You can also implement the following features:

- User registration
- Password reset functionality
- Role-based authorization for your controller actions
- Integration of external authentication services

Setting up authentication

Once you integrate Kentico membership into the project, you can implement actions that allow visitors to sign in and out of your MVC website with Kentico user accounts.

Kentico user-related features

The following settings in Kentico affect authentication on MVC sites:

- Only user accounts that are **Enabled** in Kentico can be used for authentication. Accounts that have the enabled flag disabled (either manually or due to an account lock) cannot sign in. You can manage the enabled status of users in the *Users* application within the Kentico administration interface.
- **Settings -> Security & Membership**:
  - **Share user accounts on all sites** - if enabled, users from any site in the system can be used for authentication on MVC sites. Otherwise users must be assigned to the corresponding MVC site (in the *Users* application).
  - **Use site prefixes for user names** - must be disabled. Site prefixes for user names are not compatible with authentication on MVC sites.

All other user-related settings and features are NOT supported for authentication on MVC sites. For example, successful authentication does not update the *Last sign-in* date in Kentico and failed authentication attempts are not tracked (e.g. for the purposes of account locking).

Use the following approach to develop sign-in actions:

Tip: To view the full code of a functional example directly in Visual Studio, download the Kentico MVC solution from GitHub and inspect the **LearningKit** project. You can also run the Learning Kit website after connecting the project to a Kentico database.

1. Create a new controller class in your MVC project or edit an existing one.
2. Prepare a property that gets an instance of the **Kentico.Membership.SignInManager** class for the current request – call *HttpContext.GetOwinContext().Get<SignInManager>()*.
3. Implement two sign-in actions – one basic GET action to display the sign-in form and a second POST action to handle the authentication when the form is submitted.
4. Call the **PasswordSignInAsync** method of the *SignInManager* instance to authenticate users against the Kentico database (within the code of the POST action).

```
using System;
using System.Web;
using System.Web.Mvc;
using System.Threading.Tasks;

using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;

using Kentico.Membership;

using CMS.EventLog;
using CMS.SiteProvider;
```

Sign-in actions example

```
        /// <summary>
        /// Provides access to the Kentico.Membership.SignInManager instance.
        /// </summary>
        public SignInManager SignInManager
        {
            get
            {
```

```
                return HttpContext.GetOwinContext().Get<SignInManager>();
            }
        }

        /// <summary>
        /// Basic action that displays the sign-in form.
        /// </summary>
        public ActionResult SignIn()
        {
            return View();
        }

        /// <summary>
        /// Handles authentication when the sign-in form is submitted. Accepts
parameters posted from the sign-in form via the SignInViewModel.
        /// </summary>
        [HttpPost]
        [ValidateAntiForgeryToken]
        [ValidateInput(false)]
        public async Task<ActionResult> SignIn(SignInViewModel model, string
returnUrl)
        {
            // Validates the received user credentials based on the view model
            if (!ModelState.IsValid)
            {
                // Displays the sign-in form if the user credentials are invalid
                return View();
            }

            // Attempts to authenticate the user against the Kentico database
            SignInStatus signInResult = SignInStatus.Failure;
            try
            {
                signInResult = await SignInManager.PasswordSignInAsync(model.
UserName, model.Password, model.SignInIsPersistent, false);
            }
            catch (Exception ex)
            {
                // Logs an error into the Kentico event log if the authentication
fails
                EventLogProvider.LogException("MvcApplication", "SignIn", ex);
            }

            // If the authentication was not successful, displays the sign-in
form with an "Authentication failed" message
            if (signInResult != SignInStatus.Success)
            {
                ModelState.AddModelError(String.Empty, "Authentication failed");
                return View();
            }

            // If the authentication was successful, redirects to the return URL
when possible or to a different default action
            string decodedReturnUrl = Server.UrlDecode(returnUrl);
            if (!string.IsNullOrEmpty(decodedReturnUrl) && Url.IsLocalUrl
(decodedReturnUrl))
            {
                return Redirect(decodedReturnUrl);
            }
```

```
            return RedirectToAction("Index", "Home");
        }
```

5. We recommend creating a view model for your sign-in action *(SignInViewModel* in the example above). The view model allows you to:
    - Pass parameters from the sign-in form (username, password and sign-in persistence status).
    - Use data annotations to define validation and formatting rules for the sign-in parameters. See System.ComponentModel.DataAnnotations on MSDN for more information about the available data annotation attributes.

To allow users to sign out on your website, extend your sign-in controller class:

1. Prepare a property that provides access to the authentication middleware functionality (*Microsoft.Owin.Security.IAuthenticationManager* instance) – use the *HttpContext.GetOwinContext().Authentication* property.
2. Add another action for handling of sign-out requests.
3. Call the **SignOut(DefaultAuthenticationTypes.ApplicationCookie)** method of the authentication manager to sign out the current user.

Sign-out action example

```
        /// <summary>
        /// Provides access to the Microsoft.Owin.Security.IAuthenticationManager
instance.
        /// </summary>
        public IAuthenticationManager AuthenticationManager
        {
            get
            {
                return HttpContext.GetOwinContext().Authentication;
            }
        }

        /// <summary>
        /// Action for signing out users. The Authorize attribute allows the
action only for users who are already signed in.
        /// </summary>
        [Authorize]
        public ActionResult SignOut()
        {
            // Signs out the current user
            AuthenticationManager.SignOut(DefaultAuthenticationTypes.
ApplicationCookie);

            // Redirects to a different action after the sign-out
            return RedirectToAction("Index", "Home");
        }
```

Finally, you need to design a user interface for the authentication logic:

- Create a view for the *SignIn* action and display an appropriate sign-in form for your website. We recommend using a strongly typed view based on your sign-in view model.
- Add a sign in button or link that targets the *SignIn* action (for example within your site's main layout page).
- Add a sign out button or link that targets the *SignOut* action.

Visitors can now sign in to your site with Kentico user accounts from the connected database. If you wish to allow users to register new accounts, see User registration on MVC sites.

Tip: When writing additional code or views for your website, you can access information about the currently authenticated user via the standard **User.Identity** object. For example, *User.Identity.Name* returns the username of the currently signed in user.

Ensuring the correct password format

If your Kentico application uses custom salt values when generating password hashes, you also need to set the same values for the MVC application. Authentication will always fail if the password hashes are not identical for both applications.

Check the *appSettings* section of your Kentico application's web.config for the following keys:

- **CMSPasswordSalt**
- **CMSUserSaltColumn** (obsolete key used only for backward compatibility)

If either of the keys are present, copy them to the web.config of your MVC project.

See also: Setting the user password format

Authorizing actions or controllers based on roles

After you integrate Kentico membership into the project and set up authentication, you can use Kentico roles to restrict access to your MVC site's functionality or content.

Add the standard **Authorize** attribute from the *System.Web.Mvc* API to your controller classes or action methods. Set the attribute's **Roles** property and identify the required Kentico roles using their ***Role name*** (not the display name).

Example

```
        // Allows the "RestrictedPage" action only for signed in users who belong to
the "KenticoRole" role
        [Authorize(Roles = "KenticoRole")]
        public ActionResult RestrictedPage()
        {
            return View();
        }
```

The standard MVC framework behavior applies if an unauthorized user tries to access an action or controller – the application returns a 401 HTTP status code. The 401 status code causes a redirect to your site's sign-in page if one is configured.

When determining whether a user is a member of a Kentico role on an MVC site, the following conditions apply:

- The role must be assigned to the user for the given MVC site or as a global role. Roles assigned for other sites in the Kentico system are not recognized.

  Kentico matches sites to MVC applications based on the **Presentation URL** or **Domain name** set for sites in the **Sites** application.

- Roles limited by the **Valid to** setting are not recognized by MVC applications after their expiration date.
- Roles assigned indirectly through memberships are also valid and recognized by MVC applications.

Notes:

- When a user's roles are modified in Kentico, the changes apply only after the user signs out and in again on the MVC site.
- The **Privilege level** set for users in Kentico does NOT affect authorization on MVC sites. This means administrators cannot bypass role requirements like they would on standard Kentico sites.
- Kentico permission settings for roles do NOT have any effect on MVC sites (for example page-level permissions). The only relevant factor is whether a user belongs to the roles specified by the **Authorize** attributes in the code of your controllers.

Managing user roles from the MVC application

You can use the ASP.NET Identity API to add or remove Kentico roles for users. For example, this allows you to assign roles to new users immediately after registration.

To use the available role management methods:

- Prepare a property that gets an instance of the **Kentico.Membership.UserManager** class for the current request – call *HttpContext.GetOwinContext().Get<UserManager>()*.
- Get the ID of the user whose roles you wish to manage (the methods require the ID as a parameter).

```
using System.Web;

using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;

using Kentico.Membership;
```

```
        /// <summary>
        /// Provides access to the Kentico.Membership.UserManager instance.
        /// </summary>
        public UserManager UserManager
        {
            get
            {
                return HttpContext.GetOwinContext().Get<UserManager>();
            }
        }

        /// <summary>
        /// Gets the Kentico.Membership.User representation of the currently signed in
user.
        /// You can use the object to access the user's ID, which is required by the
role management methods.
        /// </summary>
        public User CurrentUser
        {
            get
            {
                return UserManager.FindByName(User.Identity.Name);
            }
        }
```

To add roles, call the **AddToRolesAsync** method of the *UserManager* instance. You can add one or more roles, each specified by a separate string parameter (equal to the corresponding role name).

```
                        // Attempts to assign the current user to the "KenticoRole"
and "CMSBasicUsers" roles
                        IdentityResult addResult = await UserManager.AddToRolesAsync
(CurrentUser.Id, "KenticoRole", "CMSBasicUsers");
```

To remove roles, call the **RemovesFromRolesAsync** method of the *UserManager* instance. You can remove one or more roles, each specified by a separate string parameter.

```
                    // Attempts to remove the "KenticoRole" and "CMSBasicUsers" roles
from the current user
                    IdentityResult removeResult = await UserManager.
RemoveFromRolesAsync(CurrentUser.Id, "KenticoRole", "CMSBasicUsers");
```

Note: You cannot use the ASP.NET Identity API to remove roles assigned to users indirectly through Kentico memberships.

Managing membership data through the Kentico API

In addition to the ASP.NET Identity API, you can alternatively work with Kentico membership data using the *CMS.Membership* API (provided as part of the Kentico.Libraries integration package).