External authentication services (identity providers) allow website visitors to sign in with existing user accounts from other sites or services. When developing MVC websites, you can set up external authentication using the **Kentico.Membership** integration package, in combination with the standard ASP.NET Identity API.

When a user successfully authenticates through an external service, you can use the provided API to create a corresponding user in the shared Kentico database.

The following third-party authentication services are currently supported:

- **WS-Federation** services (for example Active Directory Federation Services or Azure Active Directory)
- OpenID Connect services (for example Azure Active Directory)
- Google authentication (OAuth 2.0)
- Facebook authentication (OAuth 2.0)

You may be able to set up authentication via other types of services, but only the ones listed above are guaranteed and tested with the *Kentico.Membership* implementation.

Registering external authentication services

Use the following steps to install and register external authentication services into your MVC project:

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. Install NuGet packages containing the authentication middleware components required for the services you wish to use:
   - **Microsoft.Owin.Security.WsFederation**
   - **Microsoft.Owin.Security.OpenIdConnect**
   - **Microsoft.Owin.Security.Facebook**
   - **Microsoft.Owin.Security.Google**
5. Add a **Startup.Auth** class to your project's **App_Start** folder (or modify your existing authentication startup file).
6. Set up cookie authentication to allow the application to store information about signed in users. See Working with users on MVC sites for details.

   > **Note**: The code registering cookie authentication must be placed ***before*** you add any external authentication services.

7. Register any required authentication services within the *Startup* class (using the OWIN API provided by the installed packages):

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

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

using CMS.Helpers;
using CMS.SiteProvider;

using Kentico.Membership;

using Microsoft.Owin.Security.WsFederation;
using Microsoft.Owin.Security.OpenIdConnect;
```

```
using Microsoft.Owin.Security.Facebook;
using Microsoft.Owin.Security.Google;

// Assembly attribute that sets the OWIN startup class
[assembly: OwinStartup(typeof(LearningKit.App_Start.Startup))]

namespace LearningKit.App_Start
{
    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);

            // Uses a cookie to temporarily store information about users signing
in via external authentication services
            app.UseExternalSignInCookie(DefaultAuthenticationTypes.
ExternalCookie);

            // Registers a WS-Federation authentication service
            app.UseWsFederationAuthentication(
                    new WsFederationAuthenticationOptions
                    {
                    // Set any properties required by your authentication service
                    MetadataAddress = "placeholder", // Fill in the address of
your service's WS-Federation metadata
                    Wtrealm = "",
```

```
                    // When using external services, Passive authentication mode
may help avoid redirect loops for 401 responses
                    AuthenticationMode = AuthenticationMode.Passive
                    });

            // Registers an OpenID Connect authentication service
            app.UseOpenIdConnectAuthentication(
                new OpenIdConnectAuthenticationOptions
                {
                    // Set any properties required by your authentication service
                    ClientId = "placeholder",
                    ClientSecret = "placeholder",
                    Authority = "placeholder",
                    AuthenticationMode = AuthenticationMode.Passive
                });

            // Registers the Facebook authentication service
            app.UseFacebookAuthentication(
                new FacebookAuthenticationOptions
                {
                    // Fill in the application ID and secret of your Facebook
authentication application
                    AppId = "placeholder",
                    AppSecret = "placeholder"
                });

            // Registers the Google authentication service
            app.UseGoogleAuthentication(
                new GoogleOAuth2AuthenticationOptions
                {
                    // Fill in the client ID and secret of your Google
authentication application
                    ClientId = "placeholder",
                    ClientSecret = "placeholder"
                });
        }
    }
}
```

The Kentico identity implementation and the specified external authentication services are now registered for your application. Continue by setting up the components required for using external authentication on your website (controllers, views, etc.).

Setting up external authentication

After you register the required authentication services, you need to implement actions that redirect visitors to the external authentication service, process the response, and handle the corresponding user accounts in Kentico.

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.

The following is a basic example of external authentication handling:

1. Create a new controller class in your MVC project or edit an existing one.
2. Prepare the following properties:

- **SignInManager** - gets an instance of the *Kentico.Membership.SignInManager* class for the current request – call *HttpContext.GetOwinContext().Get<SignInManager>()*.
- **UserManager** - gets an instance of the *Kentico.Membership.UserManager* class for the current request – call *HttpContext.GetOwinContext().Get<UserManager>()*.
- **AuthenticationManager** - provides access to the authentication middleware functionality (*Microsoft.Owin.Security.IAuthenticationManager* instance) – use the *HttpContext.GetOwinContext().Authentication* property.

```
using System;
using System.Web;
using System.Web.Mvc;
using System.Threading.Tasks;
using CMS.Helpers;
using CMS.SiteProvider;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;

using Kentico.Membership;
```

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

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

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

3. Add a POST action that handles redirection to external authentication services. Use a parameter to specify the name of the requested authentication middleware instance.
4. Add an action that handles the responses from external services.

- Call the **AuthenticationManager.GetExternalLoginInfoAsync** method to get login information from the external identity provided by the service.
- Use the **SignInManager.ExternalSignInAsync** method to sign in to Kentico based on external data.

5. Add logic that creates new users in the Kentico database if the first sign-in using the external data fails:
   a. Prepare a new **Kentico.Membership.User** object based on the external login data. You must enable the user object's **IsExternal** property.
   b. Call the **UserManager.CreateAsync** method to create the user in the Kentico database.
   c. Call the **UserManager.AddLoginAsync** method to create a mapping between the new user and the given authentication provider.
   d. (Optional) Add the user to any required roles. See Managing user roles from the MVC application to learn how to add users to Kentico roles using the ASP.NET Identity API.

External authentication example

```
        /// <summary>
        /// Redirects authentication requests to an external service.
        /// Posted parameters include the name of the requested authentication
middleware instance and a return URL.
        /// </summary>
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public ActionResult RequestSignIn(string provider, string returnUrl)
        {
            // Requests a redirect to the external sign-in provider
            // Sets a return URL targeting an action that handles the response
            // Fill in the name of your controller
            return new ChallengeResult(provider, Url.Action("SignInCallback",
"ExternalAuthentication", new { ReturnUrl = returnUrl }));
        }

        /// <summary>
        /// Handles responses from external authentication services.
        /// </summary>
        [AllowAnonymous]
        public async Task<ActionResult> SignInCallback(string returnUrl)
        {
            // Extracts login info out of the external identity provided by the
service
            ExternalLoginInfo loginInfo = await AuthenticationManager.
GetExternalLoginInfoAsync();

            // If the external authentication fails, displays a view with
appropriate information
            if (loginInfo == null)
            {
                return View("ExternalAuthenticationFailure");
            }

            // Attempts to sign in the user using the external login info
            SignInStatus result = await SignInManager.ExternalSignInAsync
(loginInfo, isPersistent: false);
            switch (result)
            {
                // Success occurs if the user already exists in the Kentico
database and has signed in using the given external service
                case SignInStatus.Success:
                    // Redirects to the original return URL
```

```
                    return RedirectToLocal(returnUrl);

                // The 'LockedOut' status occurs if the user exists in Kentico,
but is not enabled
                case SignInStatus.LockedOut:
                    // Returns a view informing the user about the locked account
                    return View("Lockout");

                case SignInStatus.Failure:
                default:
                    // Attempts to create a new user in Kentico if the
authentication failed
                    IdentityResult userCreation = await CreateExternalUser
(loginInfo);

                    // Attempts to sign in again with the new user created based
on the external authentication data
                    result = await SignInManager.ExternalSignInAsync(loginInfo,
isPersistent: false);

                    // Verifies that the user was created successfully and was
able to sign in
                    if (userCreation.Succeeded && result == SignInStatus.Success)
                    {
                        // Redirects to the original return URL
                        return RedirectToLocal(returnUrl);
                    }

                    // If the user creation was not successful, displays
corresponding error messages
                    foreach (string error in userCreation.Errors)
                    {
                        ModelState.AddModelError(String.Empty, error);
                    }
                    return View();

                    // Optional extension:
                    // Send the loginInfo data as a view model and create a form
that allows adjustments of the user data.
                    // Allows visitors to resolve errors such as conflicts with
existing usernames in Kentico.
                    // Then post the data to another action and attempt to create
the user account again.
                    // The action can access the original loginInfo using the
AuthenticationManager.GetExternalLoginInfoAsync() method.
            }
        }

        /// <summary>
        /// Creates users in Kentico based on external login data.
        /// </summary>
        private async Task<IdentityResult> CreateExternalUser(ExternalLoginInfo
loginInfo)
        {
            // Prepares a new user entity based on the external login data
            Kentico.Membership.User user = new User
            {
                UserName = ValidationHelper.GetSafeUserName(loginInfo.
DefaultUserName ?? loginInfo.Email, SiteContext.CurrentSiteName),
                Email = loginInfo.Email,
```

```
                Enabled = true, // The user is enabled by default
                IsExternal = true // IsExternal must always be true for users
created via external authentication
                // Set any other required user properties using the data
available in loginInfo
            };

            // Attempts to create the user in the Kentico database
            IdentityResult result = await UserManager.CreateAsync(user);
            if (result.Succeeded)
            {
                // If the user was created successfully, creates a mapping
between the user and the given authentication provider
                result = await UserManager.AddLoginAsync(user.Id, loginInfo.
Login);
            }

            return result;
        }

        /// <summary>
        /// Redirects to a specified return URL if it belongs to the MVC website
or to the site's home page.
        /// </summary>
        private ActionResult RedirectToLocal(string returnUrl)
        {
            if (Url.IsLocalUrl(returnUrl))
            {
                return Redirect(returnUrl);
            }

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

        /// <summary>
        /// Custom type that processes the result of an action method.
        /// </summary>
        internal class ChallengeResult : HttpUnauthorizedResult
        {
            public ChallengeResult(string provider, string redirectUri)
            {
                LoginProvider = provider;
                RedirectUri = redirectUri;
            }

            public string LoginProvider { get; set; }
            public string RedirectUri { get; set; }

            // Called to run the action result using the specified controller
context
            public override void ExecuteResult(ControllerContext context)
            {
                var properties = new AuthenticationProperties { RedirectUri =
RedirectUri };

                // Adds information into the response environment that causes the
specified authentication middleware to challenge the caller
                context.HttpContext.GetOwinContext().Authentication.Challenge
(properties, LoginProvider);
```

```
            }
        }
```

6. Design the user interface required for external authentication on your website

    - Add buttons onto your website's sign-in page that target the *RequestSignIn* action. Supply parameters that specify the provider type for each authentication service and optionally a return URL.

      Example of external authentication buttons (Razor view)

      ```
      @using Microsoft.Owin.Security

      @{
          @* Gets a collection of the authentication services registered in your
      application's startup class *@
          var authServices = Context.GetOwinContext().Authentication.
      GetExternalAuthenticationTypes();

          @* Generates a form with buttons targeting the RequestSignIn action.
      Optionally pass a redirect URL as a parameter. *@
          using (Html.BeginForm("RequestSignIn", "ExternalAuthentication"))
          {
              @Html.AntiForgeryToken()
              <p>
                  @foreach (AuthenticationDescription p in authServices)
                  {
                      <button type="submit" class="btn btn-default" id="@p.
      AuthenticationType" name="provider" value="@p.AuthenticationType" title="
      Sign in using your @p.Caption account">@p.AuthenticationType</button>
                  }
              </p>
          }
      }
      ```

    - Create a view for displaying information in situations where the external authentication fails *(ExternalAuthenticationFailure* in the example).
    - Create a view for displaying information in situations where the external authentication succeeds, but the matching Kentico user account is disabled *(Lockout* in the example).
    - Create a view for the *SignInCallback* action.

      > In the most basic version of the *SignInCallback* view, you only need to display error messages in situations where the external authentication succeeds, but Kentico is unable to create a matching user account.
      >
      > You can implement an extended version with an editing form that allows visitors to adjust and resubmit the data of the new user account (using a strongly typed view that receives the external login data from the action).

Visitors can now sign in on the website through the external authentication services that you registered in the *Startup* class. Upon successful authentication, the system creates a corresponding user account in the shared Kentico database, along with a mapping between the user and the given authentication provider.