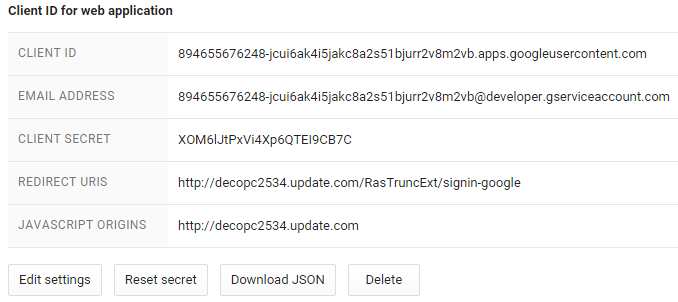
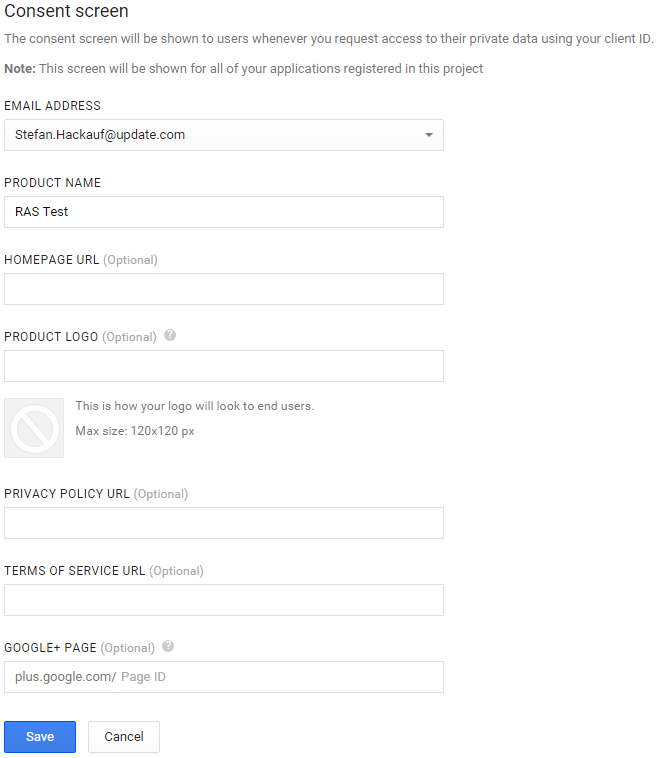
# MVC 4 – Google, Microsoft, Twitter und Facebook Login

In diesem Dokument wird beschrieben, wie innerhalb einer MVC 4 Anwendung die externen Benutzerquellen von Google, Microsoft, Twitter und Facebook mit Hilfe der MVC 5 Technik Owin abgefragt werden können.

## Google Schnittstelle administrieren

Um mit einer externen Anwendung über die OAuth Schnittstelle die Google Benutzer zu berücksichtigen muss innerhalb von Google diese Anwendung registriert werden. Hierfür meldet man sich an der Google Developers Console (<https://console.developers.google.com/>) an und erstellt ein Projekt. Im Untermenü APIs fügt man die Google+ API hinzu. Im Untermenü Credentials registriert man abschließend eine OAuth Verbindung und im Untermenü Consent screen fügt man Basisdaten seiner Anwendung hinzu:

## MVC 4 Anwendung erstellen

* Im Visual Studio 2012 ein „ASP.NET MVC 4 Web Application“ Projekt erzeugt und auf der nächsten Seite „Empty“ auswählen.
* Über NuGet folgende Packages hinzufügen:
  + Microsoft ASP.NET Identity Core
  + Microsoft ASP.NET Identity Owin
  + Microsoft.Owin.Host.SystemWeb
  + Microsoft.Owin.Security.Cookies
  + Microsoft.Owin.Security.Facebook
  + Microsoft.Owin.Security.Google
  + Microsoft.Owin.Security.MicrosoftAccount
  + Microsoft.Owin.Security.OAuth
  + Microsoft.Owin.Security.Twitter
  + Microsoft.Owin.Security
  + Microsoft.Owin
  + OWIN
* Klasse OwinStartup erstellen, in der die externen Providers registriert werden.
* Anmeldedialog erstellen
* Anmeldedialog als Startdialog definieren

### OwinStartup

Im Root-Verzeichnis der MVC 4 Anwendung muss eine Klasse für die Registrierung der externen Provider für das Owin Framework angelegt werden. Hierfür wird eine entsprechende Klasse (im Beispiel Startup) angelegt und über das Attribut OwinStartup dem Framework die Existenz dieser Klasse mitgeteilt. Die Klasse beinhaltet eine Konfigurationsmethode der das Interface IAppBuilder übergeben werden muss.

using System;

using Microsoft.AspNet.Identity;

using Microsoft.Owin;

using Mvc4ExternalLogin;

using Owin;

[assembly: OwinStartup(typeof(Startup))]

namespace Mvc4ExternalLogin

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

app.UseTwoFactorSignInCookie(DefaultAuthenticationTypes.TwoFactorCookie, TimeSpan.FromMinutes(5));

app.UseTwoFactorRememberBrowserCookie(DefaultAuthenticationTypes.TwoFactorRememberBrowserCookie);

app.UseGoogleAuthentication(

clientId: "894655676248-kfre2gebipqb11fdcb5k2le7p06aet44.apps.googleusercontent.com",

clientSecret: "gWovTYa8Dnwr4WOi2oSCrKA6");

// app.UseMicrosoftAccountAuthentication(clientId: "", clientSecret: "");

// app.UseTwitterAuthentication(consumerKey: "", consumerSecret: "");

// app.UseFacebookAuthentication(appId: "", appSecret: "");

}

}

}

Wird der Code für das Owin Framework in MVC 5 generiert, wird die Registrierungsklasse für eine bessere Übersicht aufgeteilt. Hierbei wird die Startup Klasse im Basisverzeichnis partial verwendet und die Konfigurationsmethode in ein Unterverzeichnis App\_Start mit dem Dateinamen „Startup.Auth.cs“ ausgelagert. Dies hat den Vorteil, dass alle Konfigurationen, die während des Anwendungsstarts durchgeführt werden, an einer Stelle zu finden sind.

Datei Startup.cs

using Microsoft.Owin;

using Mvc4ExternalLogin;

using Owin;

[assembly: OwinStartup(typeof(Startup))]

namespace Mvc4ExternalLogin

{

public partial class Startup

{

public void Configuration(IAppBuilder app)

{

ConfigureAuth(app);

}

}

}

Datei App\_Start\Startup.Auth.cs

using Microsoft.AspNet.Identity;

using Owin;

// ReSharper disable CheckNamespace

namespace Mvc4ExternalLogin

// ReSharper restore CheckNamespace

{

public partial class Startup

{

public void ConfigureAuth(IAppBuilder app)

{

app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

app.UseTwoFactorSignInCookie(DefaultAuthenticationTypes.TwoFactorCookie, TimeSpan.FromMinutes(5));

app.UseTwoFactorRememberBrowserCookie(DefaultAuthenticationTypes.TwoFactorRememberBrowserCookie);

app.UseGoogleAuthentication(

clientId: "894655676248-kfre2gebipqb11fdcb5k2le7p06aet44.apps.googleusercontent.com",

clientSecret: "gWovTYa8Dnwr4WOi2oSCrKA6");

// app.UseMicrosoftAccountAuthentication(clientId: "", clientSecret: "");

// app.UseTwitterAuthentication(consumerKey: "", consumerSecret: "");

// app.UseFacebookAuthentication(appId: "", appSecret: "");

}

}

}

### Anmeldedialog

Zuerst muss im Verzeichnis Controllers ein Login Kontroller erstellt werden. Hierbei ist es am einfachsten zuerst einen lehren MVC Kontroller zu erstellen und diesen anschließend mit Leben zu füllen. Anschließend wird eine View benötigt, die sinnvollerweise im Verzeichnis Views\Login\Index.cshtml zu finden ist. Hierbei muss die MVC Logik berücksichtigt werden (Verzeichnisname Login ist der Name des Kontrollers und Index.cshtml ist der Name der Methode).

Um die Kommunikation zwischen Server und Client zu vereinfachen ist es am sinnvollsten im Verzeichnis Models ein LoginViewModel anzulegen. Dies kann beispielsweise dazu verwendet werden, Meldungen an den Client zu übertragen.

namespace Mvc4ExternalLogin.Models

{

public class LoginViewModel

{

public LoginViewModel(string info = "")

{

Info = info;

}

public string Info { get; set; }

}

}

Im Kontroller werden nun mehrere Methoden für den Login Prozess benötigt.

* Index  
  Methode wird beim Starten des Dialoges aufgerufen und füllt die Startwerte. Außerdem kann diese Methode für die Übergabe von Fehlermeldungen verwendet werden.
* ExternalLogin  
  Methode wird vom Dialog aufgerufen, um einen externen Provider aufzurufen.
* ExternalLoginCallback  
  Methode wird vom externen Provider beim Rücksprung in die Anwendung aufgerufen und liest die benötigten Daten (beispielsweise die Email) aus.

Außerdem wird eine Hilfeklasse benötigt (im Beispiel ChallengeResult), die als Container für die externen Daten verwendet wird.

**Methode Index**

Die Methode erhält in diesem Beispiel zwei Aufgaben. Sie wird beim Starten des Dialoges initial aufgerufen und liefert Informationen innerhalb der Verarbeitung an den Dialog. Die während des Anwendungsstarts initialisierten Providers müssen nicht geladen werden, da sie sich bereits in der Session befinden und der GUI verwendet werden können.

[AllowAnonymous]

public ActionResult Index(string info = "")

{

return View(new LoginViewModel(info));

}

Das Attribut AllowAnonymous wird benötigt, da der Login Dialog anonym (also ohne angemeldeten Benutzer) aufgerufen werden darf.

**Methode ExternalLogin**

Die Methode wird nach der Auswahl eines Providers durch den Anwender aufgerufen und ruft diesen externen Provider auf.

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLogin(string provider)

{

return new ChallengeResult(provider, Url.Action("ExternalLoginCallback", "Login"));

}

Über die Url.Action wird die Rücksprung URL für den externen Provider generiert.

**Methode ExternalLoginCallback**

Diese Methode wird vom externen Provider mit dem Ergebnis der Anmeldung aufgerufen. Hierbei wurde dem Provider dieser Rücksprung über die Methode ExternalLogin mitgeteilt.

[AllowAnonymous]

public async Task<ActionResult> ExternalLoginCallback()

{

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync();

if (loginInfo == null)

{

return RedirectToAction("Index", new { info = "No login information found!" });

}

var email = loginInfo.Email;

// Hier gegebenenfalls die Email mit der internen Benutzerverwaltung prüfen

// und anschließend die eigentliche Anwendung aufrufen.

return RedirectToAction("Index", new { info = "[User: " + email + "]" });

}

Im Fehlerfall wird die Index Methode mit einer entsprechenden Meldung aufgerufen. Der Erfolgsfall ist hier nur symbolisch umgesetzt, da hier je nach Anwendung eine individuelle Codierung erfolgen muss.

**Klasse ChallengeResult**

Mit Hilfe der Hilfsklasse werden die vom externen Provider übertragenen Daten in die Anwendung eingelesen.

#region Helpers

// Used for XSRF protection when adding external logins

private const string XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager

{

get { return HttpContext.GetOwinContext().Authentication; }

}

internal class ChallengeResult : HttpUnauthorizedResult

{

public ChallengeResult(string provider, string redirectUri) : this(provider, redirectUri, null)

{

}

public ChallengeResult(string provider, string redirectUri, string userId)

{

LoginProvider = provider;

RedirectUri = redirectUri;

UserId = userId;

}

public string LoginProvider { get; set; }

public string RedirectUri { get; set; }

public string UserId { get; set; }

public override void ExecuteResult(ControllerContext context)

{

var properties = new AuthenticationProperties { RedirectUri = RedirectUri };

if (UserId != null)

{

properties.Dictionary[XsrfKey] = UserId;

}

context.HttpContext.GetOwinContext().Authentication.Challenge(properties, LoginProvider);

}

}

#endregion

**Gesamter Quellcode des Login Controllers**

using System.Threading.Tasks;

using System.Web;

using System.Web.Mvc;

using Microsoft.Owin.Security;

using Mvc4ExternalLogin.Models;

namespace Mvc4ExternalLogin.Controllers

{

public class LoginController : Controller

{

[AllowAnonymous]

public ActionResult Index(string info = "")

{

return View(new LoginViewModel(info));

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLogin(string provider)

{

return new ChallengeResult(provider, Url.Action("ExternalLoginCallback", "Login"));

}

[AllowAnonymous]

public async Task<ActionResult> ExternalLoginCallback()

{

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync();

if (loginInfo == null)

{

return RedirectToAction("Index", new { info = "No login information found!" });

}

var email = loginInfo.Email;

// Hier gegebenenfalls die Email mit der internen Benutzerverwaltung prüfen

// und anschließend die eigentliche Anwendung aufrufen.

return RedirectToAction("Index", new { info = "[User: " + email + "]" });

}

#region Helpers

// Used for XSRF protection when adding external logins

private const string XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager

{

get { return HttpContext.GetOwinContext().Authentication; }

}

internal class ChallengeResult : HttpUnauthorizedResult

{

public ChallengeResult(string provider, string redirectUri)

: this(provider, redirectUri, null)

{

}

public ChallengeResult(string provider, string redirectUri, string userId)

{

LoginProvider = provider;

RedirectUri = redirectUri;

UserId = userId;

}

public string LoginProvider { get; set; }

public string RedirectUri { get; set; }

public string UserId { get; set; }

public override void ExecuteResult(ControllerContext context)

{

var properties = new AuthenticationProperties { RedirectUri = RedirectUri };

if (UserId != null)

{

properties.Dictionary[XsrfKey] = UserId;

}

context.HttpContext.GetOwinContext().Authentication.Challenge(properties, LoginProvider);

}

}

#endregion

}

}

**GUI**

In diesem Beispiel liest der Dialog die administrierten Anwendungen aus dem Anwendungskontext aus und erstellt für jeden Provider einen Schalter.

@model Mvc4ExternalLogin.Models.LoginViewModel

@using Microsoft.Owin.Security

@{

ViewBag.Title = "Login";

}

<h2>Login</h2>

<fieldset>

<legend>Providers</legend>

<div>

@{

var loginProviders = Context.GetOwinContext().Authentication.GetExternalAuthenticationTypes();

using (Html.BeginForm("ExternalLogin", "Login"))

{

@Html.AntiForgeryToken()

if (loginProviders.Any())

{

foreach (AuthenticationDescription p in loginProviders)

{

<div><button type="submit" id="@p.AuthenticationType" name="provider"

value="@p.AuthenticationType">@p.AuthenticationType</button></div>

}

}

}

}

</div>

@if (!string.IsNullOrWhiteSpace(Model.Info))

{

<div><span>@(new HtmlString(Model.Info))</span></div>

}

</fieldset>

### Startdialog

Beim Generieren der MVC 4 Anwendung wurde im Verzeichnis App\_Start die Klasse RouteConfig.cs angelegt. In dieser Klasse kann der Login Dialog als Startdialog wie folgt festgelegt werden:

using System.Web.Mvc;

using System.Web.Routing;

// ReSharper disable CheckNamespace

namespace Mvc4ExternalLogin

// ReSharper restore CheckNamespace

{

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Login", action = "Index", id = UrlParameter.Optional }

);

}

}

}