

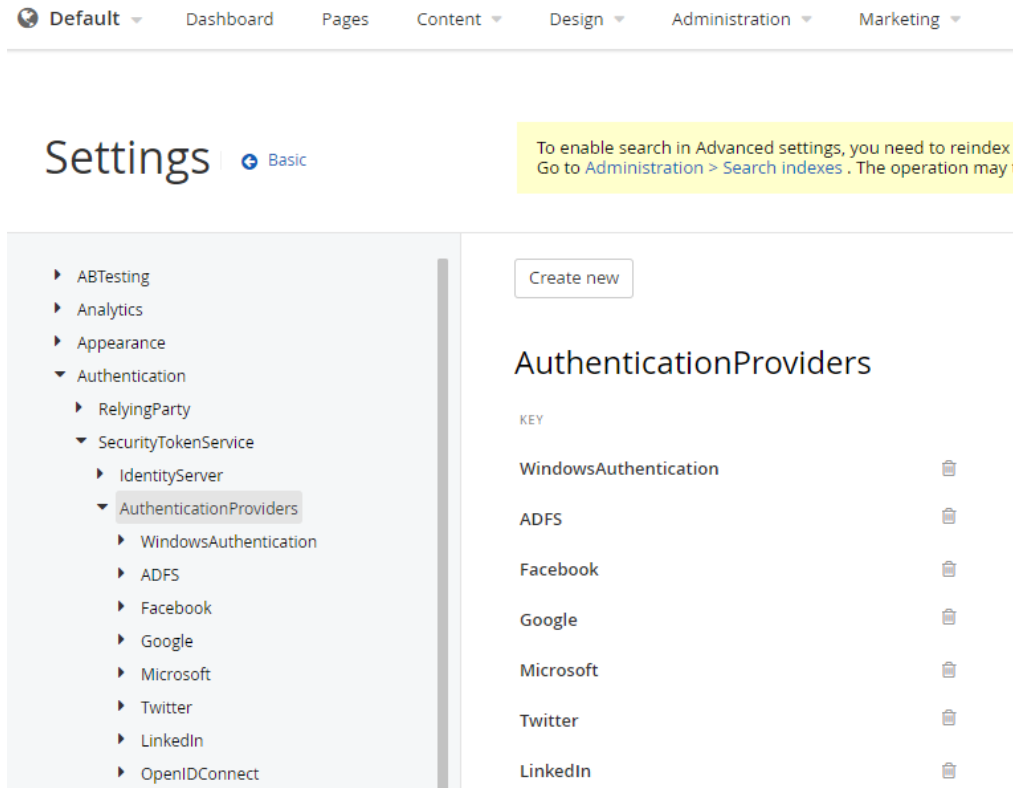
14.0 Custom external authentication provider samples

The wiki demonstrates the recommended approach when integrating with external identity providers using custom logic.








 THE SAMPLE HERE IS VALID ONLY IF YOU ARE USING THE "DEFAULT" AUTHENTICATION PROTOCOL.

 You can view the source code for all the samples in this repo: <https://github.com/todorm85/sitefinity-custom-external-authentication-samples>

Sitefinity ships out of box (OOB) with several external authentication providers (EAP) for several popular Identity Providers (IP or sometimes called STS) like Google, Facebook, Microsoft. It also has one configurable generic EIP for authenticating with systems that follow the widely adopted OpenID Connect (OIDC) authentication protocol. Customizing the OIDC provider is described in this article: [link](#)



The screenshot shows the Sitefinity administration interface. At the top, there's a navigation bar with 'Default' selected and other tabs like 'Dashboard', 'Pages', 'Content', 'Design', 'Administration', and 'Marketing'. Below this, the 'Settings' section is active, with a 'Basic' sub-tab. A yellow warning box states: 'To enable search in Advanced settings, you need to reindex. Go to Administration > Search indexes. The operation may take some time.' On the left, a sidebar menu lists various settings categories, with 'AuthenticationProviders' under 'Authentication' highlighted. The main content area is titled 'AuthenticationProviders' and features a 'Create new' button. Below the title, there's a table with a 'KEY' header and a list of providers: WindowsAuthentication, ADFS, Facebook, Google, Microsoft, Twitter, and LinkedIn. Each provider has a trash icon to its right for deletion.

KEY	
WindowsAuthentication	
ADFS	
Facebook	
Google	
Microsoft	
Twitter	
LinkedIn	

If you cannot use any of the OOB EAPs you can implement a fully custom one by following the samples in this wiki. Please note that for the purpose of the demo the samples here are not for production use as there are additional security implementations omitted for brevity.

The job of the EAP is to authenticate the user and then pass a claims based identity to Sitefinity with a few required claims. The wiki contains two samples - one where the login page is on a remote server (Remote Login) and one where the login page is in Sitefinity and credentials provided by the user are sent to a remote server for verification (Local Login).

Sample 1 - Remote Login

- Create the configuration class - it is used to contain custom settings that you might want to configure via advanced settings view

RemoteLoginExternalAuthenticationProviderOptions

```
using Microsoft.Owin.Security;

namespace AutehnticationSamples
{
    public class RemoteLoginExternalAuthenticationProviderOptions : AuthenticationOptions
    {
        public RemoteLoginExternalAuthenticationProviderOptions(string authenticationType) : base
(authenticationType)
        {
        }

        public string IdentityProviderAddress { get; set; }
        public string SignInAsAuthenticationType { get; set; }
    }
}
```

- Create the authentication handler class - this is where all the custom logic resides, the class should derive from `AuthenticationHandler<RemoteLoginExternalAuthenticationProviderOptions>`

RemoteLoginExternalAuthenticationProviderHandler

```
using Microsoft.Owin.Security;
using Microsoft.Owin.Security.Infrastructure;
using SitefinityWebApp.AuthProvider;
using System.Security.Claims;
using System.Threading.Tasks;
using System.Web;
using Telerik.Sitefinity.Security.Claims;

namespace AutehnticationSamples
{
    public class RemoteLoginExternalAuthenticationProviderHandler :
AuthenticationHandler<RemoteLoginExternalAuthenticationProviderOptions>
    {
        public AuthPropertiesSerializer propertiesSerializer = new AuthPropertiesSerializer();

        // handle signin requests - redirect the user browser to the external identity provider for
authentication
        protected override Task ApplyResponseChallengeAsync()
        {
            // challenges change the status code to 401
            if (Response.StatusCode != 401)
            {
                return Task.CompletedTask;
            }

            AuthenticationResponseChallenge challenge = Helper.LookupChallenge(Options.AuthenticationType,
Options.AuthenticationMode);
            if (challenge == null)
            {
                return Task.CompletedTask;
            }

            var redirectUri = challenge.Properties.RedirectUri;

            // you should also cryptographically protect the state or do not send the entire obejct
            // but store it in db and send only an identifier associated with it
            var state = this.propertiesSerializer.Serialize(challenge.Properties);
            redirectUri = redirectUri + "?state=" + HttpUtility.UrlEncode(state);

            // the url of the identity provider where authentication should take place
            var ipUri = Options.IdentityProviderAddress;
            ipUri = ipUri + "/authorize?callbackUri=" + HttpUtility.UrlEncode(redirectUri);

            Response.Redirect(ipUri);
        }
    }
}
```

```

        return Task.CompletedTask;
    }

    public override async Task<bool> InvokeAsync()
    {
        AuthenticationTicket ticket = TryProcessResponseFromIP();
        if (ticket != null)
        {
            if (ticket.Identity != null)
            {
                Request.Context.Authentication.SignIn(ticket.Properties, ticket.Identity);
            }
        }

        return false;
    }

    // try to handle the redirect back from the external identity provider with the authentication token
    and return null if failed or
    // the generated identity if successful
    protected AuthenticationTicket TryProcessResponseFromIP()
    {
        // extract user information from redirect request back from IP
        // token should be cryptographically protected
        // also avoid sending the token in the request query, instead a POST request with the token in the
        body should be implemented for increased security
        // or something similar to authorization code flow of OAuth2.0 protocol could be implemented
        var token = HttpUtility.UrlDecode(Context.Request.Query.Get("token"));
        var state = HttpUtility.UrlDecodeToBytes(Context.Request.Query.Get("state"));

        // return null if unsuccessful
        if (token == null || state == null)
        {
            return null;
        }

        // otherwise generate an identity with calims required by Sitefinity
        // sub identifier is required it must be unique for each user on the external system
        // email is required if turned on in the advanced settings
        var tokenParts = token.Split(':');
        var externalId = tokenParts[0];
        var email = tokenParts[1];
        ClaimsIdentity identity = new ClaimsIdentity(Options.SignInAsAuthenticationType);
        identity.AddClaim(new Claim("sub", externalId));
        identity.AddClaim(new Claim(SitefinityClaimTypes.ExternalUserEmail, email));

        // optionally you can also add profile picture claim - SitefinityClaimTypes.ExternalUserPictureUrl
        // to map any other information from the remote server about the user to the automatically created
        user profile use
        // the following KB: https://knowledgebase.progress.com/articles/Article/Authentication-What-types-of-claims-can-be-mapped-to-profile-fields-when-using-a-custom-external-identity-provider

        // state must be further cryptographically protected
        var props = this.propertiesSerializer.Deserialize(state);
        var ticket = new AuthenticationTicket(identity, props);

        return ticket;
    }

    // handle signout requests
    protected override Task ApplyResponseGrantAsync()
    {
        AuthenticationResponseRevoke signout = Helper.LookupSignOut(Options.AuthenticationType, Options.
AuthenticationMode);
        if (signout != null)
        {
            // send a request to the IP to signout the user
            Response.Redirect(Options.IdentityProviderAddress + "/signoutCurrent" + "?redirectUri=" +
HttpUtility.UrlEncode(signout.Properties.RedirectUri));
        }
    }

```

```

        return Task.CompletedTask;
    }

    // the auth middleware is concerned with delegating the authentication to a remote system
    protected override Task<AuthenticationTicket> AuthenticateCoreAsync()
    {
        return Task.FromResult<AuthenticationTicket>(null);
    }
}

```

The handler depends on a serializer class for the auth properties, here's the implementation

AuthPropertiesSerializer

```

using Microsoft.Owin.Security;
using System;
using System.Collections.Generic;
using System.IO;

namespace SitefinityWebApp.AuthProvider
{
    public class AuthPropertiesSerializer
    {
        private const int FormatVersion = 1;

        public byte[] Serialize(AuthenticationProperties model)
        {
            using (var memory = new MemoryStream())
            {
                using (var writer = new BinaryWriter(memory))
                {
                    Write(writer, model);
                    writer.Flush();
                    return memory.ToArray();
                }
            }
        }

        public AuthenticationProperties Deserialize(byte[] data)
        {
            using (var memory = new MemoryStream(data))
            {
                using (var reader = new BinaryReader(memory))
                {
                    return Read(reader);
                }
            }
        }

        public static void Write(BinaryWriter writer, AuthenticationProperties properties)
        {
            if (writer == null)
            {
                throw new ArgumentNullException("writer");
            }
            if (properties == null)
            {
                throw new ArgumentNullException("properties");
            }

            writer.Write(FormatVersion);
            writer.Write(properties.Dictionary.Count);
            foreach (var kv in properties.Dictionary)
            {
                writer.Write(kv.Key);
                writer.Write(kv.Value);
            }
        }
    }
}

```

```

public static AuthenticationProperties Read(BinaryReader reader)
{
    if (reader == null)
    {
        throw new ArgumentNullException("reader");
    }

    if (reader.ReadInt32() != FormatVersion)
    {
        return null;
    }
    int count = reader.ReadInt32();
    var extra = new Dictionary<string, string>(count);
    for (int index = 0; index != count; ++index)
    {
        string key = reader.ReadString();
        string value = reader.ReadString();
        extra.Add(key, value);
    }
    return new AuthenticationProperties(extra);
}
}
}

```

- create the custom authentication middleware class

RemoteLoginExternalAuthenticationProviderMiddleware

```

using Microsoft.Owin;
using Microsoft.Owin.Security.Infrastructure;

namespace AuthnticationSamples
{
    public class RemoteLoginExternalAuthenticationProviderMiddleware :
        AuthenticationMiddleware<RemoteLoginExternalAuthenticationProviderOptions>
    {
        public RemoteLoginExternalAuthenticationProviderMiddleware(OwinMiddleware next,
            RemoteLoginExternalAuthenticationProviderOptions options) : base(next, options)
        {
        }

        protected override AuthenticationHandler<RemoteLoginExternalAuthenticationProviderOptions>
        CreateHandler()
        {
            return new RemoteLoginExternalAuthenticationProviderHandler();
        }
    }
}

```

- register the custom authentication middleware with Sitefinity

Register custom auth middleware with Sitefinity

```
using Owin;
using System;
using System.Collections.Generic;
using Telerik.Sitefinity.Authentication;
using Telerik.Sitefinity.Authentication.Configuration.SecurityTokenService.ExternalProviders;

namespace AuthnticationSamples
{
    public class AuthenticationProvidersInitializerExtender : AuthenticationProvidersInitializer
    {
        public override Dictionary<string, Action<IAppBuilder, string, AuthenticationProviderElement>>
        GetAdditionalIdentityProviders()
        {
            var providers = base.GetAdditionalIdentityProviders();

            // 'CustomIP' is the name of the external authentication provider as configured in the Advanced
            settings
            providers.Add("CustomIP", (IAppBuilder app, string signInAsType, AuthenticationProviderElement
            providerConfig) =>
            {
                var options = new RemoteLoginExternalAuthenticationProviderOptions(providerConfig.Name)
                {
                    AuthenticationMode = Microsoft.Owin.Security.AuthenticationMode.Passive,
                    IdentityProviderAddress = providerConfig.GetParameter("identityProviderAddress"),
                    SignInAsAuthenticationType = signInAsType
                };

                app.Use(typeof(RemoteLoginExternalAuthenticationProviderMiddleware), options);
            });

            return providers;
        }
    }
}
```

Global.asax

```
using System;
using System.Web.Security;
using Telerik.Microsoft.Practices.Unity;
using Telerik.Sitefinity.Abstractions;
using Telerik.Sitefinity.Authentication;

namespace AuthnticationSamples
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            AuthenticationModule.Initialized += this.AuthenticationModule_Initialized;
        }

        private void AuthenticationModule_Initialized(object sender, EventArgs e)
        {
            ObjectFactory.Container.RegisterType<AuthenticationProvidersInitializer,
            AuthenticationProvidersInitializerExtender>(new ContainerControlledLifetimeManager());
        }
    }
}
```

- Finally start Sitefinity and go to advanced settings and create a new authentication provider element

Settings

Basic

To enable search in Advanced settings, you need to reindex the :
Go to [Administration > Search indexes](#) . The operation may take

- ▶ ABTesting
- ▶ Analytics
- ▶ Appearance
- ▼ Authentication
 - ▶ RelyingParty
 - ▼ SecurityTokenService
 - ▶ IdentityServer
 - ▼ AuthenticationProviders
 - ▶ WindowsAuthentication
 - ▶ ADFS
 - ▶ Facebook
 - ▶ Google
 - ▶ Microsoft
 - ▶ Twitter
 - ▶ LinkedIn
 - ▶ OpenIDConnect

Create new

- AuthenticationProviderElement
- WindowsAuthenticationProviderElement
- AdfsAuthenticationProviderElement
- FacebookAuthenticationProviderElement
- GoogleAuthenticationProviderElement
- MicrosoftAuthenticationProviderElement
- TwitterAuthenticationProviderElement
- LinkedInAuthenticationProviderElement
- OpenIDConnectAuthenticationProviderElement

Microsoft



Twitter



LinkedIn



- use the same name that you used when registering the custom authentication provider in the AuthenticationProvidersInitializerExtender, in this sample "CustomIP"

Settings | Basic

To enable search in Advanced settings, you need to reindex the settings content. Go to [Administration > Search indexes](#). The operation may take several minutes.

- ▶ ABTesting
- ▶ Analytics
- ▶ Appearance
- ▼ Authentication
 - ▶ RelyingParty
 - ▼ SecurityTokenService
 - ▶ IdentityServer
 - ▼ AuthenticationProviders
 - ▶ WindowsAuthentication
 - ▶ ADFS
 - ▶ Facebook
 - ▶ Google
 - ▶ Microsoft
 - ▶ Twitter
 - ▶ LinkedIn
 - ▶ OpenIDConnect
 - ▼ CustomIP
 - ▶ Parameters
 - ▶ LocalLoginCustomIP
 - ▶ OAuthServer
 - ▶ Providers
 - ▶ Blogs
 - ▶ CommentsModule
 - ▶ Content
 - ▶ ContentLinks
 - ▶ ContentLocations
 - ContentPlugins
 - ▶ ContentView
 - ContextualHelp
 - ▶ Controls
 - ▶ ControlTemplates
 - ▶ Cultures
 - ▶ CustomFields
 - ▶ Dashboard
 - ▶ Data

Save changes

Delete

CustomIP

Saved on the file system

Data provider (?)

Default

Data provider for external login users.

Name (?)

CustomIP

External provider name.

Title (?)

CustomIP

Caption in Sitefinity for the provider (e.g. Login button).

Enabled (?)



Specifies whether the provider is enabled or not.

Auto assigned roles (?)

Administrators,BackendUsers

Specifies the roles that will be automatically assigned to users registered with this provider (separated with comma)

Link CSS class (?)

btn btn-default

Specifies the CSS class that will be applied to the provider's link.

Require email claim from this provider (?)



Users can login only if their email is included in the provided claim.

Save changes

Delete

- next add the properties that you want to be able to configure via advanced settings and that you have added to the RemoteLoginExternalAuthenticationProviderOptions class
 - in this demo we only want to be able to configure the address of the external Identity Provider

Settings [Basic](#)

To enable search in Advanced settings, you need to reindex the settings content. Go to [Administration > Search indexes](#) . The operation may take several minutes.

- ▶ ABTesting
- ▶ Analytics
- ▶ Appearance
- ▼ Authentication
 - ▶ RelyingParty
 - ▼ SecurityTokenService
 - ▶ IdentityServer
 - ▼ AuthenticationProviders
 - ▶ WindowsAuthentication
 - ▶ ADFS
 - ▶ Facebook
 - ▶ Google
 - ▶ Microsoft
 - ▶ Twitter
 - ▶ LinkedIn
 - ▶ OpenIDConnect
 - ▼ CustomIP
 - ▼ Parameters
 - identityProviderAddress**
 - ▶ LocalLoginCustomIP
 - ▶ OAuthServer

Save changes

Delete

identityProviderAddress

Saved on the file system

Key

identityProviderAddress

Value (?)

http://sf0:52833/customip

Save changes

Delete

Sample 2 - Local Login

If you want to host the login page locally and only send the credentials to the IP for verification use this sample for the authentication handler class. Here for the purpose of the demo we host the login page in the handler itself.

Authentication Handler for local login

```

using Microsoft.Owin.Security;
using Microsoft.Owin.Security.Infrastructure;
using SitefinityWebApp.AuthProvider;
using System;
using System.Net.Http;
using System.Security.Claims;
using System.Threading.Tasks;
using System.Web;
using Telerik.Sitefinity.Security.Claims;
using Telerik.Sitefinity.Web;

namespace AuthnticationSamples
{
    public class LocalLoginExternalAuthenticationProviderHandler :
        AuthenticationHandler<LocalLoginExternalAuthenticationProviderOptions>
    {
        private const string LocalCustomLoginPagePath = "custom-login";
        public AuthPropertiesSerializer propertiesSerializer = new AuthPropertiesSerializer();
    }
}

```

```

        // this middleware is concerned only with proper redirection to a custom login page that contains the
authentication logic
        protected override async Task<AuthenticationTicket> AuthenticateCoreAsync()
        {
            return null;
        }

        // handle signin requests - redirect the user browser to the external identity provider for
authentication
        protected override Task ApplyResponseChallengeAsync()
        {
            // challenges change the status code to 401
            if (Response.StatusCode != 401)
            {
                return Task.CompletedTask;
            }

            AuthenticationResponseChallenge challenge = Helper.LookupChallenge(Options.AuthenticationType,
Options.AuthenticationMode);
            if (challenge == null)
            {
                return Task.CompletedTask;
            }

            // redirect to a login page, you can redirect to any sitefinity page with custom widget
            // for simplicity we will handle the page request in the same auth middleware
            Response.Redirect(
                UrlPath.ResolveAbsolutePath(
                    "~/ " + LocalCustomLoginPagePath + "?redirectUrl=" + HttpUtility.UrlEncode(challenge.
Properties.RedirectUri)) + "&signInAs=" + HttpUtility.UrlEncode(Options.SignInAsAuthenticationType));

            return Task.CompletedTask;
        }

        public override async Task<bool> InvokeAsync()
        {
            // for brevity we handle the login page processing in the middleware but that logic could be
extracted in any codebehind of a widget or page outside of this middleware
            if (Request.Uri.GetLeftPart(UriPartial.Path).ToLower() == UrlPath.ResolveAbsolutePath("~/ " +
LocalCustomLoginPagePath).ToLower())
            {
                await HandleCustomLoginPageRequest();
                return true; // stop further processing
            }

            return false;
        }

        private async Task HandleCustomLoginPageRequest()
        {
            if (Request.Method == "GET")
            {
                // render login page
                await Response.WriteAsync($"<!DOCTYPE html>

<html>
<body>
<h2>Custom Sitefinity Login</h2>

<form action='{Request.Uri}' method='POST'>
    <label for='user'>User:</label><br>
    <input type='text' id='user' name='user'><br>
    <label for='pass'>Pass:</label><br>
    <input type='password' id='pass' name='pass'><br><br>
    <input type='submit' value='Submit'>
</form>
</body>
</html>
");
            }
            else if (Request.Method == "POST")

```


DummyIdentityProviderMiddleware

```
using Microsoft.Owin;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Web;

namespace AutehnticationSamples.DummyIdentityProvider
{
    /// <summary>
    /// Identity providers are separate web apps, but for the purpose of the demo we are hosting it inside
    Sitefinity
    /// </summary>
    public class DummyIdentityProviderMiddleware : OwinMiddleware
    {
        public DummyIdentityProviderMiddleware(OwinMiddleware next) : base(next)
        {
        }

        public override Task Invoke(IOwinContext context)
        {
            if (context.Request.Path.Value.ToLower().StartsWith("/customip/authorize"))
            {
                // process user authentication and generate response to send back to Sitefinity
                var dummyAuthToken = HttpUtility.UrlEncode("idl:user1@test.test");
                context.Response.Redirect(context.Request.Query.Get("callbackUri") + "&token=" +
dummyAuthToken);
                return Task.CompletedTask;
            }
            else if (context.Request.Path.Value.ToLower().StartsWith("/customip/signout"))
            {
                // signout the user from the Identity Provider and redirect back to Sitefinity
                return Task.CompletedTask;
            }
            else if (context.Request.Path.Value.ToLower().StartsWith("/customip/remoteverification"))
            {
                // verify the credentials sent by Sitefinity backend and return user unique id if successful
                // this endpoint is used by the LocalLogin external auth provider
                // for the purpose of the sample the endpoint will return the hash of the user email as unique
                identifier
                var authHeader = context.Request.Headers.Get("Authorization");
                var encoding = Encoding.GetEncoding("iso-8859-1");
                var credentials = encoding.GetString(Convert.FromBase64String(authHeader.Split(' ')[1]));

                int separator = credentials.IndexOf(':');
                string userEmail = credentials.Substring(0, separator);
                context.Response.WriteAsync(new HMACMD5().ComputeHash(encoding.GetBytes(userEmail)));
                return Task.CompletedTask;
            }
            else
            {
                return Next.Invoke(context);
            }
        }
    }
}
```

- create a startup class to register the dummy identity provider

Startup.cs

```
using AutehnticationSamples.DummyIdentityProvider;
using Owin;
using Telerik.Sitefinity.Owin;

namespace AutehnticationSamples
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            // Register default Sitefinity middlewares in the pipeline
            app.UseSitefinityMiddleware();

            app.Use(typeof(DummyIdentityProviderMiddleware));
        }
    }
}
```

- register the startup class in the web.config

```
</sectionGroup>
</configSections>
<appSettings>
  <add key="TestLocalization" value="false" />
  <add key="enableSimpleMembership" value="false" />
  <add key="vs:EnableBrowserLink" value="false" />
  <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
  <add key="aspnet:UseHostHeaderForRequestUrl" value="true" />
  <add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />
  <add key="sf:serviceStackEnableFeatures" value="Json,Html" />
  <add key="sf:enableRendererFeatures14" value="true" />
  <!--<add key="sf:enableRendererDiagnostics" value="true" />-->
  <!--<add key="sf:methodPerformanceLoggingPath" value="~/App_Data/Sitefinity/Logs"
  <add key="sf:methodPerformanceLogThresholdInMilliseconds" value="500" />-->
  <add key="owin:appStartup" value="AutehnticationSamples.Startup" />
</appSettings>
<system.web.webPages.razor>
```