EBook Gratis

APRENDIZAJE asp.net-web-api2

Free unaffiliated eBook created from
Stack Overflow contributors.

#asp.net-web-api2

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: asp-net-web-api2

It is an unofficial and free asp.net-web-api2 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official asp.net-web-api2.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con asp.net-web-api2

Observaciones

Esta sección proporciona una descripción general de qué es asp.net-web-api2, y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema importante en asp.net-web-api2, y vincularlo a los temas relacionados. Dado que la Documentación para asp.net-web-api2 es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Examples

Instalación o configuración

Instrucciones detalladas sobre cómo configurar o instalar asp.net-web-api 2.

¿Qué y por qué Asp.Net Web API2?

Qué y por qué ?

Web API2 de Asp.Net es la última versión de Web API. Es una forma fácil de implementar un servicio web RESTful utilizando toda la bondad que proporciona el marco Asp.Net. Una vez que entienda los principios básicos de REST, entonces será muy fácil implementar una API web de Asp.net. Web API2 se basa en el modelo de canalización modular y conectable de Asp.Net. Esto significa que cuando un servidor que aloja una API2 web recibe una solicitud, pasa primero a través de la red de solicitudes Asp.Nets. Esto le permite agregar fácilmente sus propios módulos si encuentra que las capacidades predeterminadas no son suficientes para sus necesidades. Con los anuncios recientes en `ASP.net vNext` esto también significa que puede alojar su Web API2 fuera de Windows Server, lo que abre una amplia gama de casos de uso. Vea aquí para más detalles.

Como funciona

Web API2 utiliza los conceptos de Controlador y Acción de MVC. Los recursos se asignan directamente a los controladores; por lo general, tendrá un controlador diferente para cada una de sus principales entidades de datos (Producto, Persona, Pedido, etc.). Web API2 utiliza el motor de enrutamiento Asp.Net para asignar direcciones URL a los controladores. Normalmente, las API se mantienen dentro de una `/api/` ruta que ayuda a distinguir los controladores de API de otros que no son API en el mismo sitio web.

Las acciones se utilizan para asignar verbos HTTP específicos, por ejemplo, normalmente tendría una acción GET que devuelve todas las entidades. Esta acción respondería a `/api/Products` (donde 'productos' es su controlador) y se vería así:

```
public IEnumerable<string> Get()
{
    return new string[] { "value1", "value2" };
}
```

También puede tener una acción `GET` que acepta una `ID` específica y devuelve una entidad específica. Respondería a `/api/Products/81` y se vería así:

```
public string Get(int id)
 {
    return "value";
 }
```

Hay muchas ventajas ocultas en el uso de la API web que puede que no sepas pero que en realidad te ahorran mucho trabajo.

Web API2 es parte de 'One Asp.Net'

Web API2 es parte de la familia 'One Asp.Net', lo que significa que admite de forma nativa todas las excelentes funciones compartidas que puede usar actualmente con MVC o formularios web, esto incluye (estos son solo algunos ejemplos):

- Marco de la entidad
- Autorización e identidad
- Andamio
- Enrutamiento

Serialización y encuadernación de modelos

Web API2 está configurado de forma predeterminada para proporcionar respuestas en XML o JSON (JSON es el predeterminado). Sin embargo, como desarrollador, no necesita realizar ninguna conversión o análisis: simplemente devuelve un objeto fuertemente tipado y Web API2 lo convertirá a XML o JSON y lo devolverá al cliente que realiza la llamada, este es un proceso denominado Negociación de contenido. Este es un ejemplo de una acción `GET` que devuelve un objeto Producto fuertemente tipado.

```
public Product GetProduct(int id)
{
   var product = _products.FirstOrDefault(p => p.ID == id);
   if (product == null)
   {
      throw new HttpResponseException(HttpStatusCode.NotFound);
   }
   return Request.CreateResponse(HttpStatusCode.OK, product);
}
```

Esto también funciona para solicitudes entrantes utilizando una función llamada Validación de modelo. Con la Validación del modelo, Web API2 puede validar y analizar los datos del cuerpo de respuesta entrante a un objeto fuertemente tipado para que pueda trabajar con su código. Este es un ejemplo de enlace de modelo:

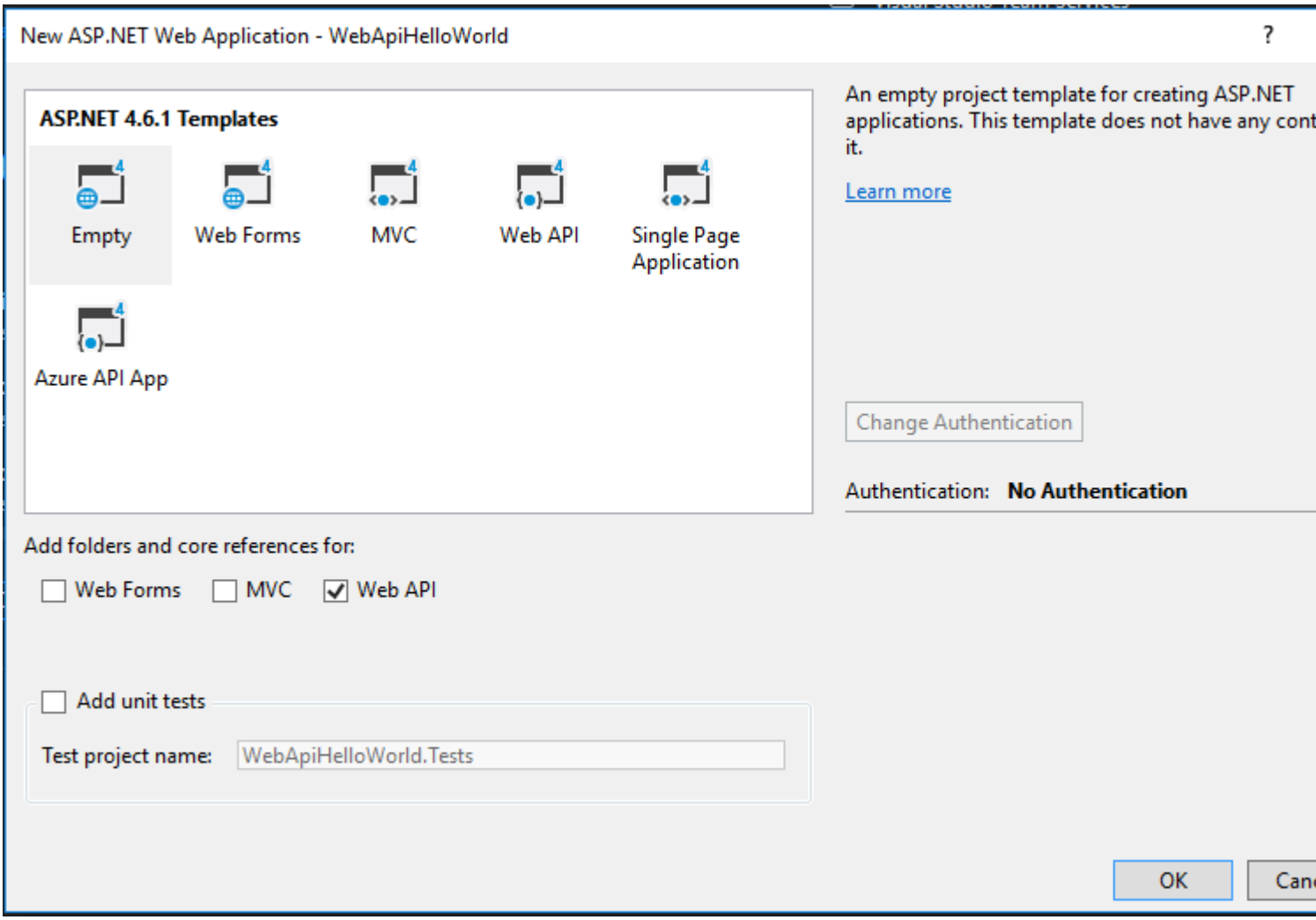
```
public HttpResponseMessage Post(Product product)
{
    if (ModelState.IsValid)
    {
        // Do something with the product (not shown).

        return new HttpResponseMessage(HttpStatusCode.OK);
    }
    else
    {
        return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ModelState);
    }
}
```

Hola Web Api

Web Api 2 - ejemplo de Hello World

Vamos a crear una nueva aplicación simple de Web Api que nos devolverá a Json con mensaje y nombre de usuario.
¡Empecemos! Primero cree un nuevo proyecto de Web Api usando Visual Studio y seleccione Plantilla vacía. Asegúrese de revisar la carpeta "Web Api":



NOTA No elegí la plantilla "Web Api" porque agrega una referencia a ASP.NET MVC para

proporcionar la página de ayuda de la API. Y en una aplicación tan básica no la necesitamos realmente.

Añadiendo un modelo

Un modelo es una clase de C # que representa algunos datos en nuestra aplicación. ASP.NET Web API puede serializar automáticamente el modelo a JSON, XML o algún otro formato (depende de la configuración).

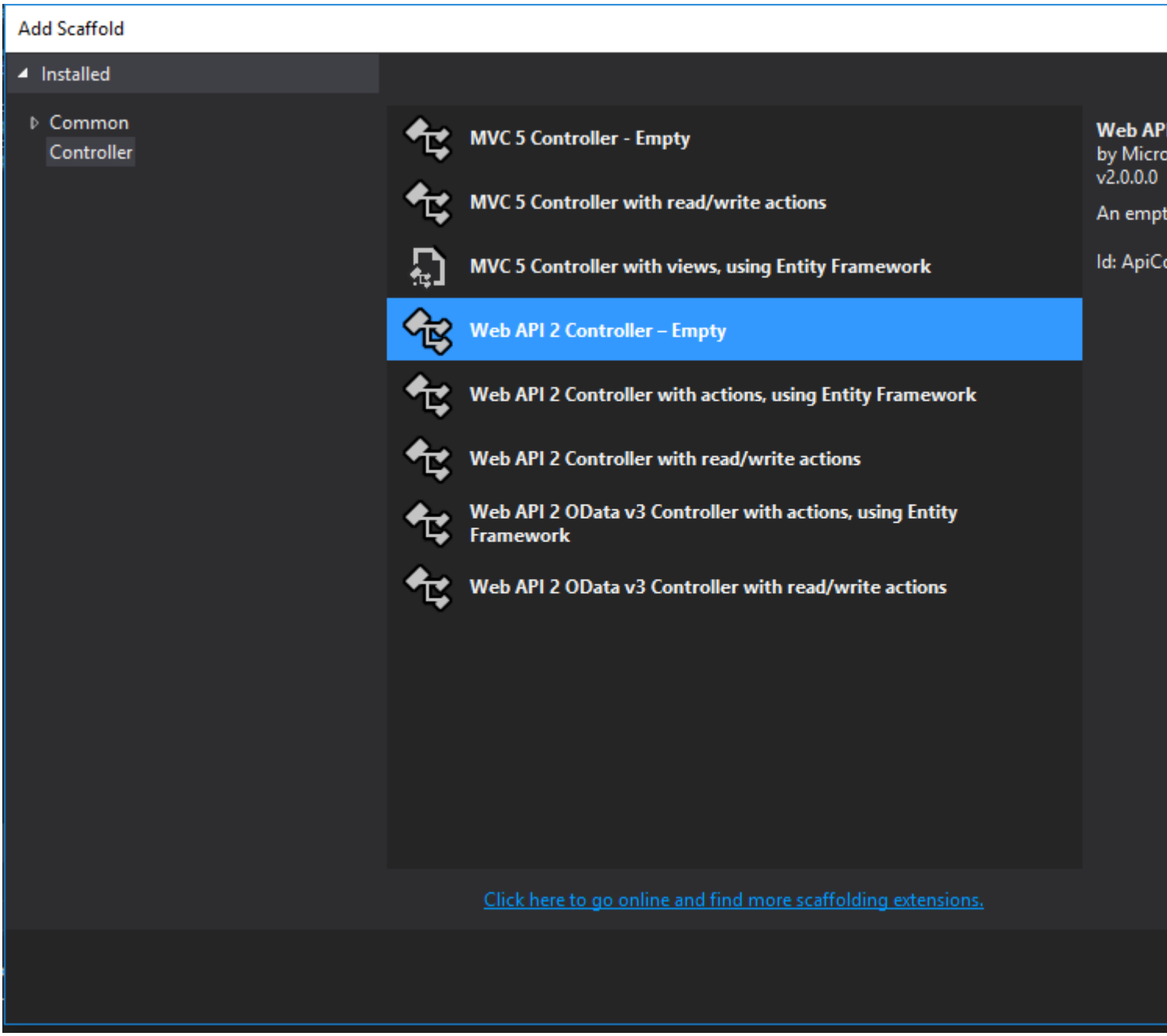
En nuestra aplicación, crearemos un solo modelo, pero las aplicaciones del mundo real generalmente tienen muchos de ellos.

En el Explorador de soluciones, haga clic con el botón derecho en la **carpeta Modelos** . A continuación, seleccione **Agregar** y luego seleccione **Clase** . Nombra la clase "HelloMessage". Nuestro modelo necesita dos propiedades: *MessageText* y *UserName* :

```
namespace WebApiHelloWorld.Models
{
    public class HelloMessage
    {
        public string MessageText { get; set; }
        public string UserName { get; set; }
    }
}
```

Añadiendo un controlador

Un controlador maneja las solicitudes HTTP. Nuestra aplicación solo necesita un controlador que devuelva Json con el mensaje de saludo y el nombre de usuario (que pasaremos en la URL).
En el Explorador de soluciones, haga clic con el botón derecho en la **carpeta Controladores** . A continuación, seleccione **Agregar** y luego seleccione **Controlador** . En la ventana abierta, seleccione **Web API Controller - Vacío** y haga clic en **Agregar** .



Establecer el nombre del controlador como "HelloController". Siguiente código de edición del controlador creado. Necesitamos agregar el método que devuelve el mensaje de saludo.

```
using System.Web.Http;
using WebApiHelloWorld.Models;

namespace WebApiHelloWorld.Controllers
{
    public class HelloController : ApiController
    {
        public HelloMessage GetMessage(string name)
        {
            HelloMessage message = new HelloMessage
            {
                MessageText = "Hello my Dear!",
                UserName = name
            };
```

```
            return message;
        }
    }
}
```

NOTA Asegúrese de agregar el `using WebApiHelloWorld.Models` . Sin él, su controlador no encontrará la clase HelloMessage.

Terminar

¡Eso es todo! Ahora solo necesitas compilar e iniciar tu aplicación. Simplemente presione **Ctrl + F5** o simplemente **F5** (para comenzar sin depurar). Visual studio lanzará navegador web. Necesitas llamar a tu controlador. Para hacerlo, agregue al final de la URL "/ api / hello? Name = John". El resultado debe ser:

```
{
    "MessageText": "Hello my Dear!",
    "UserName": "John"
}
```

Lea Empezando con asp.net-web-api2 en línea: https://riptutorial.com/es/asp-net-web-api2/topic/3323/empezando-con-asp-net-web-api2

Capítulo 2: Atributo de enrutamiento en ASP.NET Web API 2

Introducción

Como su nombre indica, esto utiliza atributos para enrutar. Esto le da al usuario más control sobre los URI en la WebAPI. Por ejemplo, puede describir las jerarquías del recurso. Sin embargo, el anterior 'Enrutamiento convencional' es totalmente compatible. Los usuarios pueden tener una mezcla de ambos también.

Sintaxis

- [RoutePrefix ("api / books")] - para la clase de controlador
- [Ruta ("getById")] - para acciones
- [Ruta ("~ / api / autores / {autorId: int} / libros")] - para anular prefijo de ruta

Parámetros

Nombre del parámetro	Detalles
RoutePrefix	atribuir a la clase de controlador. todos los prefijos de URL comunes en las acciones son aplastados aquí. toma la cadena como entrada
Ruta	atribuir a las acciones del controlador. Cada acción tendrá una ruta asociada (no necesariamente).
Ruta ("~ / api /")	esto anula el prefijo de ruta

Observaciones

Actualmente, las rutas de atributos no tienen controladores de mensajes específicos del controlador. Como no hay manera de especificar qué controlador ejecutar para qué ruta en el momento de la declaración. Esto es posible en enrutamiento convencional.

Examples

Enrutamiento de atributos básicos

Simplemente agregue un atributo a la acción del controlador

```
[Route("product/{productId}/customer")]
```

```
public IQueryable<Product> GetProductsByCustomer(int productId)
{
    //action code goes here
}
```

esto se consultará como / product / 1 / customer y productId = 1 se enviará a la acción del controlador.

Asegúrese de que el que está dentro de '{}' y el parámetro de acción sean los mismos. productId en este caso.

antes de usar esto, debe especificar que está utilizando el enrutamiento de atributos por:

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        config.MapHttpAttributeRoutes();
    }
}
```

Prefijos de ruta

Generalmente, las rutas en un controlador tienen el mismo prefijo conectado de alguna manera con la funcionalidad de este controlador. Por ejemplo:

```
public class ProductsController : ApiController
{
    [Route("api/products")]
    public IEnumerable<Product> GetProducts() { ... }

    [Route("api/products/{id:int}")]
    public Product GetProduct(int id) { ... }

    [Route("api/products")]
    [HttpPost]
    public HttpResponseMessage CreateProduct(Product product) { ... }
}
```

En tal escenario podemos establecer un prefijo común para todo el controlador. Para ello usamos el atributo `[RoutePrefix]` :

```
[RoutePrefix("api/products")]
public class ProductsController : ApiController
{
    // GET api/products
    [Route("")]
    public IEnumerable<Product> GetProducts() { ... }

    // GET api/products/5
    [Route("{id:int}")]
    public Product GetProduct(int id) { ... }

    //POST api/products
```

```
    [Route("")]
    [HttpPost]
    public HttpResponseMessage CreateProduct(Product product) { ... }
}
```

Anular prefijo de ruta

Si queremos reemplazar el prefijo de ruta, podemos usar una tilde **(~)** en el atributo de enrutamiento del método:

```
[RoutePrefix("api/products")]
public class ProductsController : ApiController
{
    // GET api/owners/products
    [Route("~/api/owners/products")]
    public IEnumerable<Product> GetProducts() { ... }

    //...
}
```

Lea Atributo de enrutamiento en ASP.NET Web API 2 en línea: https://riptutorial.com/es/asp-net-web-api2/topic/9559/atributo-de-enrutamiento-en-asp-net-web-api-2

Capítulo 3: OAuth 2.0 en ASP.NET Web API

Observaciones

Registrarse en una aplicación de Android

Estos son los pasos que he tomado para iniciar sesión / registrarme usando una aplicación de Android:

- Tenga una actividad de inicio de sesión que consulte la ruta de ExternalLogins y obtenga los proveedores disponibles. Esta actividad debe tener el indicador NoHistory habilitado y ejecutarse como una sola instancia.
- Al presionar el botón de un usuario, inicie una pestaña personalizada de Chrome con la URL del proveedor. El usuario debe iniciar sesión y redirigirse a su sitio publicado en la URL de retorno dada. No utilice un WebView.
- Haga que esta página redirija al usuario nuevamente, utilizando un esquema de URI personalizado para iniciar una actividad posterior al inicio de sesión dentro de su aplicación con el token de acceso, la fecha de caducidad y los detalles de la cuenta de usuario agregados como datos adicionales. Esto deberá hacerse en JavaScript en la página web, ya que los controladores del servidor no pueden acceder a los parámetros de la URL.
- Almacene los datos y el token del usuario en una base de datos MySQL local. En cada inicio de sesión, verifique si el token todavía está en la fecha.
- Ahora se puede autorizar cualquier llamada a la API utilizando el encabezado HTTP de Autorización, con su token almacenado agregado como tal: "Portador {token}"

Examples

Configurando un proveedor de OAuth

Necesita obtener algunos detalles de su proveedor OAuth de elección. Buscaremos en Google, pero ASP.NET también está configurado para permitir el uso directo de Twitter, Facebook y Microsoft (obviamente).

Querrá ir a la consola de desarrolladores de Google (https://console.developers.google.com/) y crear un proyecto, habilitar la API de Google+ (para obtener la información del perfil del usuario, como su nombre y avatar) y cree un nuevo ID de cliente de OAuth 2 en la sección "Credenciales". Los orígenes de JavaScript autorizados deben ser la URL raíz de su proyecto (por ejemplo, https://yourapi.azurewebsites.net) y los URI de redireccionamiento deben incluir el punto final de devolución de llamada de Google incorporado de ASP (https://yourapi.azurewebsites.net/signin-google) así como la ruta de su elección de devolución de llamada (https://yourapi.azurewebsites.net/callback) . Obtener estos errores resultará en que Google tenga un ataque sibilante.

De vuelta en su proyecto de Visual Studio, abra App_Start> Startup.Auth.cs. Reemplace la sección de Google comentada en la parte inferior con el código a continuación, agregando el ID y

el Secreto desde la Consola de Desarrolladores de Google:

```
var googleAuthOptions = new GoogleOAuth2AuthenticationOptions()
{
      ClientId = "YOUR ID",
      ClientSecret = "YOUR SECRET",
            Provider = new GoogleOAuth2AuthenticationProvider()
            {
                  OnAuthenticated = (context) =>
                  {
                        context.Identity.AddClaim(new Claim("urn:google:name",
context.Identity.FindFirstValue(ClaimTypes.Name)));
                        context.Identity.AddClaim(new Claim("urn:google:email",
context.Identity.FindFirstValue(ClaimTypes.Email)));
                        //This following line is need to retrieve the profile image
                        context.Identity.AddClaim(new Claim("urn:google:accesstoken",
context.AccessToken, ClaimValueTypes.String, "Google"));
                        return System.Threading.Tasks.Task.FromResult(0);
                  }
            }
      };
app.UseGoogleAuthentication(googleAuthOptions);
```

Estas reclamaciones adicionales le permiten consultar a Google sobre la información del perfil del usuario, como su nombre y la URL de su avatar.

Almacenamiento de perfiles de usuario de OAuth

Cuando alguien se registra en su aplicación, un nuevo objeto ApplicationUser se almacenará en la base de datos. De forma predeterminada, la clase es muy simple, pero se puede personalizar; puede encontrarla en Modelos> IdentityModels.cs. Esto es mío:

```
public class ApplicationUser : IdentityUser
{
      public string ImageUrl { get; set; }
      public DateTime DateCreated { get; set; }
      public string FirstName { get; set; }
      public string AuthProvider { get; set; }
      public string Surname { get; set; }

      public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser>
manager, string authenticationType)
      {
            // Note the authenticationType must match the one defined in
CookieAuthenticationOptions.AuthenticationType
            var userIdentity = await manager.CreateIdentityAsync(this, authenticationType);
            // Add custom user claims here
            return userIdentity;
      }
}
```

Para referencia, el perfil de usuario devuelto por Google con la API de Google+ habilitada toma la siguiente estructura JSON:

```
{{
```

```
    "id": "1****************6",
    email": "dan********@gmail.com",
    "verified_email": true,
    "name": "Dan Richardson",
    "given_name": "Dan",
    "family_name": "Richardson",
    "link": "https://plus.google.com/+DanRichardson",
    "picture": "https://lh4.googleusercontent.com/photo.jpg",
    "gender": "male",
    "locale": "en"
}}
```

Permitir redireccionar URL que no sean raíz del sitio

Vaya a Proveedores> ApplicationOAuthProvider.cs y edite la función ValidateClientRedirectUri. Esto fue muy importante para mí, ya que si no lo haces, aparecerá un mensaje de error increíblemente inútil. De manera predeterminada, este código hará que cualquier devolución de llamada a su sitio no sea válida a menos que estén en la raíz del sitio. Es probable que desee poder manejar las devoluciones de llamada en un controlador, por lo que deberá cambiarlo a algo como esto:

```
public override Task ValidateClientRedirectUri(OAuthValidateClientRedirectUriContext context)
{
      if (context.ClientId == _publicClientId)
      {
            Uri expectedRootUri = new Uri(context.Request.Uri, "/");
            Uri expectedCallbackUri = new Uri(context.Request.Uri, "/callback");

            if (expectedRootUri.AbsoluteUri == context.RedirectUri ||
                  expectedCallbackUri.AbsoluteUri == context.RedirectUri)
            {
                  context.Validated();
            }
      }
      return Task.FromResult<object>(null);
}
```

Flujo de registro

Aquí está el flujo predeterminado de registrar un usuario en la API web. Todas estas rutas se pueden encontrar en el AccountController:

- El usuario solicita una lista de los proveedores de inicio de sesión que utilizan la ruta GetExternalLogins, pasando una URL de retorno como parámetro. Esto devuelve una matriz de objetos del proveedor, que contiene el nombre del proveedor y la ruta que se debe solicitar para iniciar sesión con él (cada uno configurado para usar la url de retorno dada).

  por ejemplo, GET: / api / Account / ExternalLogins? returnUrl = / callback & generateState = true, donde la URL de retorno solicitada es / callback

- El usuario llama a una de estas URL devueltas en un navegador, donde son redirigidas a la página de inicio de sesión del proveedor. Una vez que ha iniciado sesión, el proveedor devuelve una cookie a ASP, que maneja la creación de una cuenta de usuario externa.

- El usuario será redirigido a la URL de retorno que pasó en el primer paso. Se pasa un token de acceso externo al usuario, que se adjunta a la URL en un # param. Este token solo se puede utilizar en rutas seleccionadas, como RegisterExternal.
- El usuario ahora envía una solicitud POST a RegisterExternal, utilizando el nuevo token de acceso como clave de portador. Luego, ASP crea un nuevo ApplicationUser y devuelve un token de acceso adecuado que puede usarse en cualquier ruta.

Almacenamiento de información de perfil de OAuth

Descubrí que la plantilla de la API web está rota: la implementación predeterminada se basa en las cookies en el último paso, que probablemente no desee utilizar en una API Rest. Sin una cookie, GetExternalLoginInfoAsync en RegisterExternal siempre devuelve null. Quité el RegisterExternal por completo, en lugar de eso, creé la cuenta de usuario final en GetExternalLogin, llamada a mi regreso del proveedor de OAuth (en este caso, Google):

```
[OverrideAuthentication]
[HostAuthentication(DefaultAuthenticationTypes.ExternalCookie)]
[AllowAnonymous]
[Route("ExternalLogin", Name = "ExternalLogin")]
public async Task<IHttpActionResult> GetExternalLogin(string provider, string error = null)
{
      if (error != null)
      {
            return Redirect(Url.Content("~/") + "#error=" + Uri.EscapeDataString(error));
      }

      if (!User.Identity.IsAuthenticated)
      {
            return new ChallengeResult(provider, this);
      }

      ExternalLoginData externalLogin = ExternalLoginData.FromIdentity(User.Identity as
ClaimsIdentity);

      if (externalLogin == null)
      {
            return InternalServerError();
      }

      if (externalLogin.LoginProvider != provider)
      {
            Authentication.SignOut(DefaultAuthenticationTypes.ExternalCookie);
            return new ChallengeResult(provider, this);
      }

      ApplicationUser user = await UserManager.FindAsync(new
UserLoginInfo(externalLogin.LoginProvider,
                externalLogin.ProviderKey));

      bool hasRegistered = user != null;

      if (hasRegistered)
      {
            Authentication.SignOut(DefaultAuthenticationTypes.ExternalCookie);
```

```
            ClaimsIdentity oAuthIdentity = await user.GenerateUserIdentityAsync(UserManager,
                    OAuthDefaults.AuthenticationType);
            ClaimsIdentity cookieIdentity = await user.GenerateUserIdentityAsync(UserManager,
            CookieAuthenticationDefaults.AuthenticationType);

            AuthenticationProperties properties =
ApplicationOAuthProvider.CreateProperties(user.UserName);
                Authentication.SignIn(properties, oAuthIdentity, cookieIdentity);
      }
      else
      {
            var accessToken = Authentication.User.Claims.Where(c =>
c.Type.Equals("urn:google:accesstoken")).Select(c => c.Value).FirstOrDefault();
            Uri apiRequestUri = new
Uri("https://www.googleapis.com/oauth2/v2/userinfo?access_token=" + accessToken);

            RegisterExternalBindingModel model = new RegisterExternalBindingModel();

            using (var webClient = new System.Net.WebClient())
            {
                  var json = webClient.DownloadString(apiRequestUri);
                  dynamic jsonResult = JsonConvert.DeserializeObject(json);
                  model.Email = jsonResult.email;
                  model.Picture = jsonResult.picture;
                  model.Family_name = jsonResult.family_name;
                  model.Given_name = jsonResult.given_name;
            }

            var appUser = new ApplicationUser
            {
                  UserName = model.Email,
                  Email = model.Email,
                  ImageUrl = model.Picture,
                  FirstName = model.Given_name,
                  Surname = model.Family_name,
                  DateCreated = DateTime.Now
            };

            var loginInfo = await Authentication.GetExternalLoginInfoAsync();

            IdentityResult result = await UserManager.CreateAsync(appUser);
            if (!result.Succeeded)
            {
                  return GetErrorResult(result);
            }

            result = await UserManager.AddLoginAsync(appUser.Id, loginInfo.Login);
            if (!result.Succeeded)
            {
                  return GetErrorResult(result);
            }

            ClaimsIdentity oAuthIdentity = await
appUser.GenerateUserIdentityAsync(UserManager,
                   OAuthDefaults.AuthenticationType);
            ClaimsIdentity cookieIdentity = await
appUser.GenerateUserIdentityAsync(UserManager,
                    CookieAuthenticationDefaults.AuthenticationType);

            AuthenticationProperties properties =
ApplicationOAuthProvider.CreateProperties(appUser.UserName);
```

```
            Authentication.SignIn(properties, oAuthIdentity, cookieIdentity);
        }

        return Ok();
    }
```

Esto elimina la necesidad de que la aplicación cliente realice una solicitud POST innecesaria después de recibir un token de acceso externo.

Lea OAuth 2.0 en ASP.NET Web API en línea: https://riptutorial.com/es/asp-net-web-api2/topic/7603/oauth-2-0-en-asp-net-web-api

Creditos

S. No	Capítulos	Contributors
1	Empezando con asp.net-web-api2	Archil Labadze, Community, Krzyserious, Mostafiz
2	Atributo de enrutamiento en ASP.NET Web API 2	Ajay Aradhya, Krzyserious
3	OAuth 2.0 en ASP.NET Web API	GS_Dan