ComponentSpace

ComponentSpace

SAML v2.0

Examples Guide

www.componentspace.com

Contents

Introduction

This document describes the example projects shipped with the product.

Refer to the SAML v2.0 Installation Guide for instructions on installing the product.

The example projects include SAML configurations. Refer to the SAML v2.0 Configuration Guide for information on SAML configuration.

The SAML v2.0 Developer Guide describes the SAML APIs called by the example projects.

Visual Studio Solution Files

Solution files for the various supported versions of Visual Studio are included in the installation folder.

Select the appropriate solution file to open the example projects in Visual Studio.

Visual Studio 2017

The installation folder includes an ExamplesVS2017.sln file.

No changes are required for the example projects to build cleanly and run without error in Visual Studio 2017.

Setting the Startup Projects

Open the Visual Studio solution properties to edit the start-up project to ensure the required projects are run.

For example, start the ExampleIdentityProvider, MiddlewareServiceProvider and ExampleServiceProvider projects for SSO between these applications.

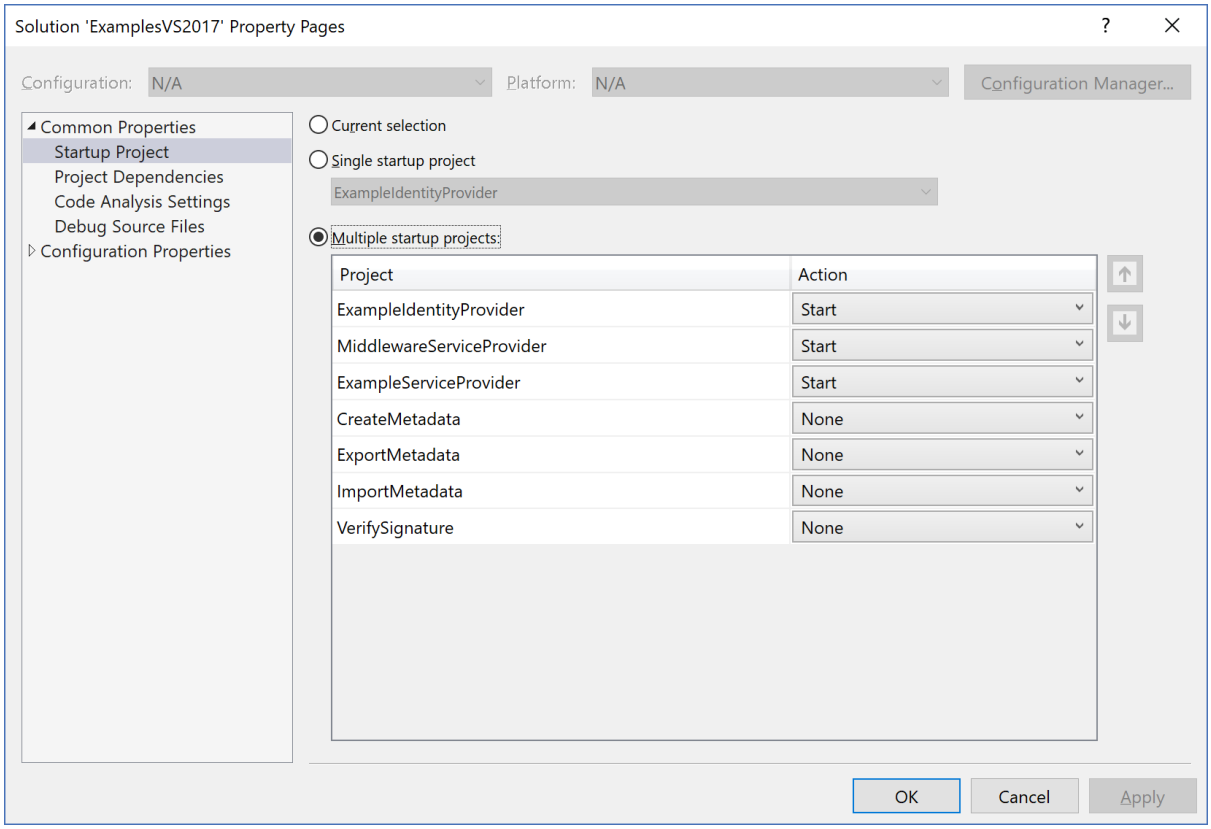


Figure 1 Startup Projects

Example Identity Provider

The ExampleIdentityProvider project is an ASP.NET Core web application based off the Visual Studio template.

It demonstrates acting as a SAML identity provider and supports:

- IdP-initiated SSO
- SP-initiated SSO
- IdP-initiated SLO
- SP-initiated SLO

Through changes to its SAML configuration, it can support all SAML v2.0 compliant third-party offerings.

Building and Running

The ExampleIdentityProvider should build without any errors or warnings.

As it is configured to use the default LocalDB connection string, the simplest approach is to run the application on IIS Express through the Visual Studio debugger.

Note that this database is not used by the SAML API but is the application's user registry.

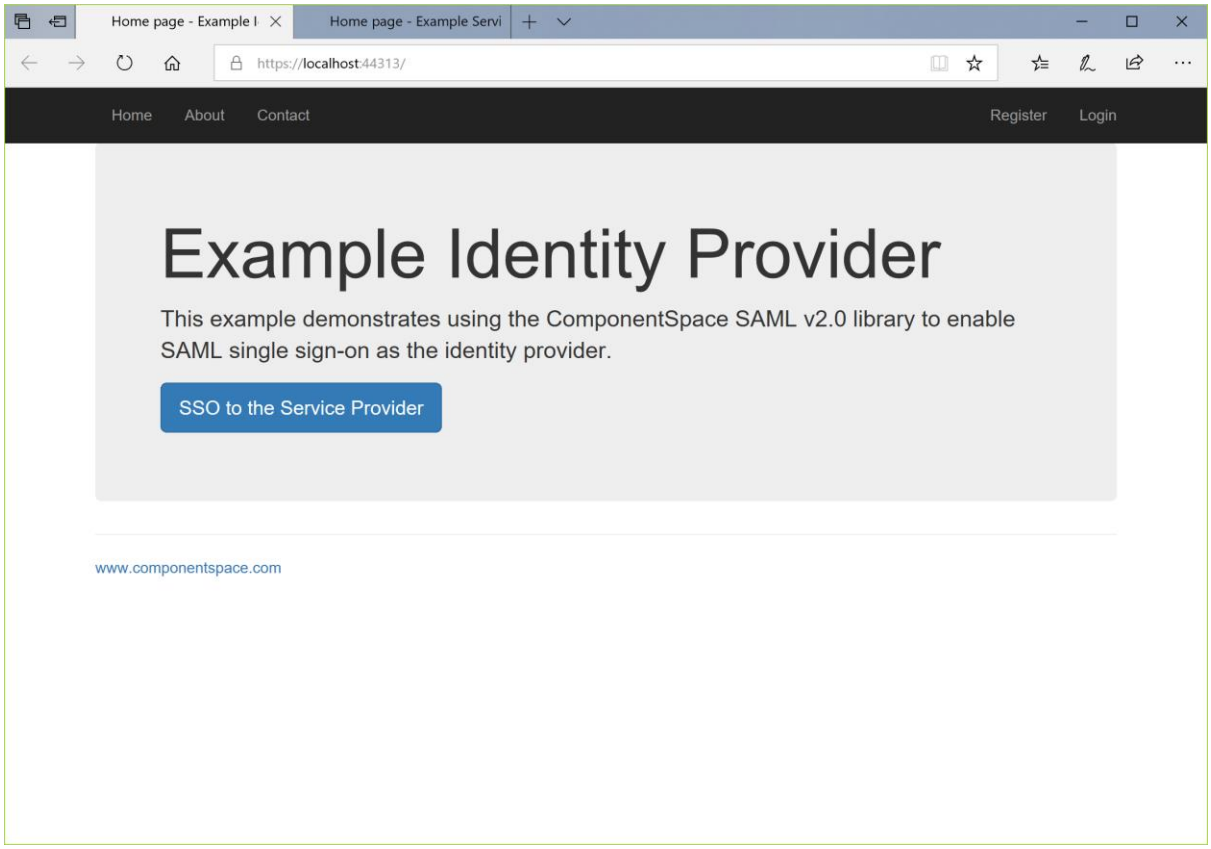
To run on IIS, the application must be configured in and published to IIS. It should use a database provider other than LocalDB.

The application is configured to run at https://localhost:44313/.

If this is changed, the corresponding ExampleServiceProvider's and MiddlewareServiceProvider's SAML configuration must be updated to match the new URLs.

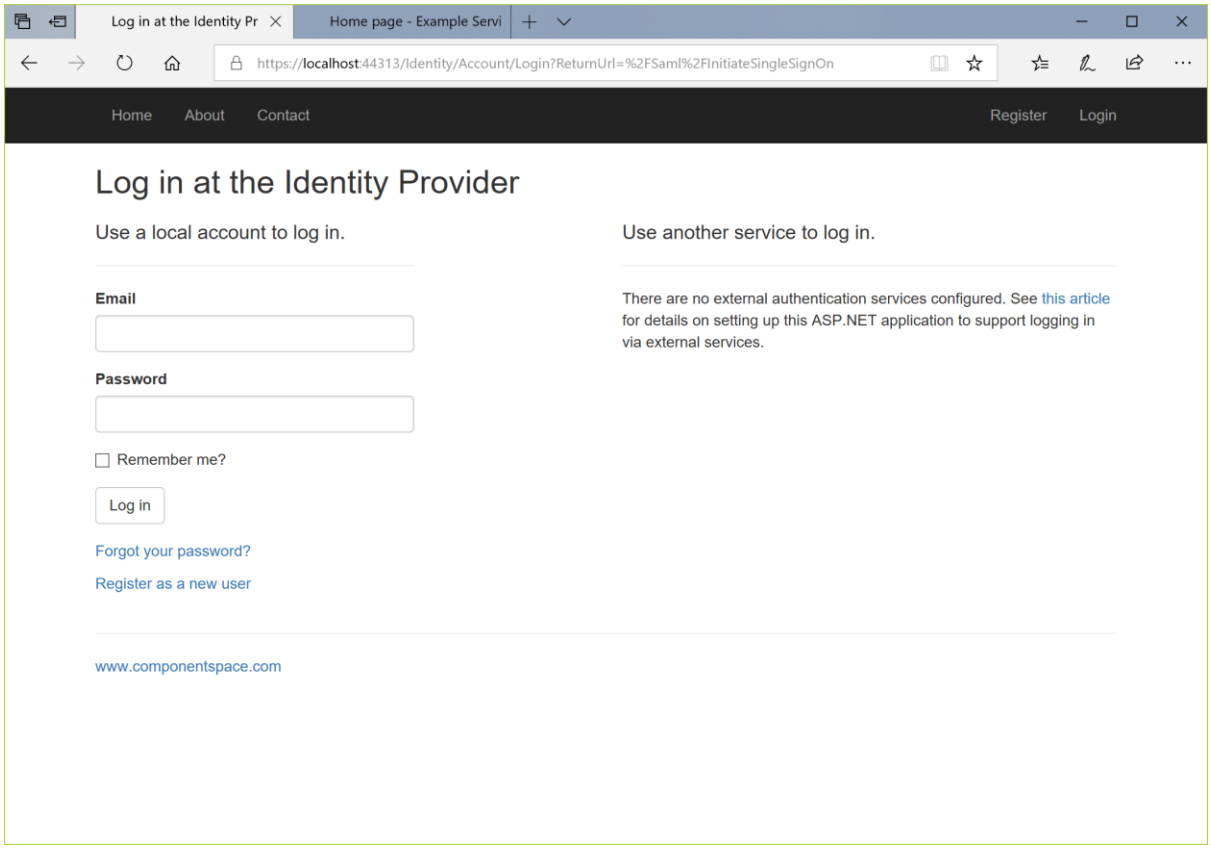
IdP-initiated SSO

Browse to the example identity provider's home page at http://localhost:44313/.



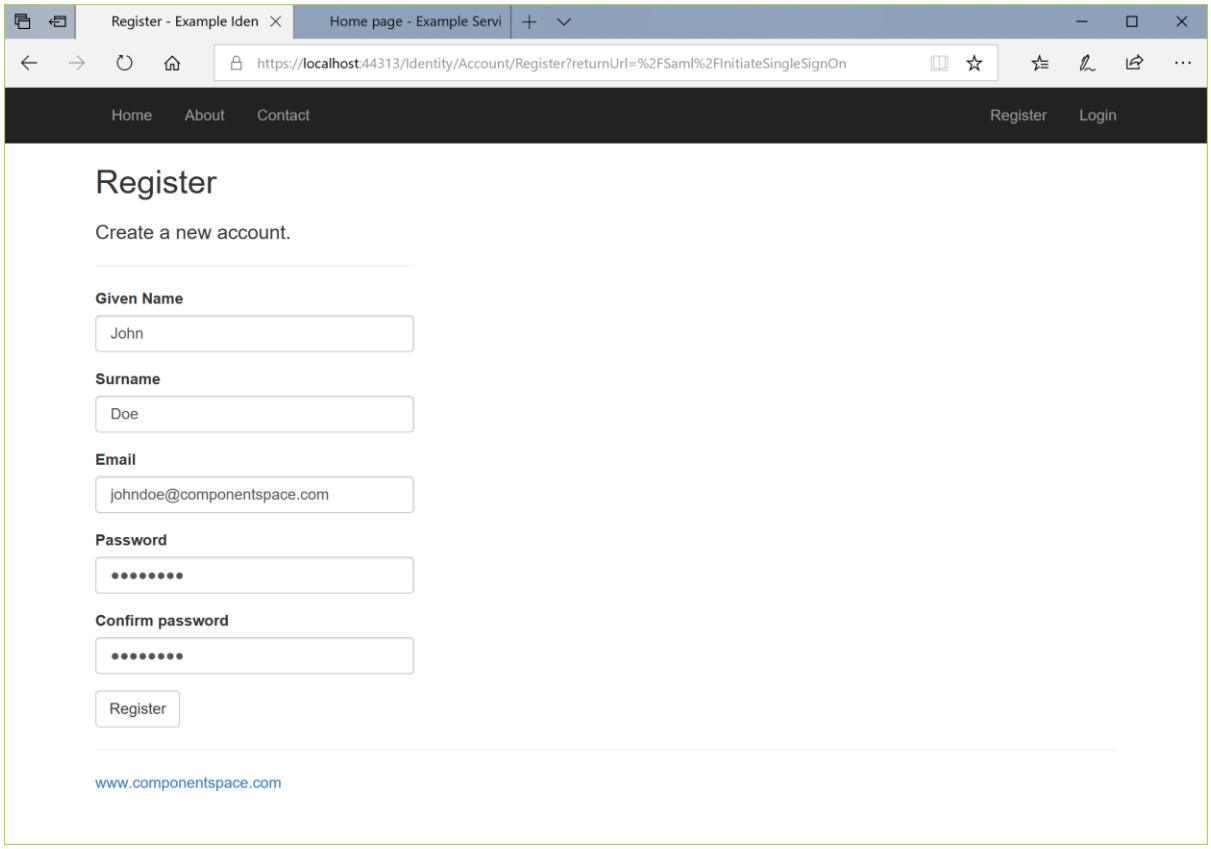
Click the SSO to the Service Provider button.

As you haven't been authenticated at the ExampleIdentityProvider, you are prompted to login or register.

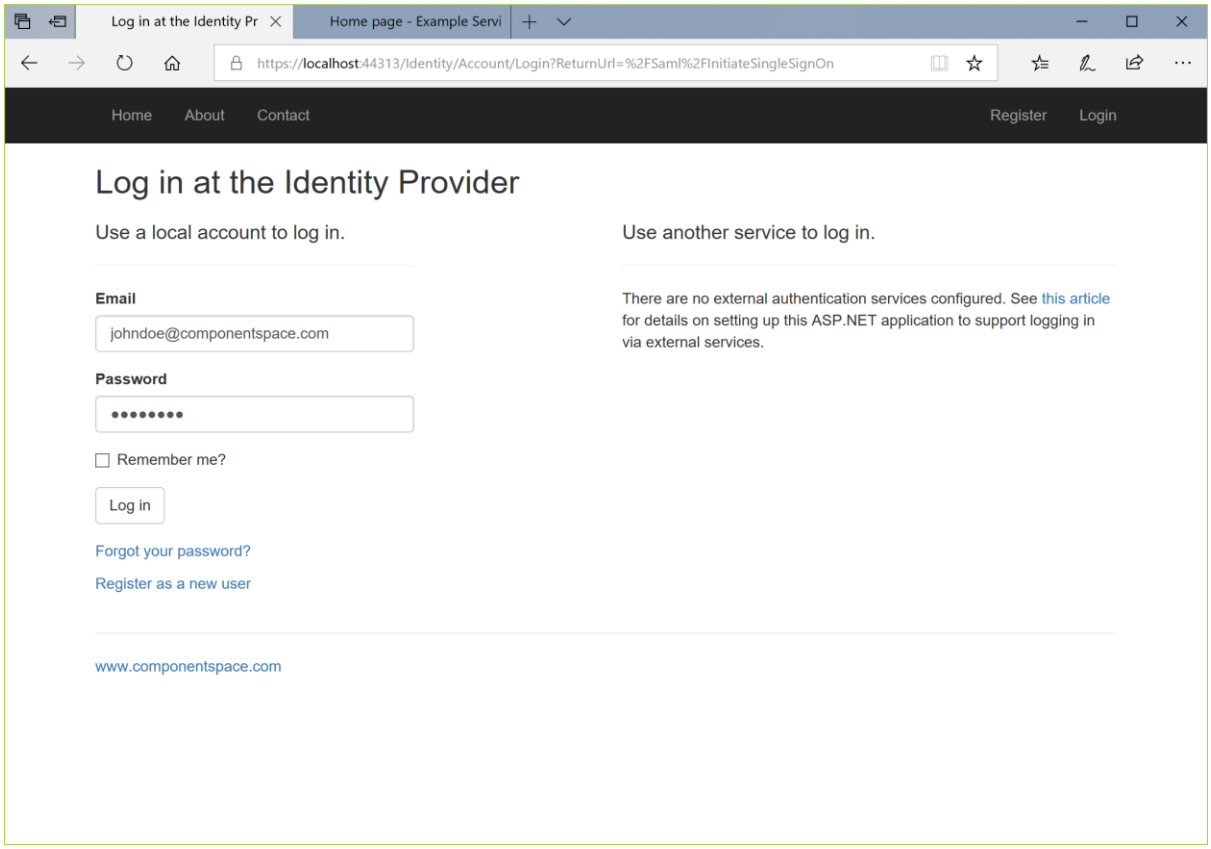


Click the link to register as a new user.

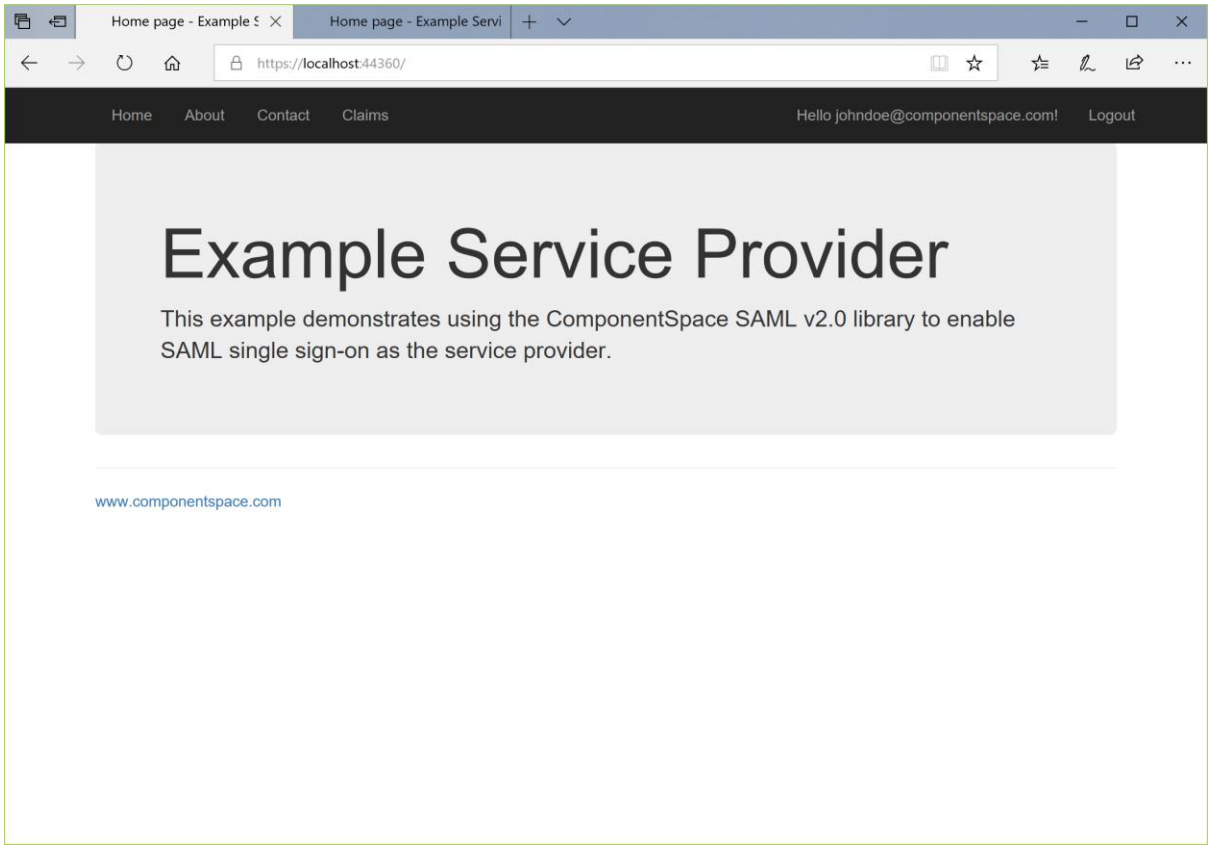
Complete the registration process.



If you've previously registered, simply login.

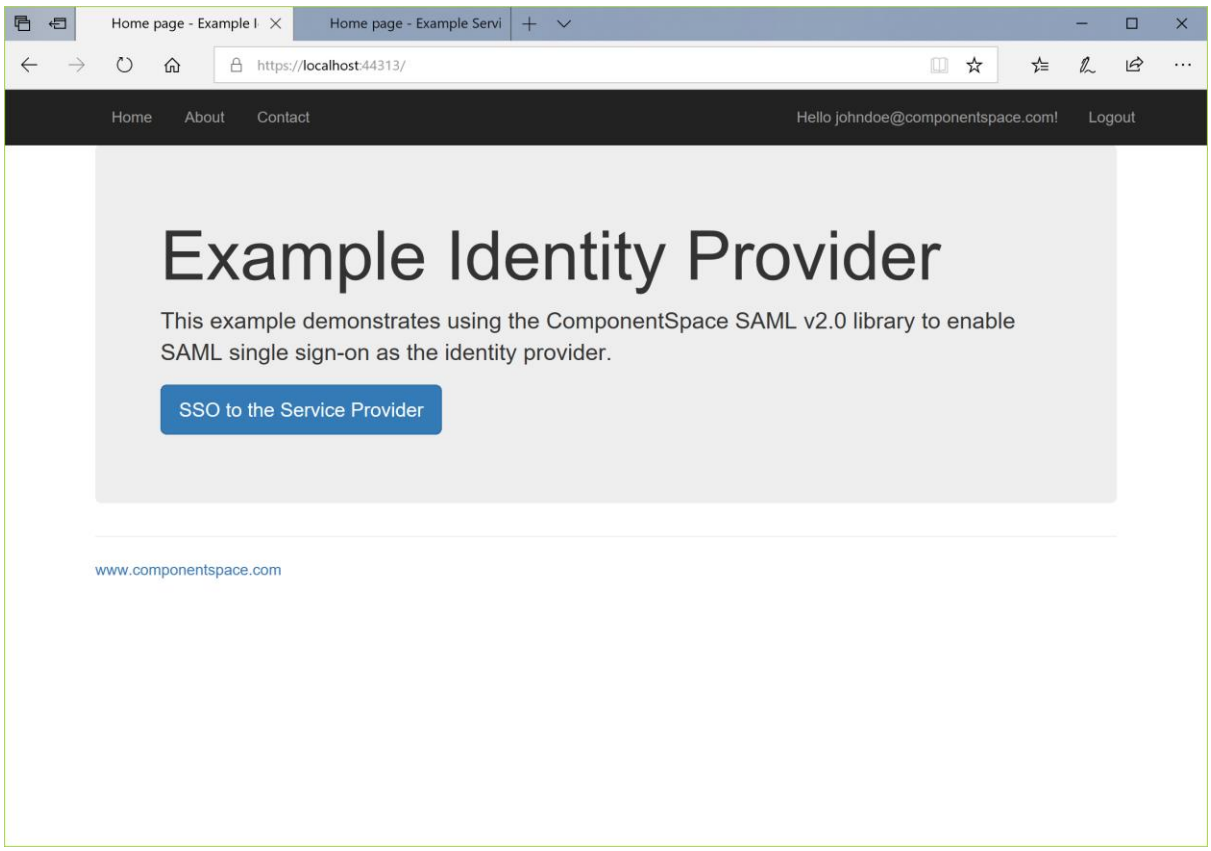


SSO completes with automatic login at the service provider. The user identity is that specified by the identity provider.

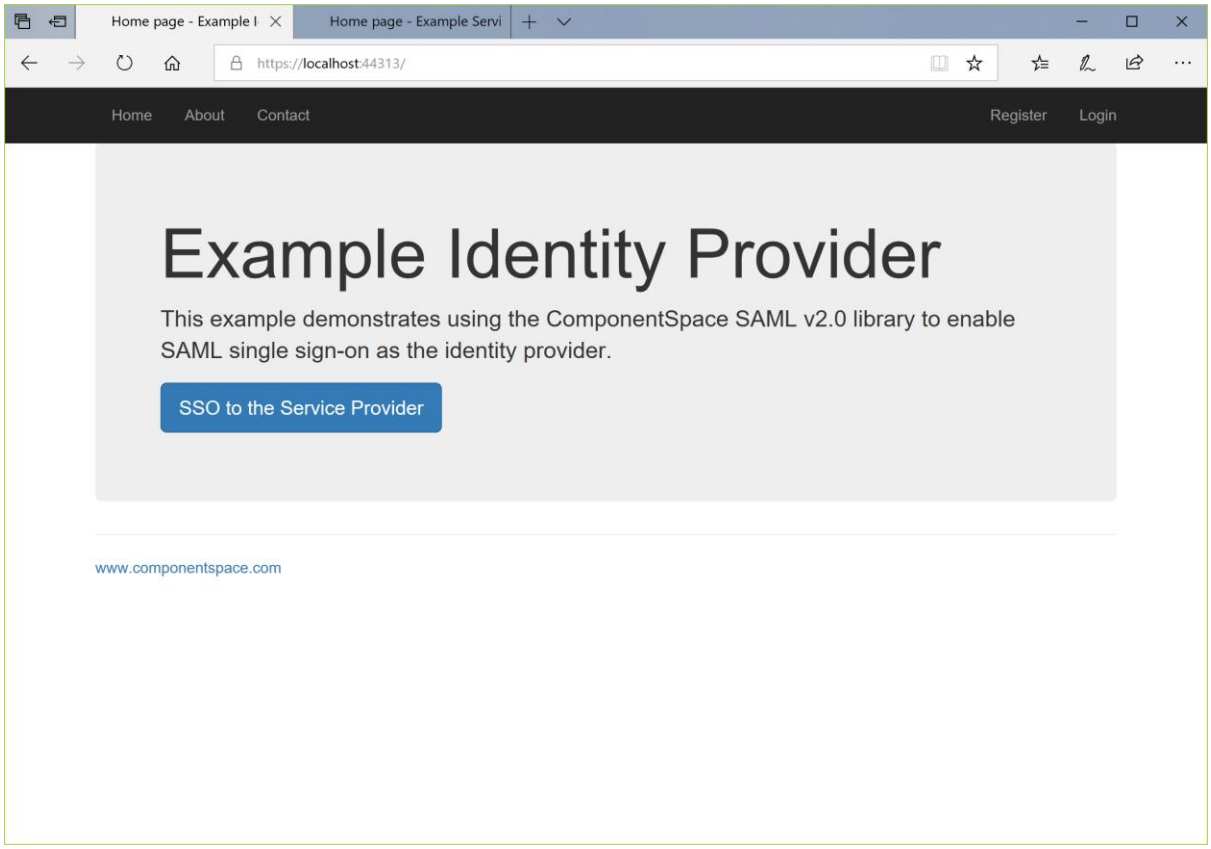


IdP-initiated SLO

Having completed SSO, in the same browser window, browse to the example identity provider's home page at https://localhost:44313/.



Click the Log out link. Logout occurs at both the identity provider and service provider.



Example Service Provider

The ExampleServiceProvider project is an ASP.NET Core web application based off the Visual Studio template.

It demonstrates acting as a SAML service provider and supports:

- IdP-initiated SSO
- SP-initiated SSO
- IdP-initiated SLO
- SP-initiated SLO

Through changes to its SAML configuration, it can support all SAML v2.0 compliant third-party offerings.

Building and Running

The ExampleServiceProvider should build without any errors or warnings.

As it is configured to use the default LocalDB connection string, the simplest approach is to run the application on IIS Express through the Visual Studio debugger.

Note that this database is not used by the SAML API but is the application's user registry.

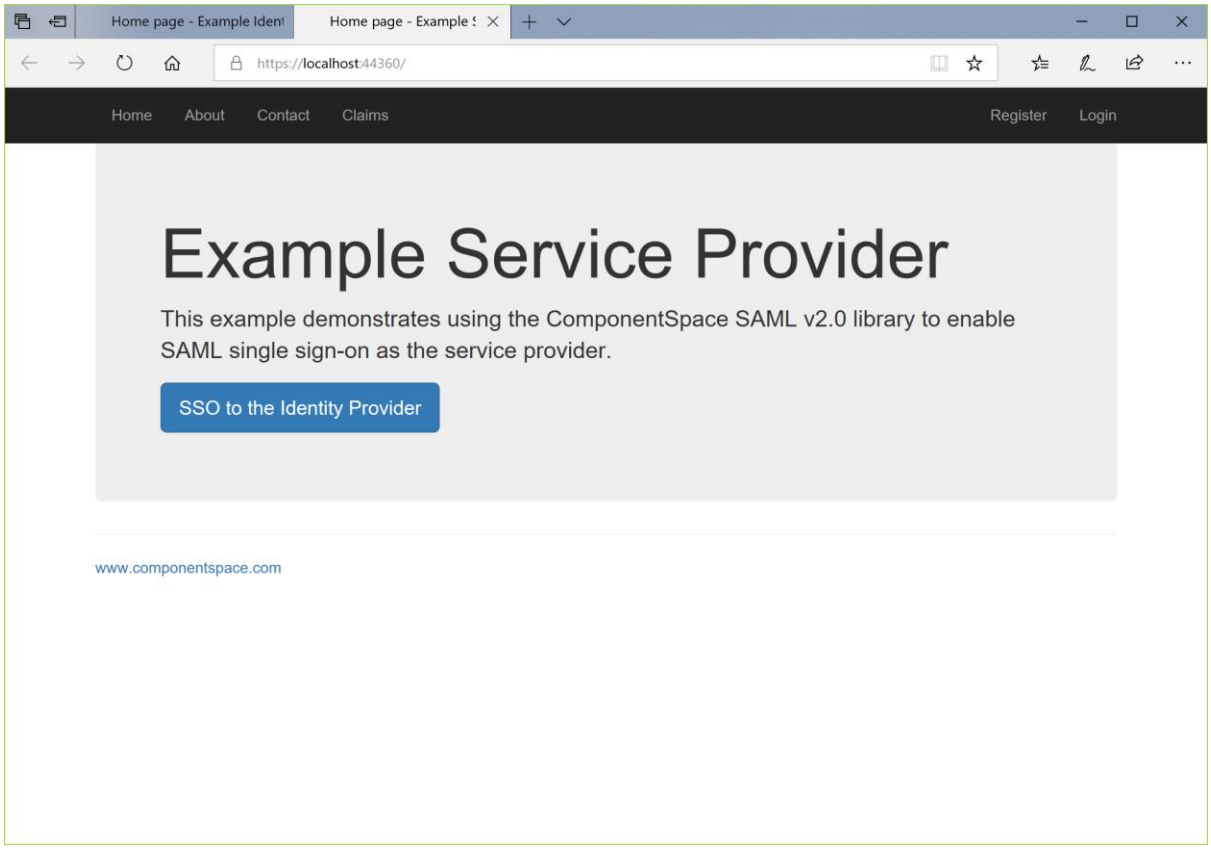
To run on IIS, the application must be configured in and published to IIS. It should use a database provider other than LocalDB.

The application is configured to run at https://localhost:44360/.

If this is changed, the corresponding ExampleIdentityProvider's SAML configuration must be updated to match the new URLs.

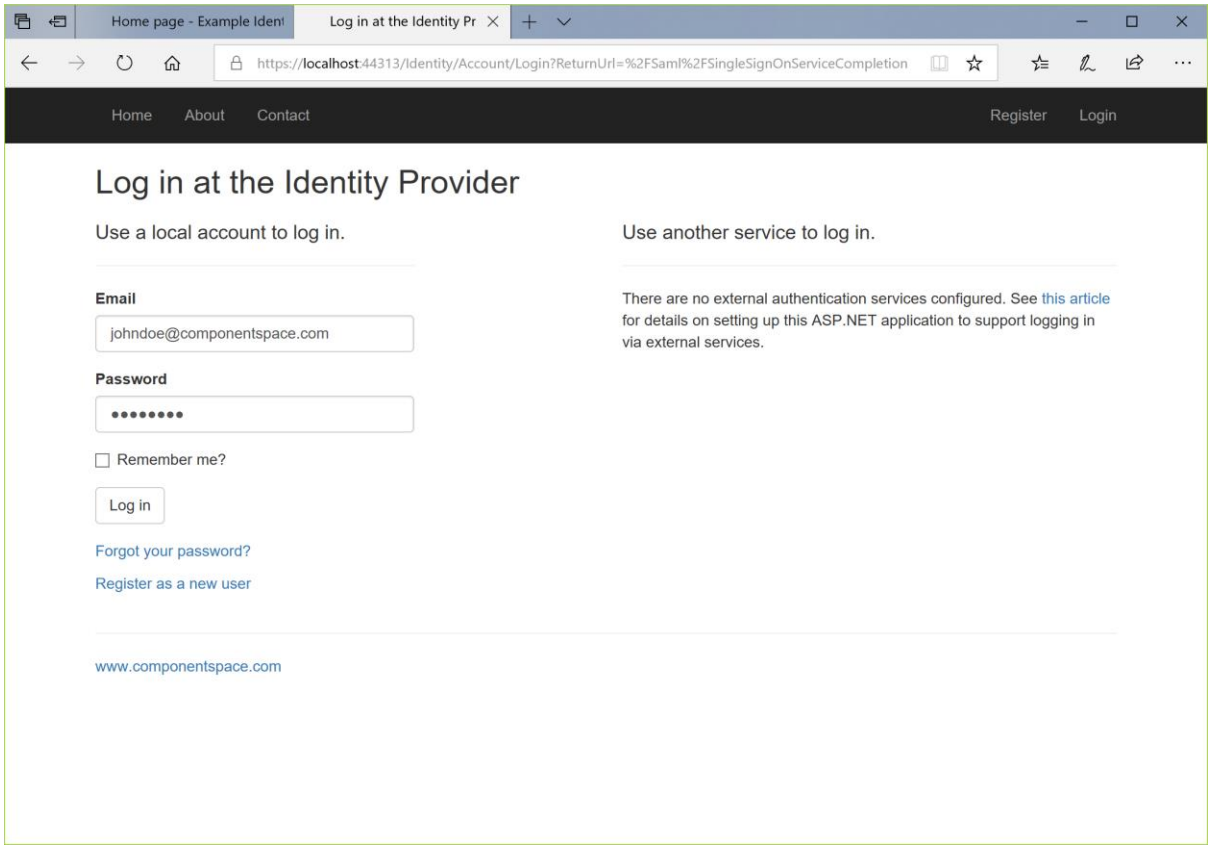
SP-initiated SSO

Browse to the example service provider's home page at https://localhost: 44360/.

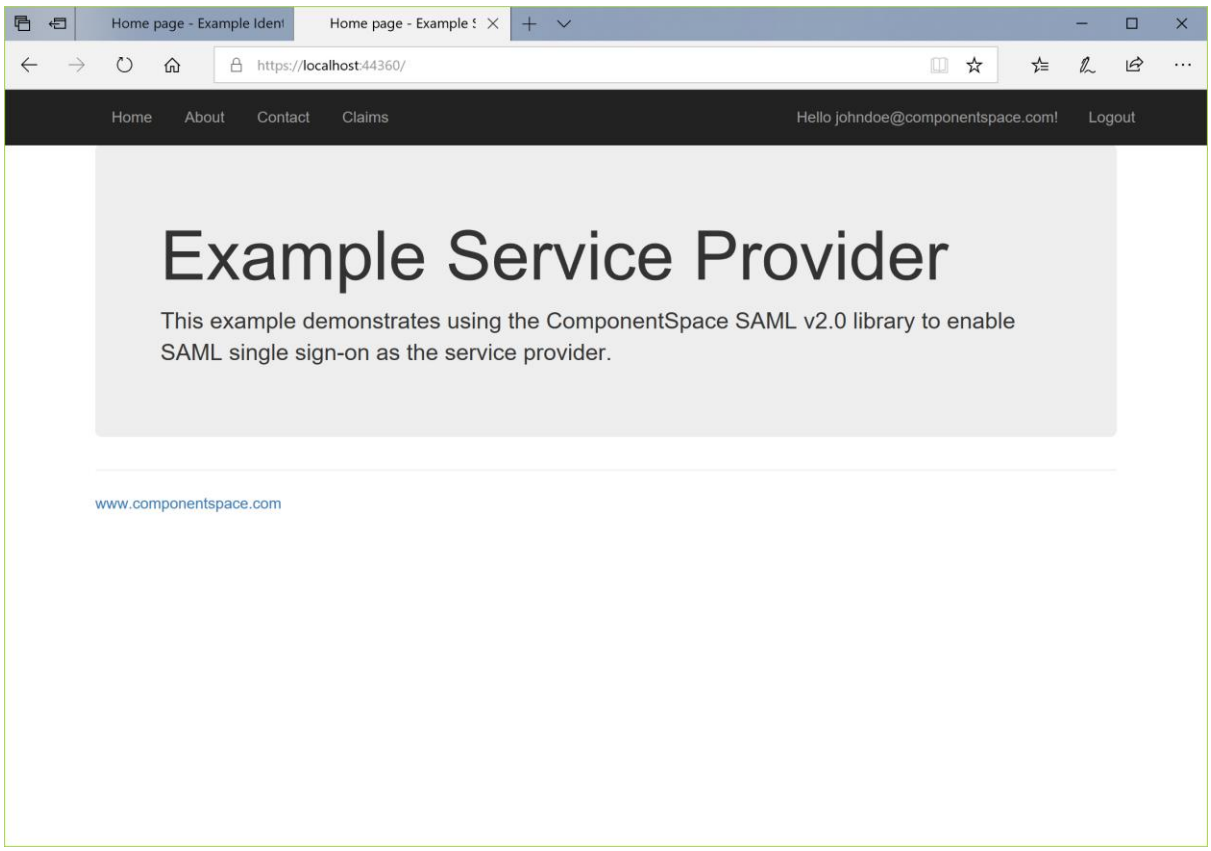


Click the SSO to the Identity Provider button.

You are prompted to login at the identity provider.

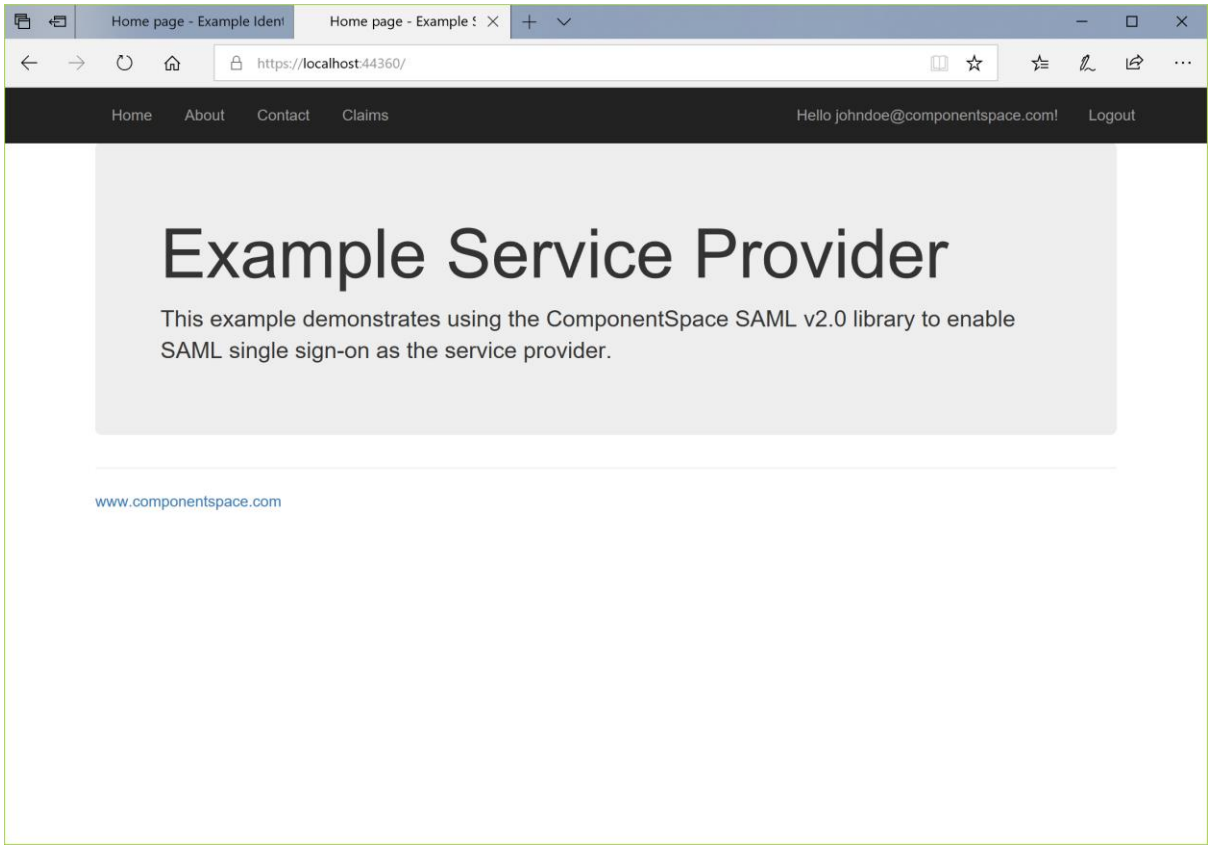


SSO completes with automatic login at the service provider. The user identity is that specified by the identity provider.

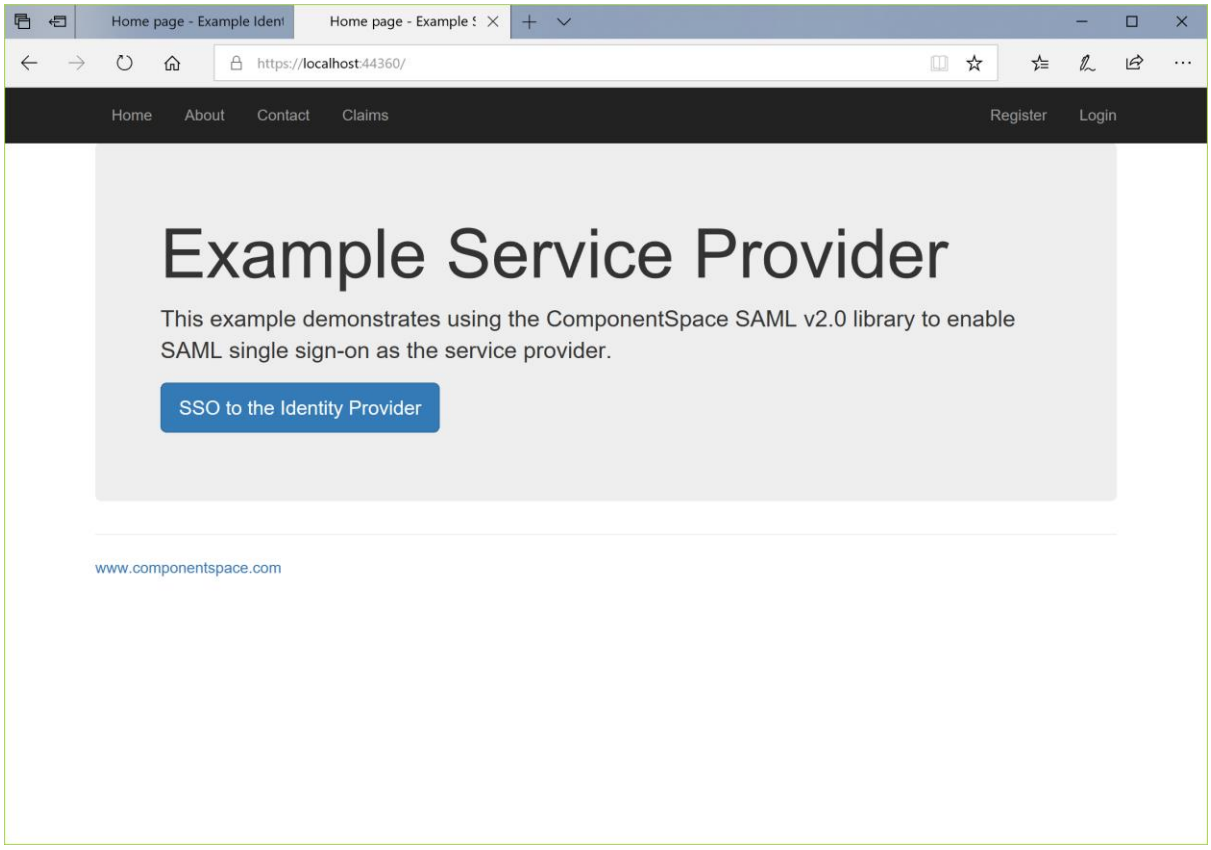


SP-initiated SLO

Having completed SSO, in the same browser window, browse to the example service provider's home page at https://localhost:44360/.



Click the Log out link. Logout occurs at both the identity provider and service provider.



Middleware Identity Provider

The MiddlewareIdentityProvider project is an ASP.NET Core web application based off the Visual Studio template.

It demonstrates acting as a SAML identity provider and supports:

- IdP-initiated SSO
- SP-initiated SSO
- IdP-initiated SLO
- SP-initiated SLO

Rather than making explicit SAML API calls, the SAML middleware is used to support SSO.

Through changes to its SAML configuration, it can support all SAML v2.0 compliant third-party offerings.

Building and Running

The MiddlewareIdentityProvider should build without any errors or warnings.

As it is configured to use the default LocalDB connection string, the simplest approach is to run the application on IIS Express through the Visual Studio debugger.

Note that this database is not used by the SAML API but is the application's user registry.

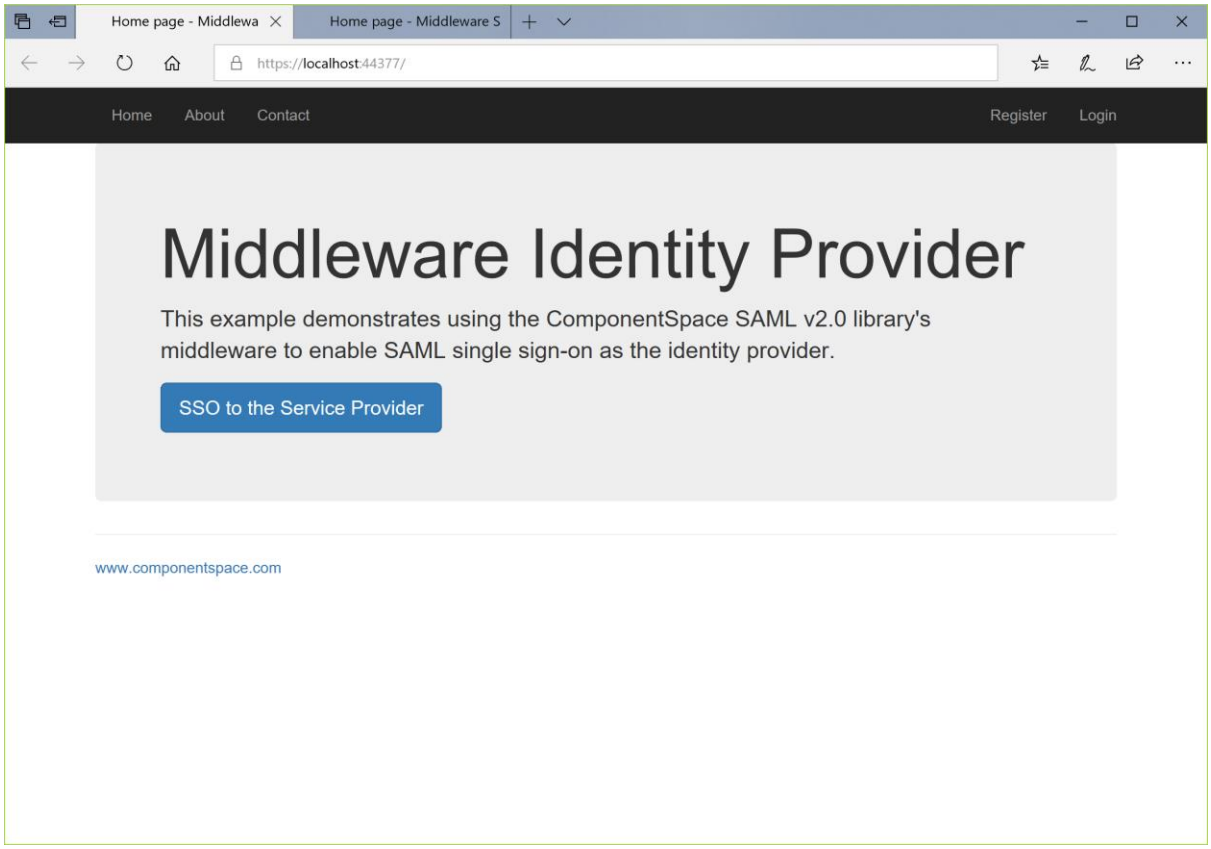
To run on IIS, the application must be configured in and published to IIS. It should use a database provider other than LocalDB.

The application is configured to run at https://localhost:44377/.

If this is changed, the corresponding ExampleServiceProvider's and MiddlewareServiceProvider's SAML configuration must be updated to match the new URLs.

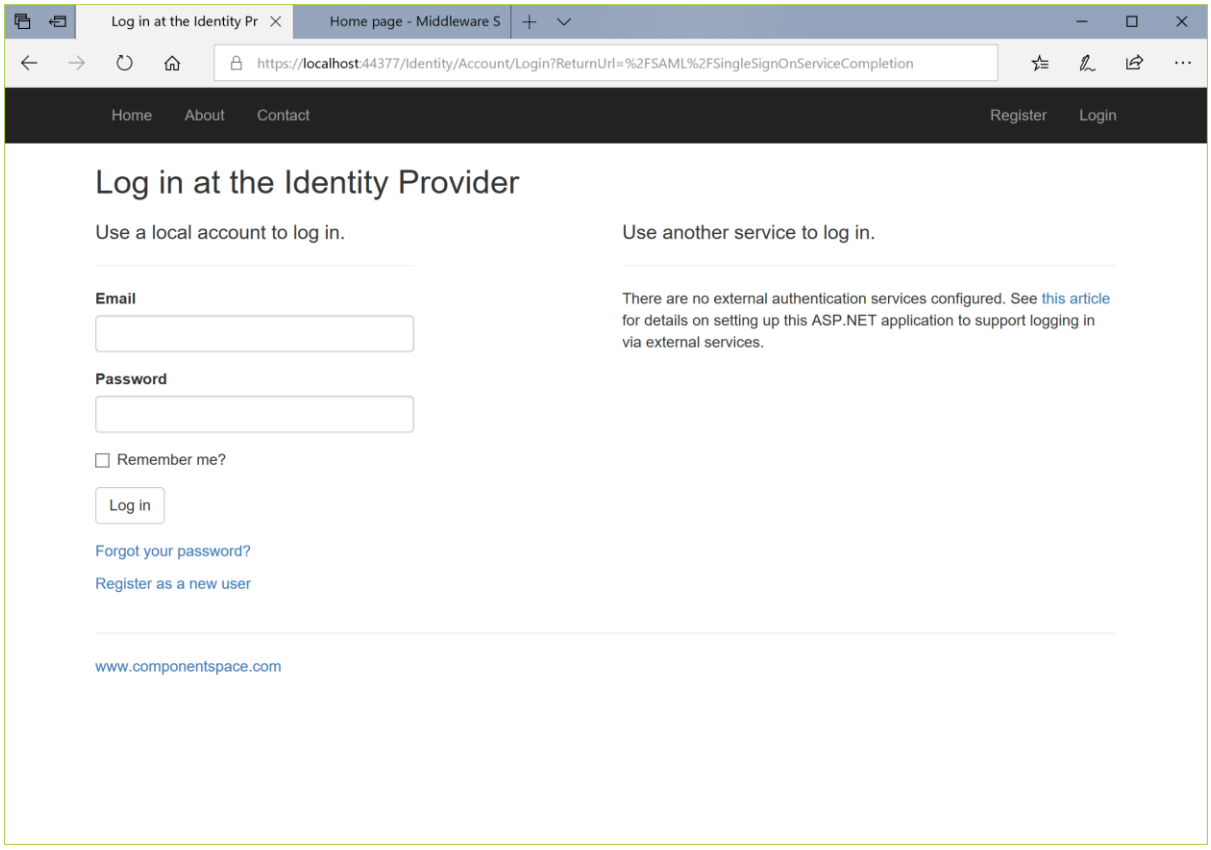
IdP-initiated SSO

Browse to the middleware identity provider's home page at http://localhost:44377/.



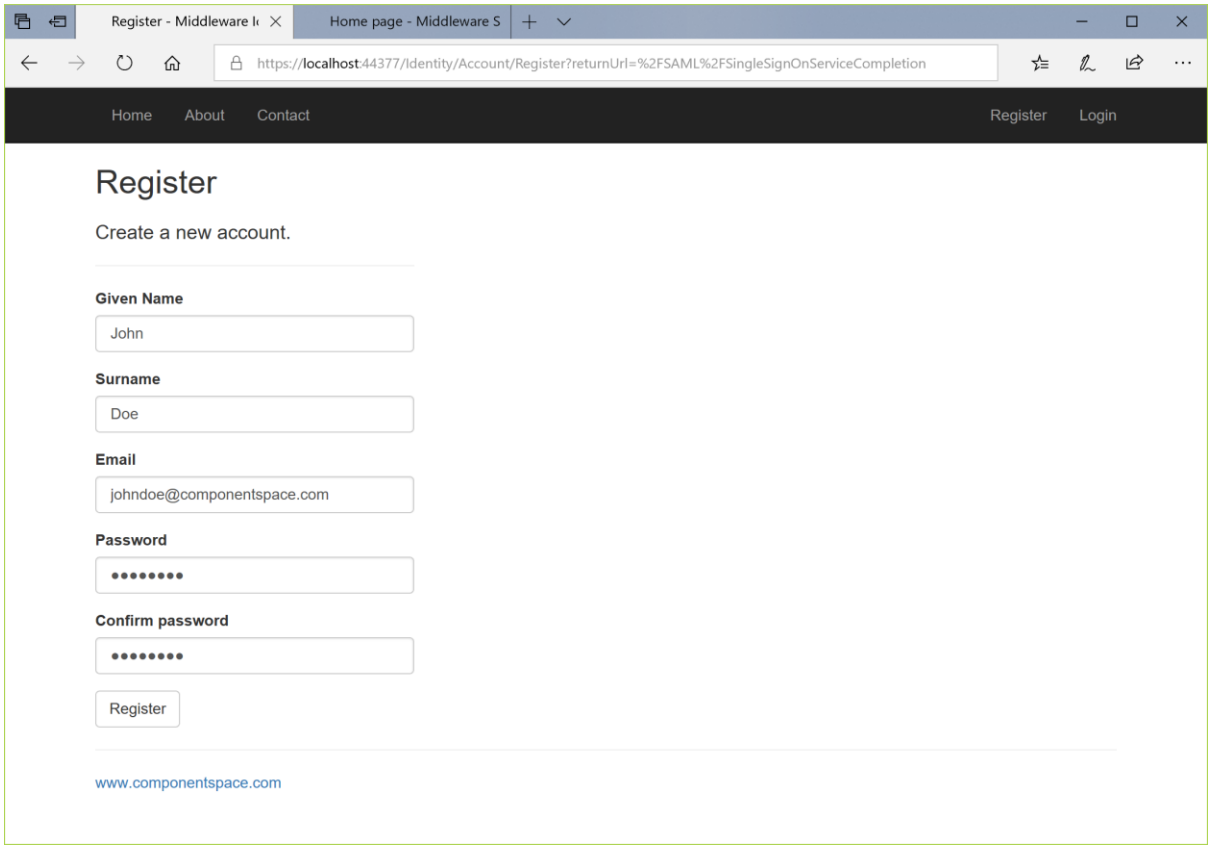
Click the SSO to the Service Provider button.

As you haven't been authenticated at the MiddlewareIdentityProvider, you are prompted to login or register.

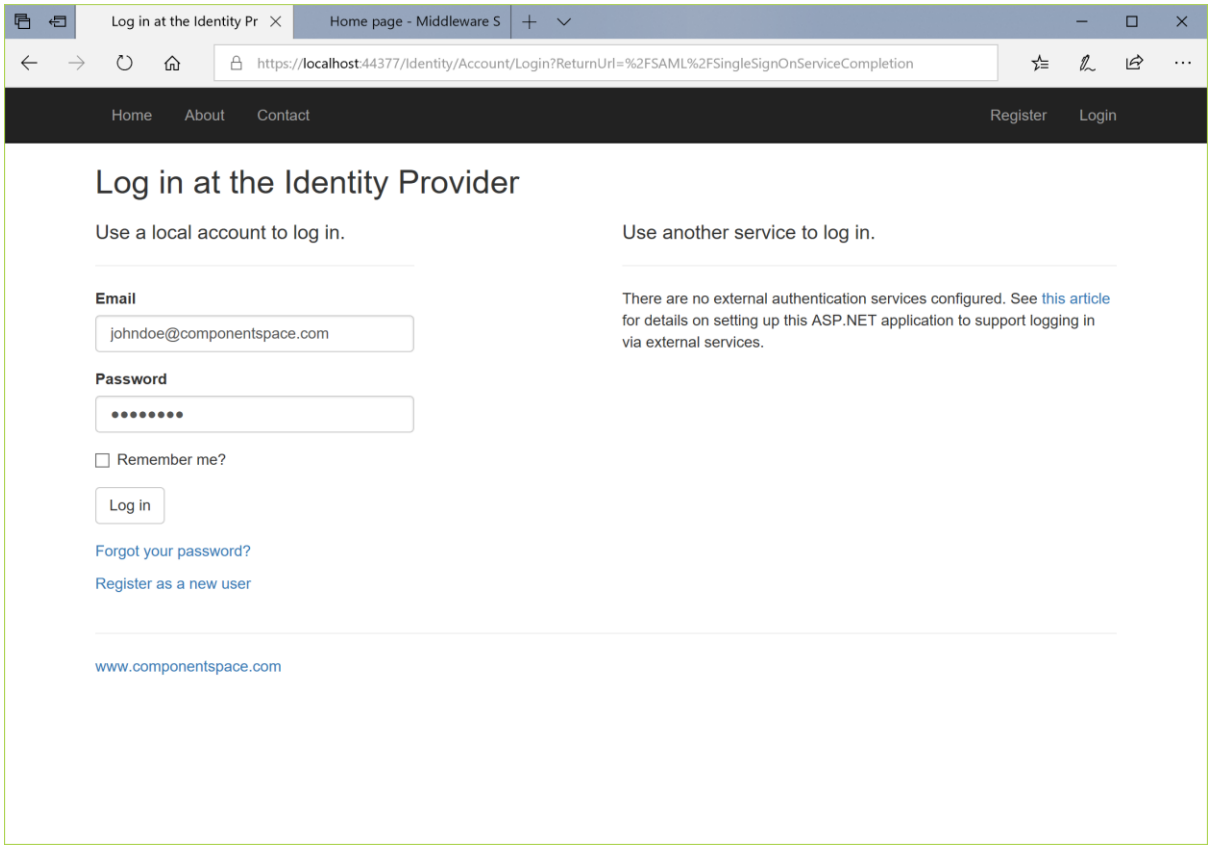


Click the link to register as a new user.

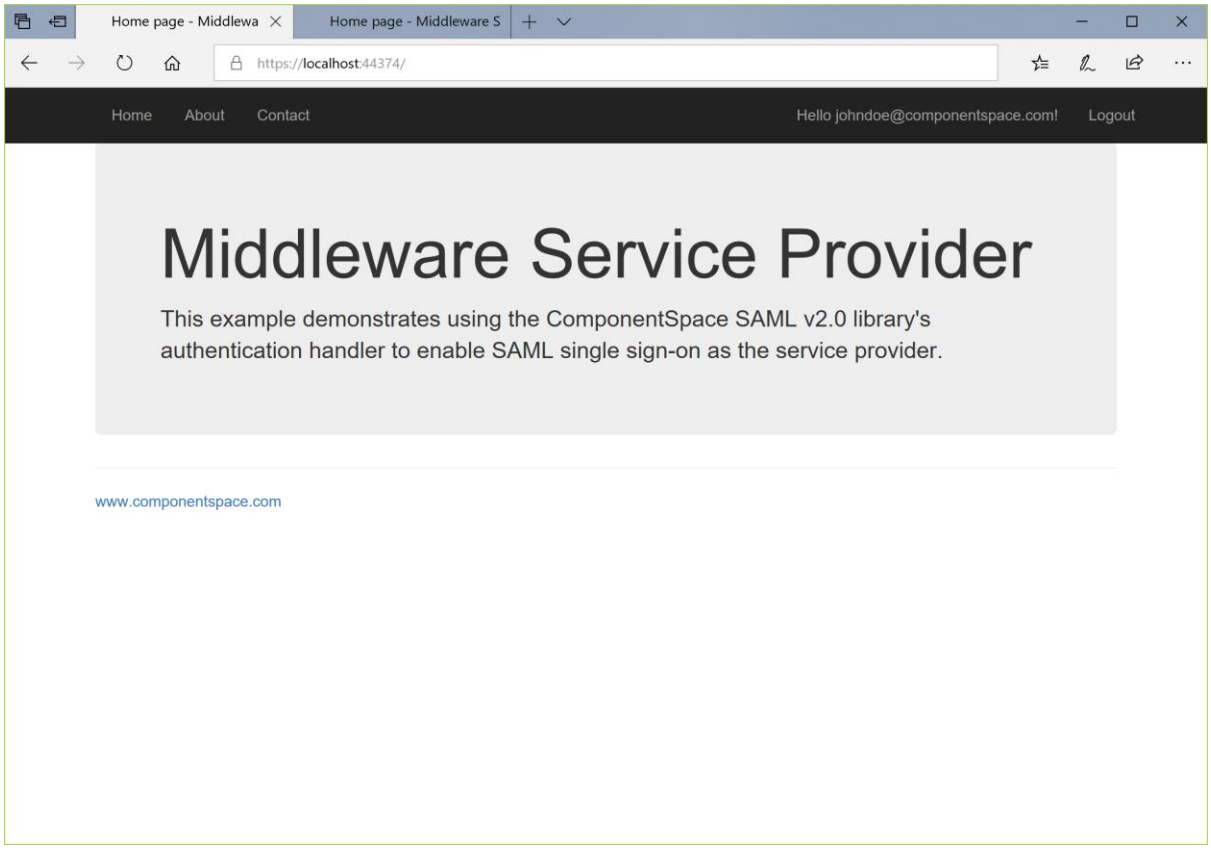
Complete the registration process.



If you've previously registered, simply login.

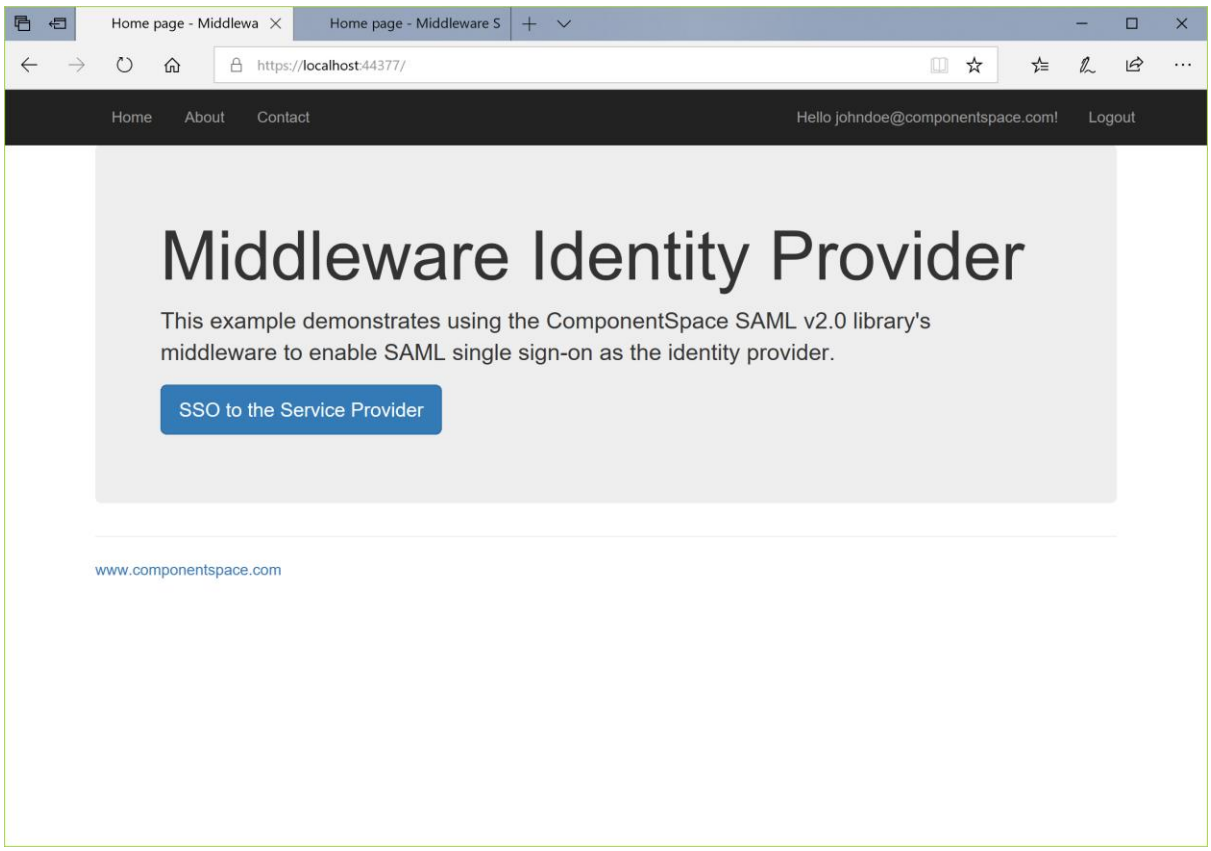


SSO completes with automatic login at the service provider. The user identity is that specified by the identity provider.

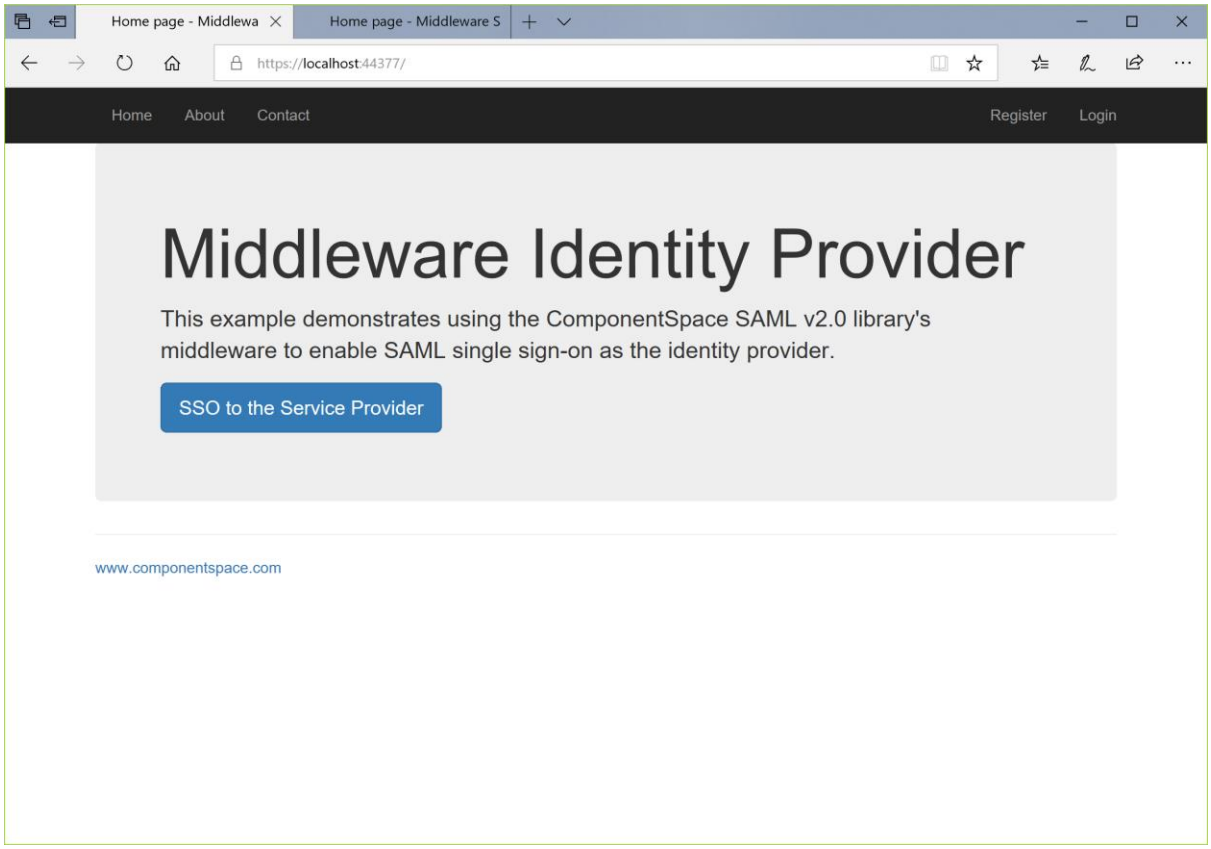


IdP-initiated SLO

Having completed SSO, in the same browser window, browse to the example identity provider's home page at https://localhost:44377/.



Click the Log out link. Logout occurs at both the identity provider and service provider.



Middleware Service Provider

The MiddlewareServiceProvider project is an ASP.NET Core web application based off the Visual Studio template.

It demonstrates acting as a SAML service provider and supports:

- IdP-initiated SSO
- SP-initiated SSO
- IdP-initiated SLO
- SP-initiated SLO

Rather than making explicit SAML API calls, the SAML authentication handler is used to support SSO.

Through changes to its SAML configuration, it can support all SAML v2.0 compliant third-party offerings.

Building and Running

The MiddlewareServiceProvider should build without any errors or warnings.

As it is configured to use the default LocalDB connection string, the simplest approach is to run the application on IIS Express through the Visual Studio debugger.

Note that this database is not used by the SAML API but is the application's user registry.

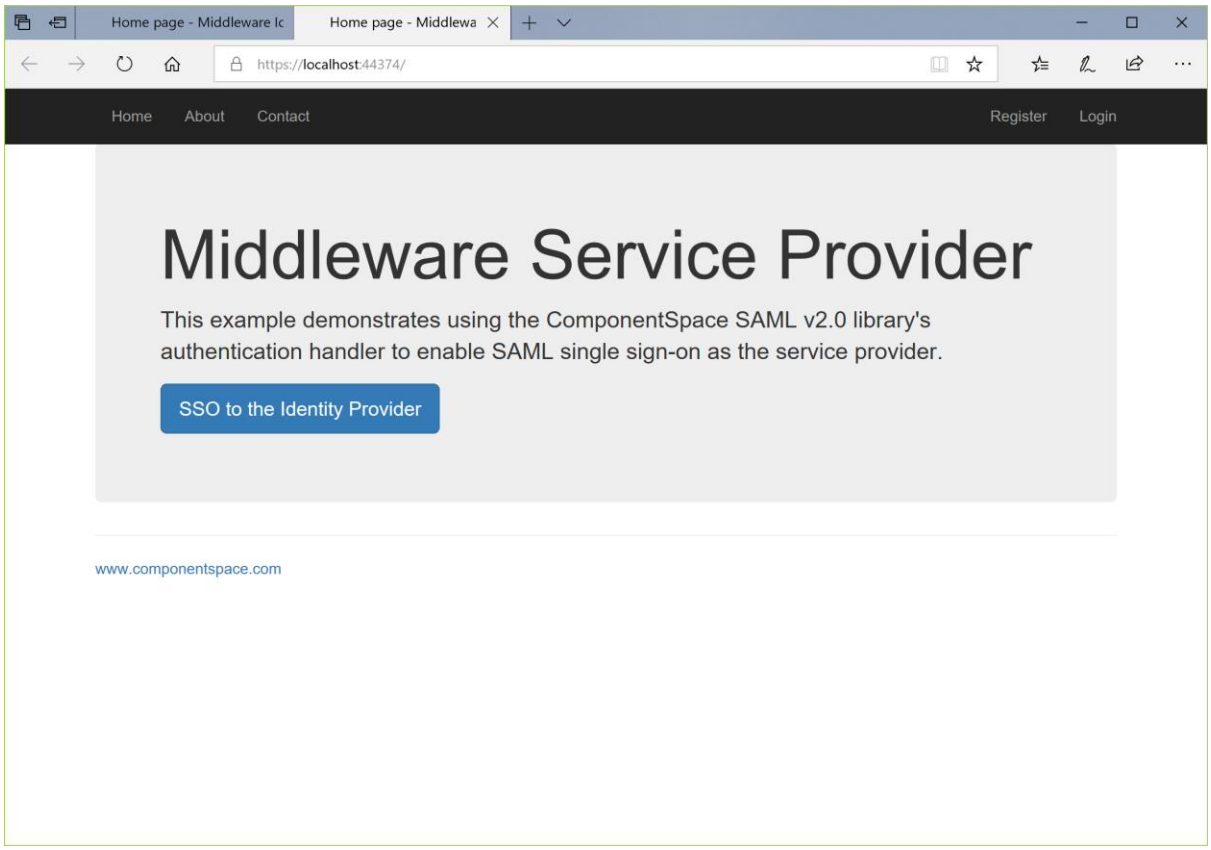
To run on IIS, the application must be configured in and published to IIS. It should use a database provider other than LocalDB.

The application is configured to run at https://localhost:44374/.

If this is changed, the corresponding ExampleIdentityProvider's SAML configuration must be updated to match the new URLs.

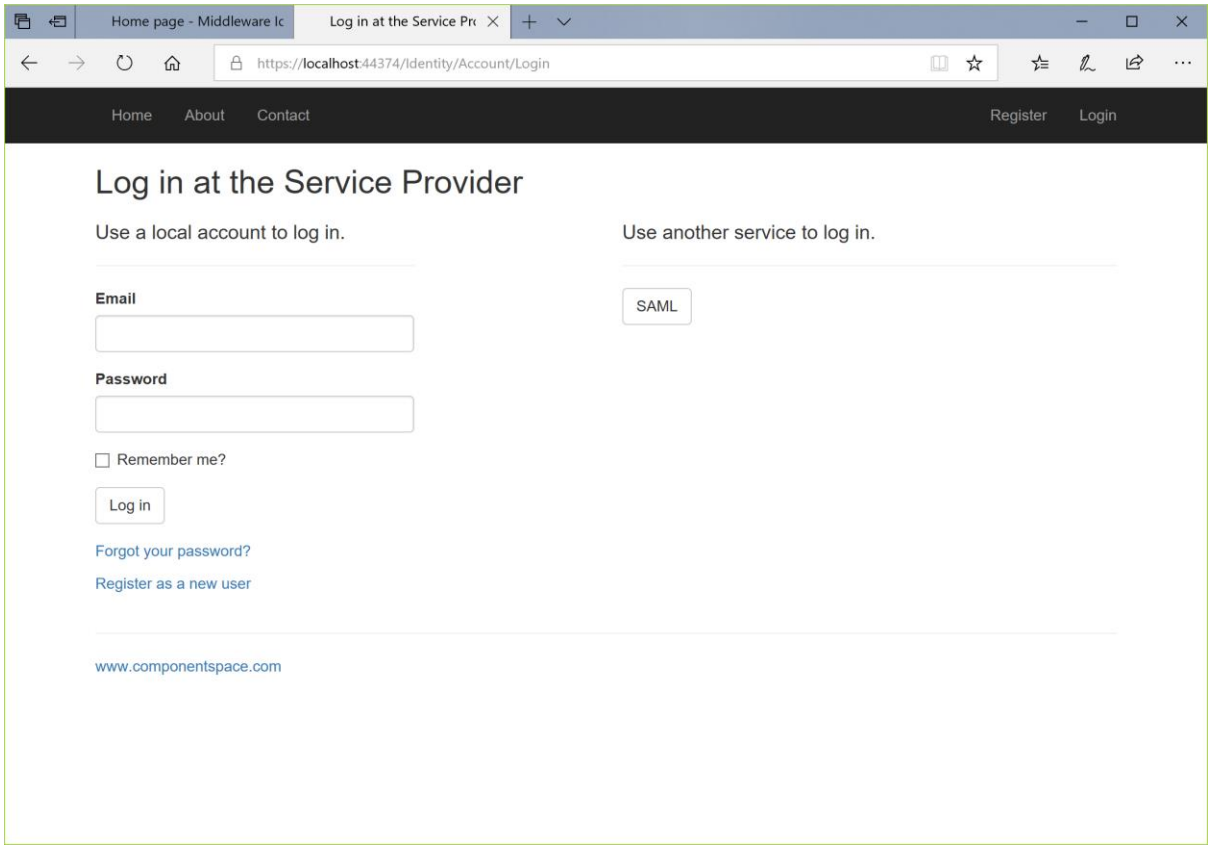
SP-initiated SSO

Browse to the middleware service provider's home page at https://localhost:44374/.

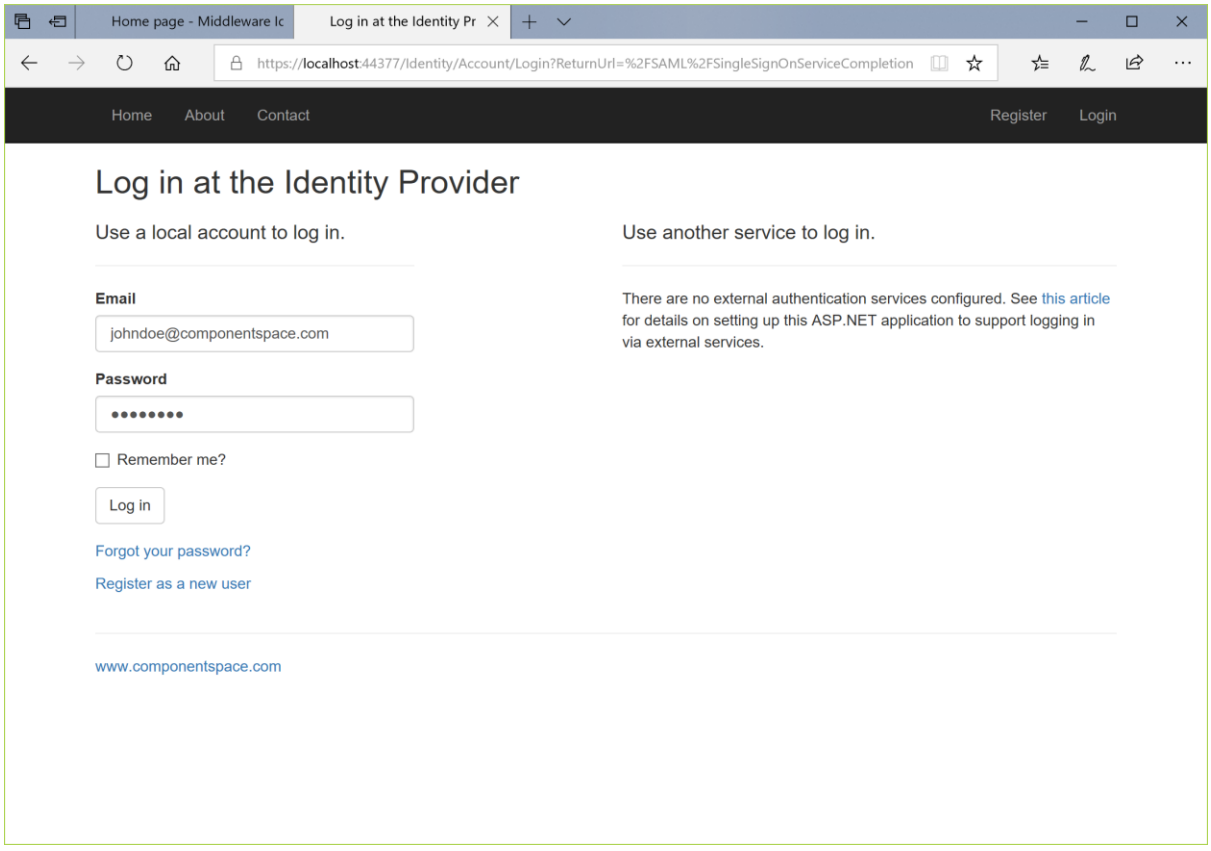


Click the SSO to the Identity Provider button.

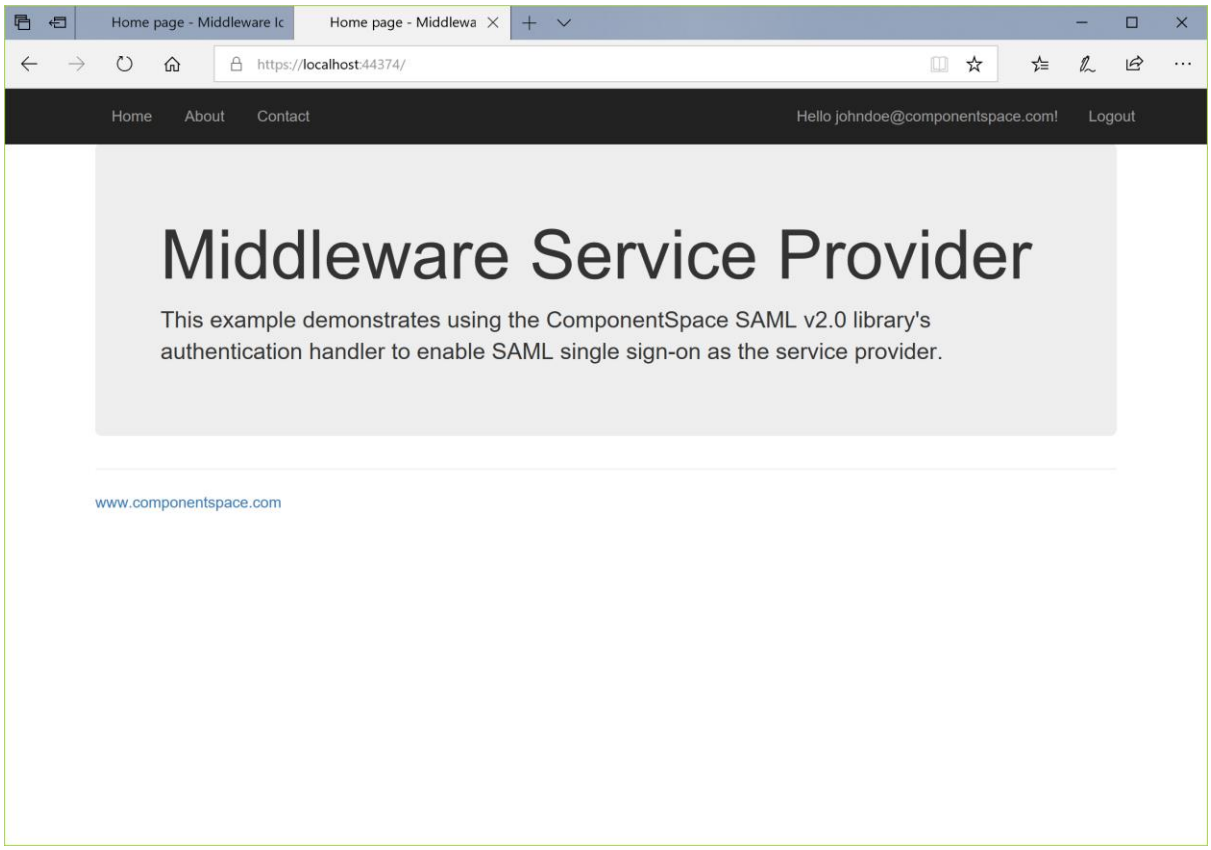
Alternatively, click the Log in link and then the SAML button.



You are prompted to login at the identity provider.

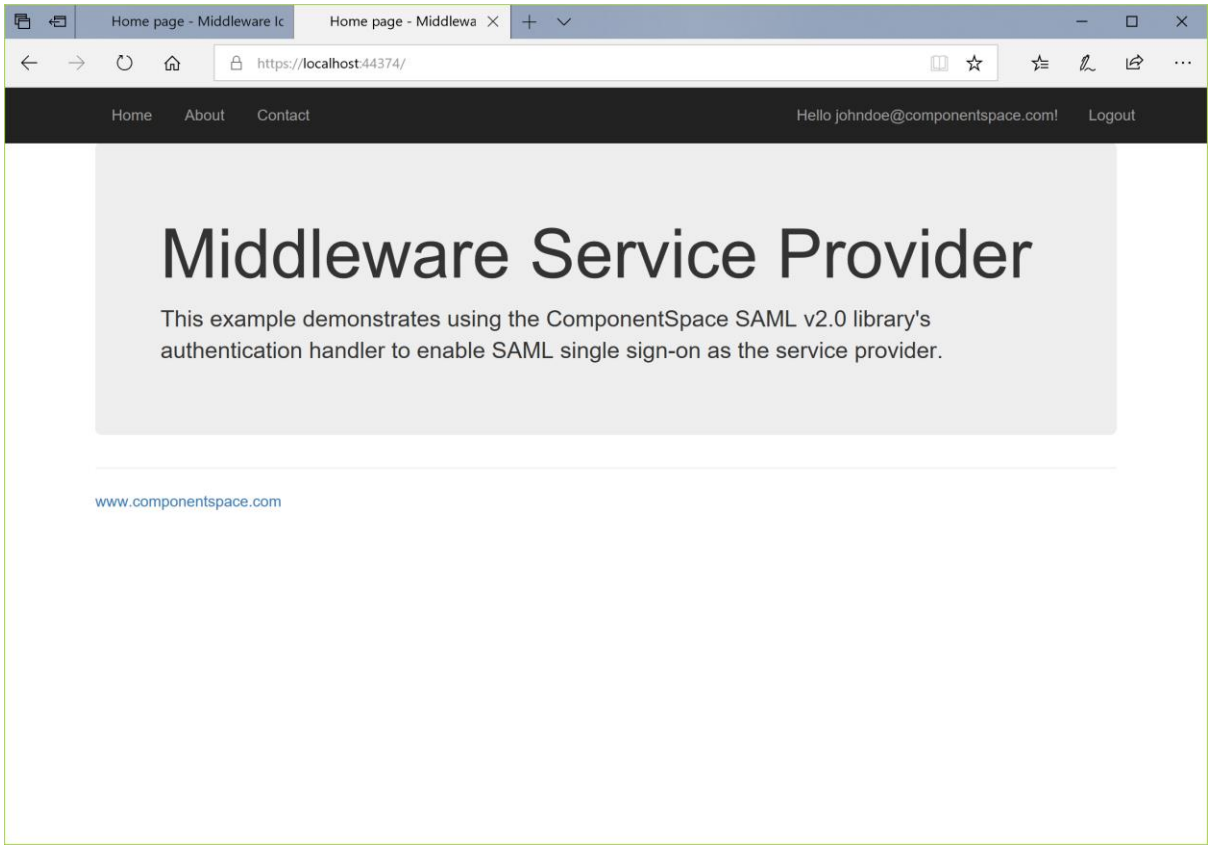


SSO completes with automatic login at the service provider. The user identity is that specified by the identity provider.

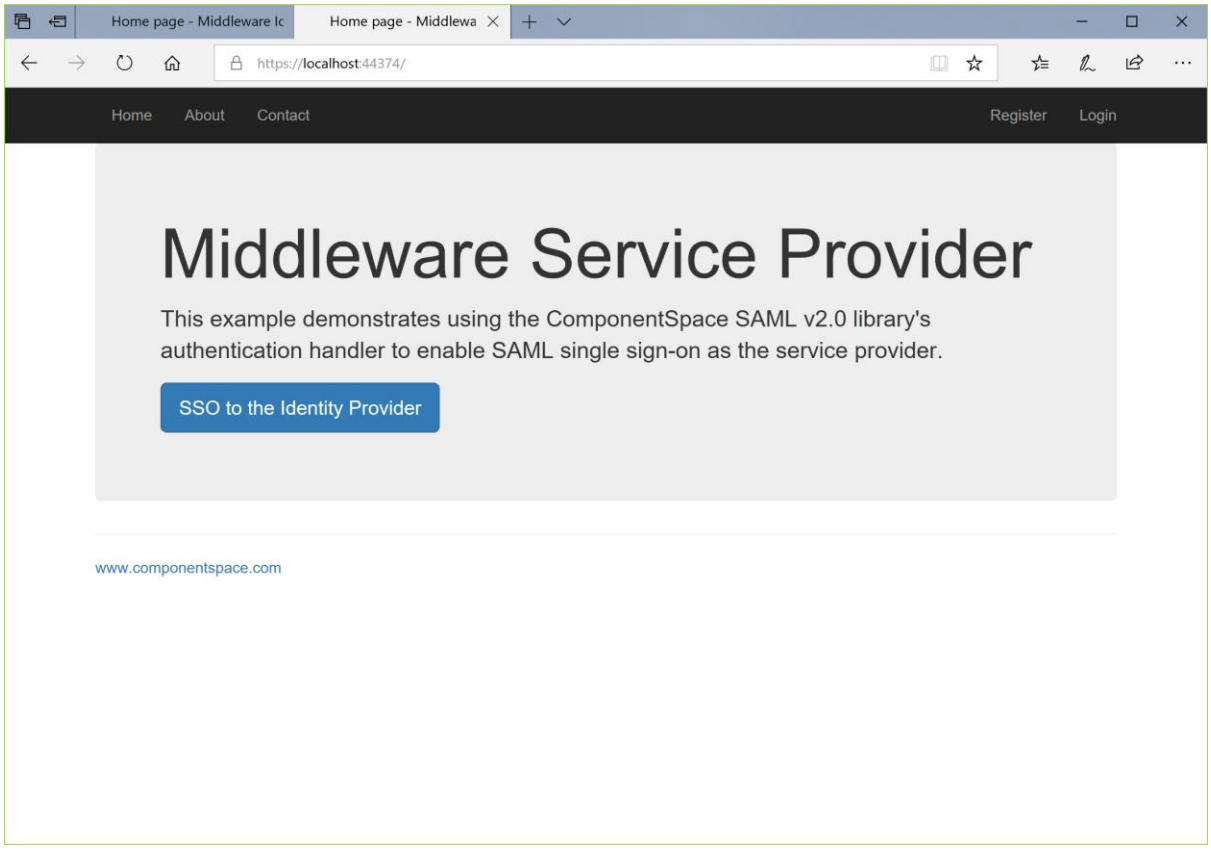


SP-initiated SLO

Having completed SSO, in the same browser window, browse to the middleware service provider's home page at https://localhost:44374/.



Click the Log out link. Logout occurs at both the identity provider and service provider.



Example Web API

The ExampleWebApi project is an ASP.NET Core web application based off the Visual Studio template.

It demonstrates a web API acting as a SAML service provider and supports:

- IdP-initiated SSO
- SP-initiated SSO
- IdP-initiated SLO
- SP-initiated SLO

Through changes to its SAML configuration, it can support all SAML v2.0 compliant third-party offerings.

In conjunction with a JavaScript SPA, the ExampleWebApi demonstrates authentication through SAML SSO and web API authorization through JWT bearer tokens.

It doesn't demonstrate revoking JWT bearer tokens on logout but this functionality could be added.

Building and Running

The ExampleWebApi should build without any errors or warnings.

The application is configured to run at https://localhost:44319/.

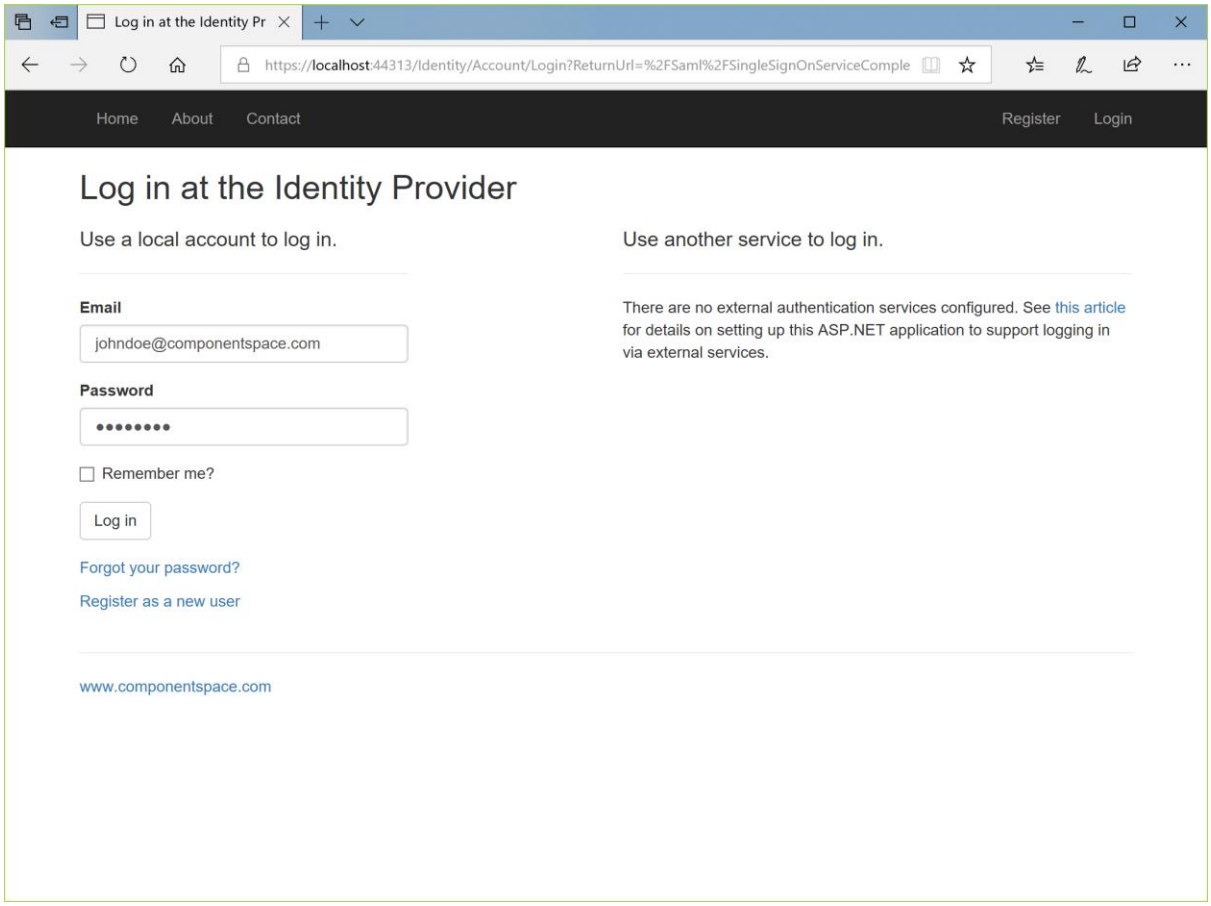
If this is changed, the corresponding ExampleIdentityProvider's SAML configuration must be updated to match the new URLs.

SP-initiated SSO

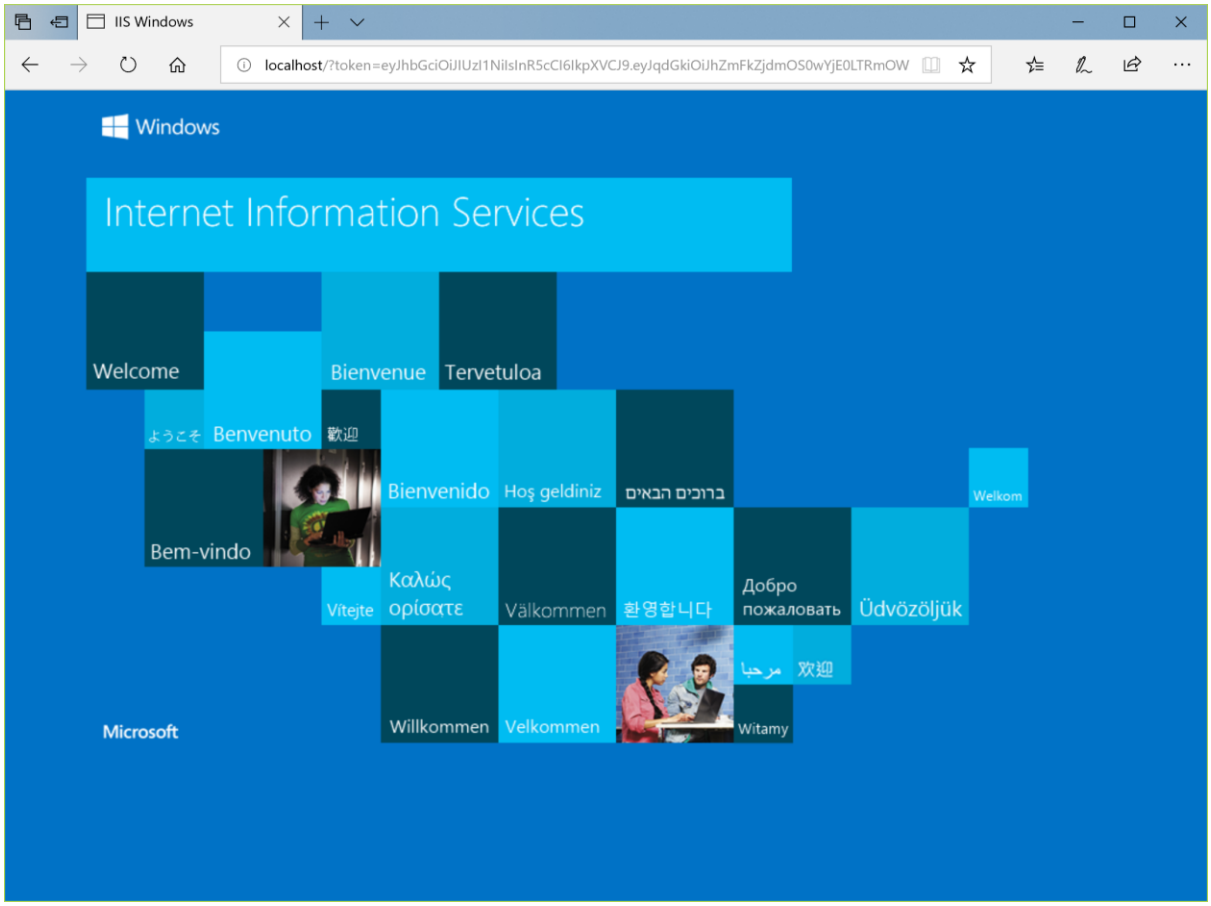
Browse to https://localhost:44319/Saml/InitiateSingleSignOn?returnUrl=http://localhost.

The returnUrl parameter must be present but it doesn't need to specify an actual web page.

You are prompted to login at the identity provider.



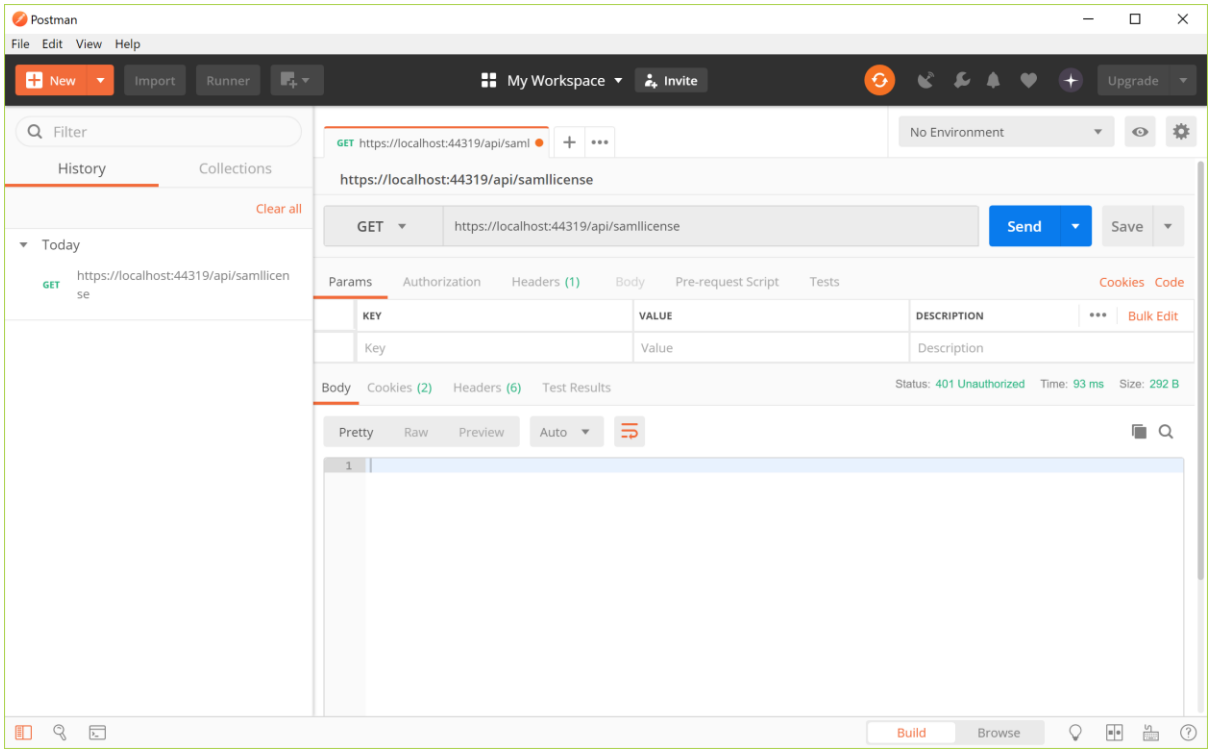
SSO completes with the generated JWT returned as the token query string parameter.



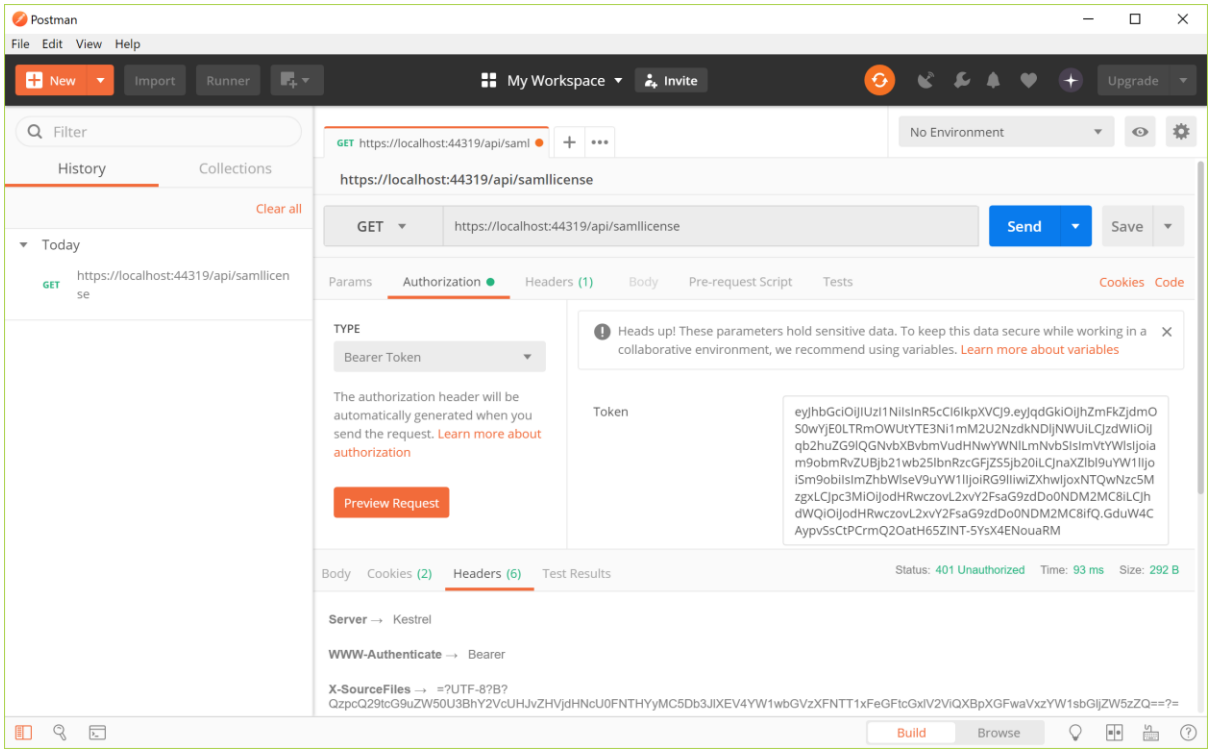
Save a copy of the token.

Use Postman to access the web API.

A 401 unauthorized status code is returned as a valid JWT bearer token wasn't presented.

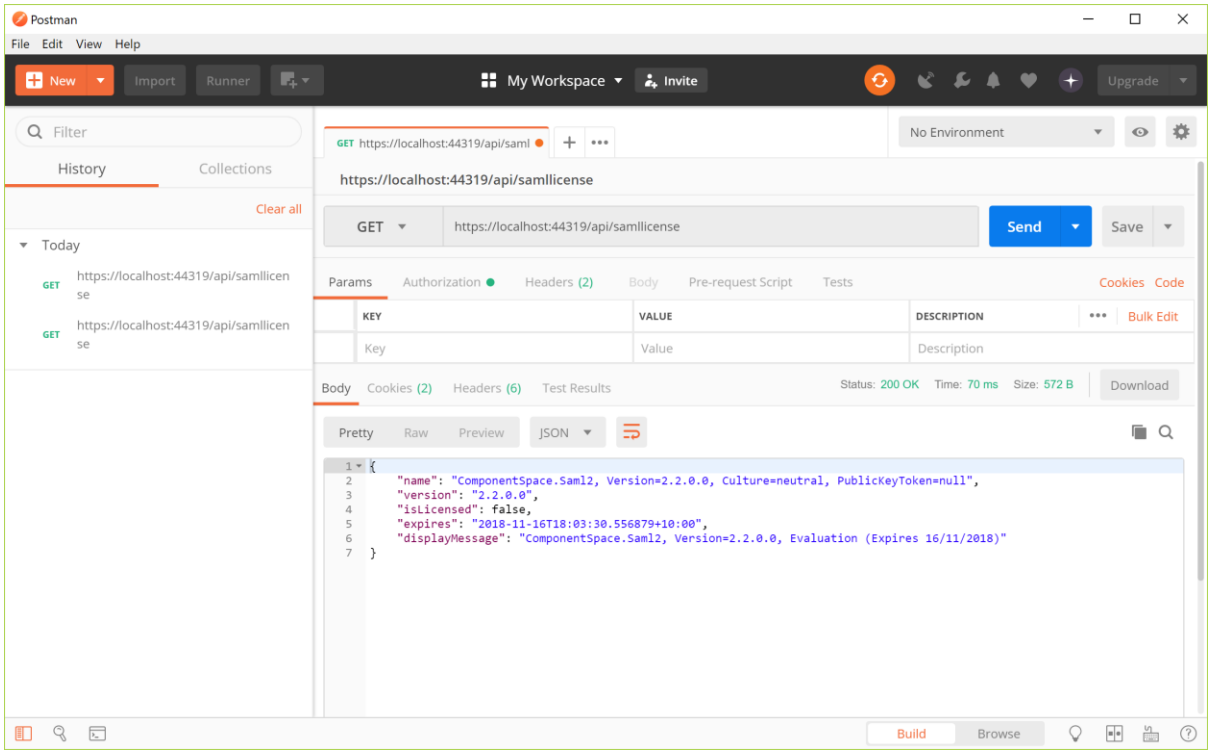


Specify the previously saved token as the bearer token.



Access the web API again.

A 200 OK status is returned along with the expected web API result.



SP-initiated SLO

Having completed SSO, in the same browser window, browse to https://localhost:44319/Saml/InitiateSingleLogout?returnUrl=http://localhost.

The returnUrl parameter must be present but it doesn't need to specify an actual web page.

Logout occurs at both the identity provider and service provider.

Example Angular SPA

The ExampleAngularSpa project is an Angular application created using the Angular CLI (https://cli.angular.io/).

In conjunction with the ExampleWebApi, it demonstrates authentication through SAML SSO and web API authorization through JWT bearer tokens.

Building and Running

The ExampleAngularSpa should build without any errors or warnings.

From the application's top-level folder, restore the packages.

```
npm install
```

From the application's top-level folder, run the application.

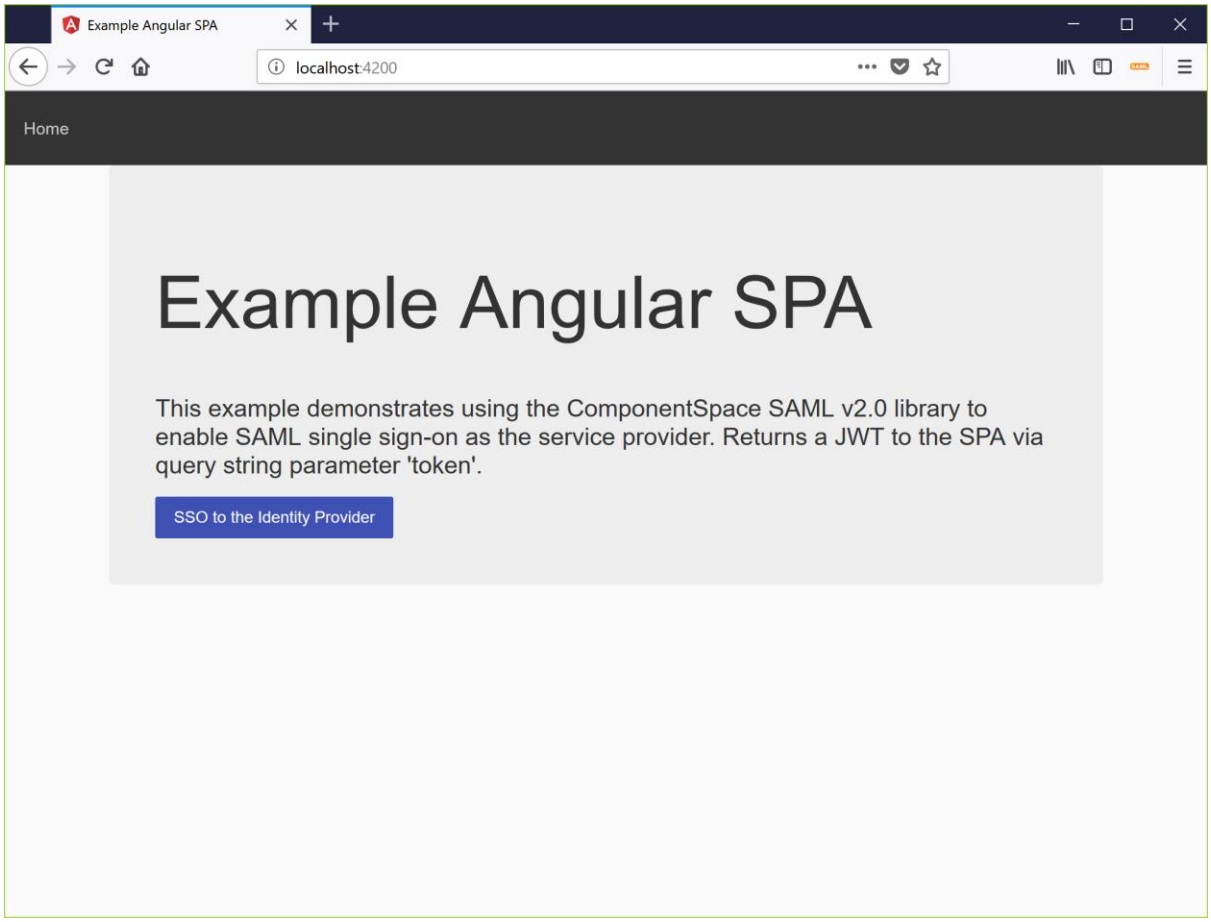
```
ng serve
```

The application is configured to run at http://localhost:4200/.

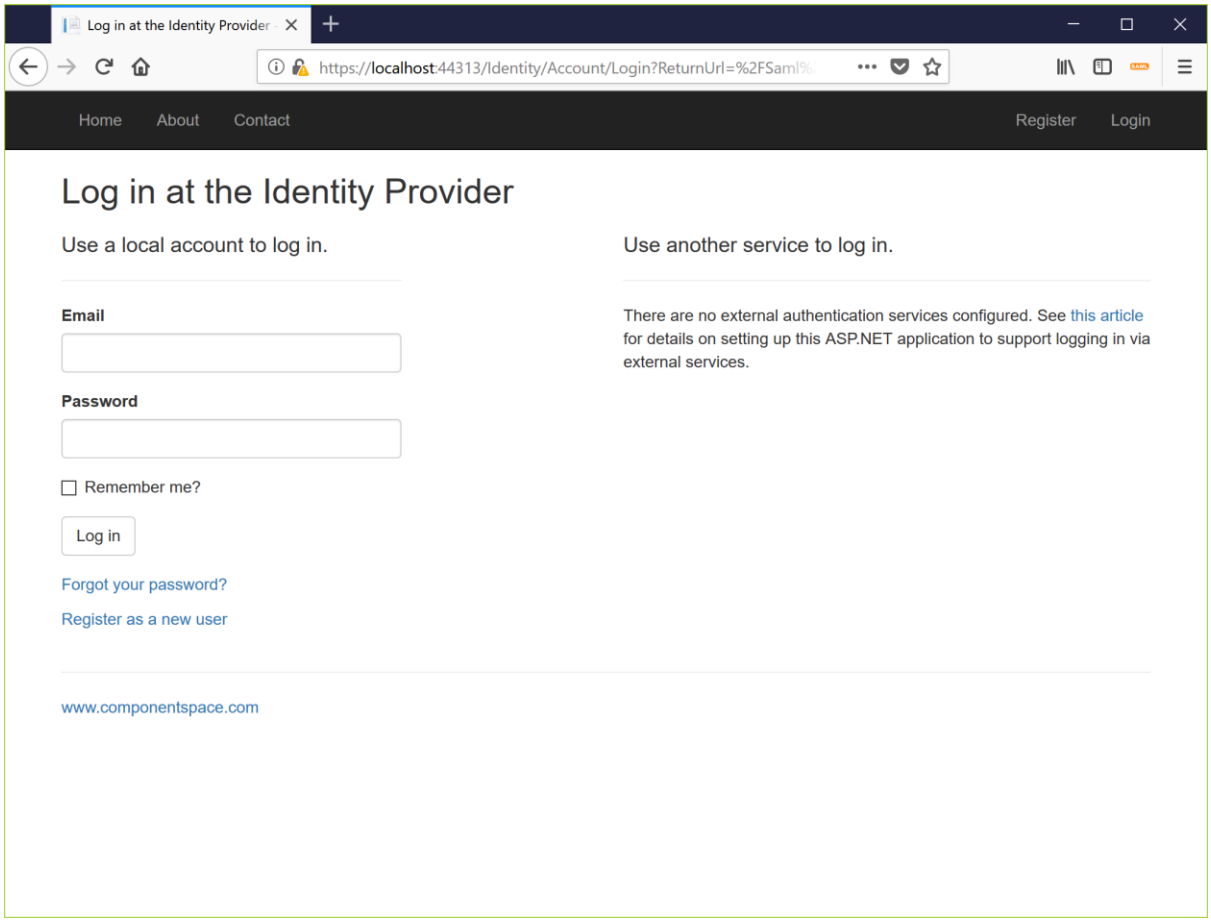
Ensure the ExampleWebApi and ExampleIdentityProvider projects are also running.

SP-initiated SSO

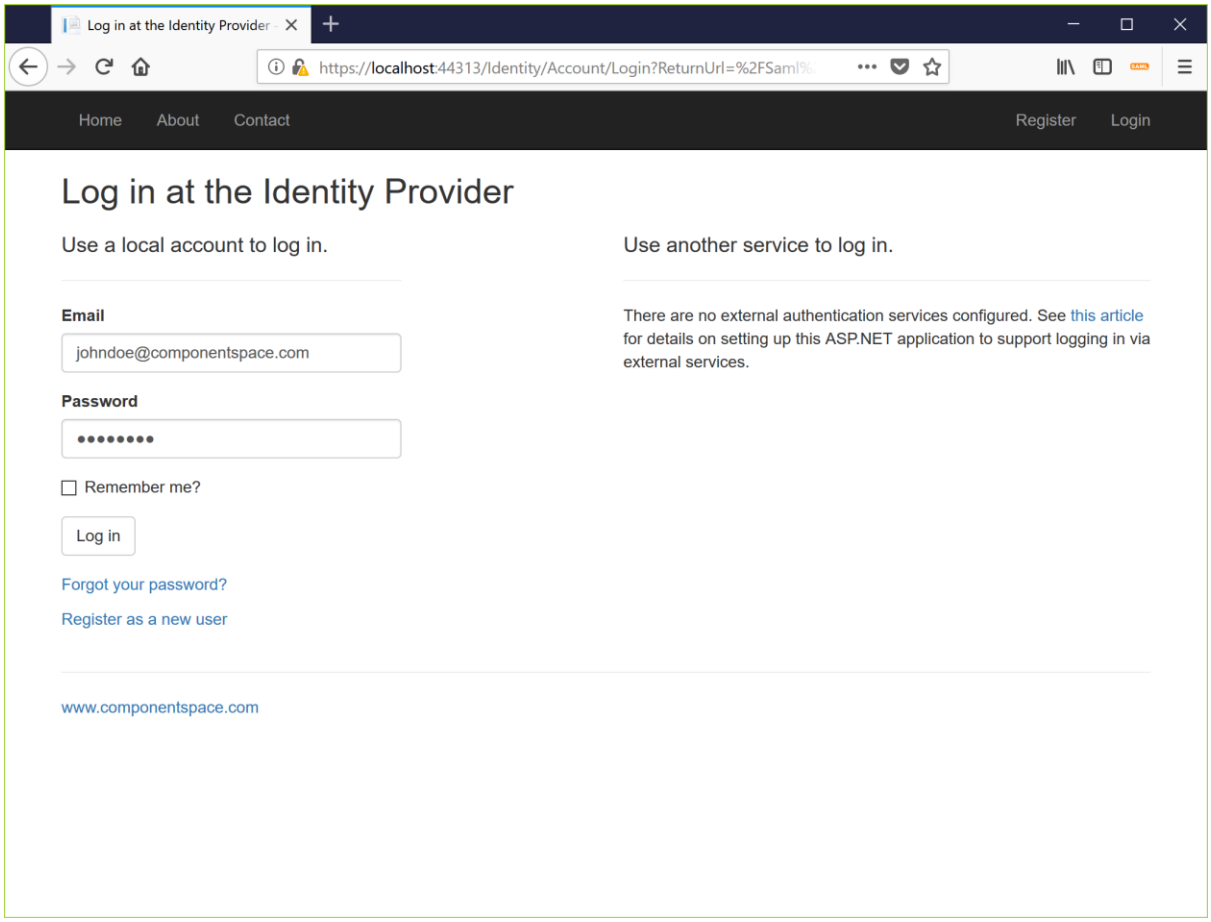
Browse to the Angular application at http://localhost:4200/.



Click the SSO to the Identity Provider button.

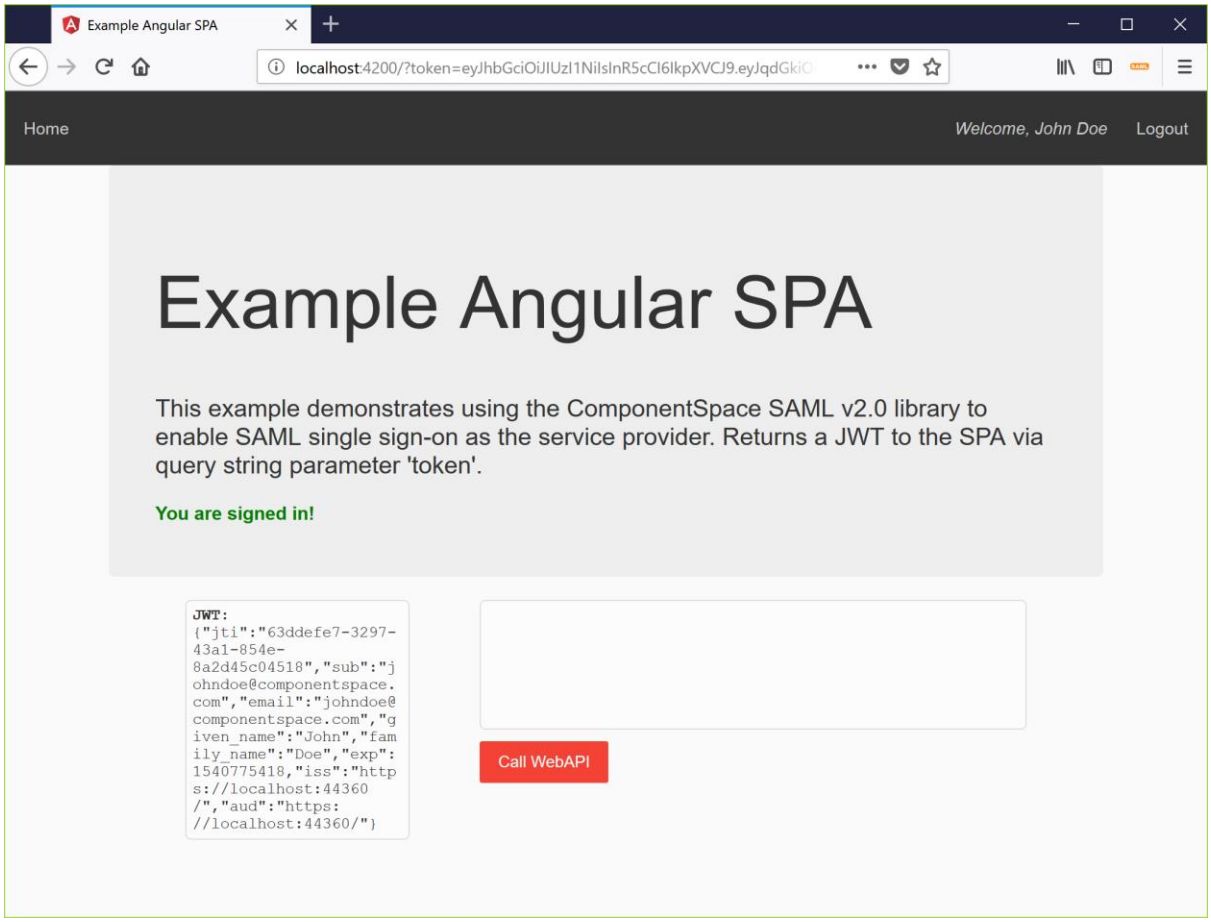


You are prompted to login at the identity provider.



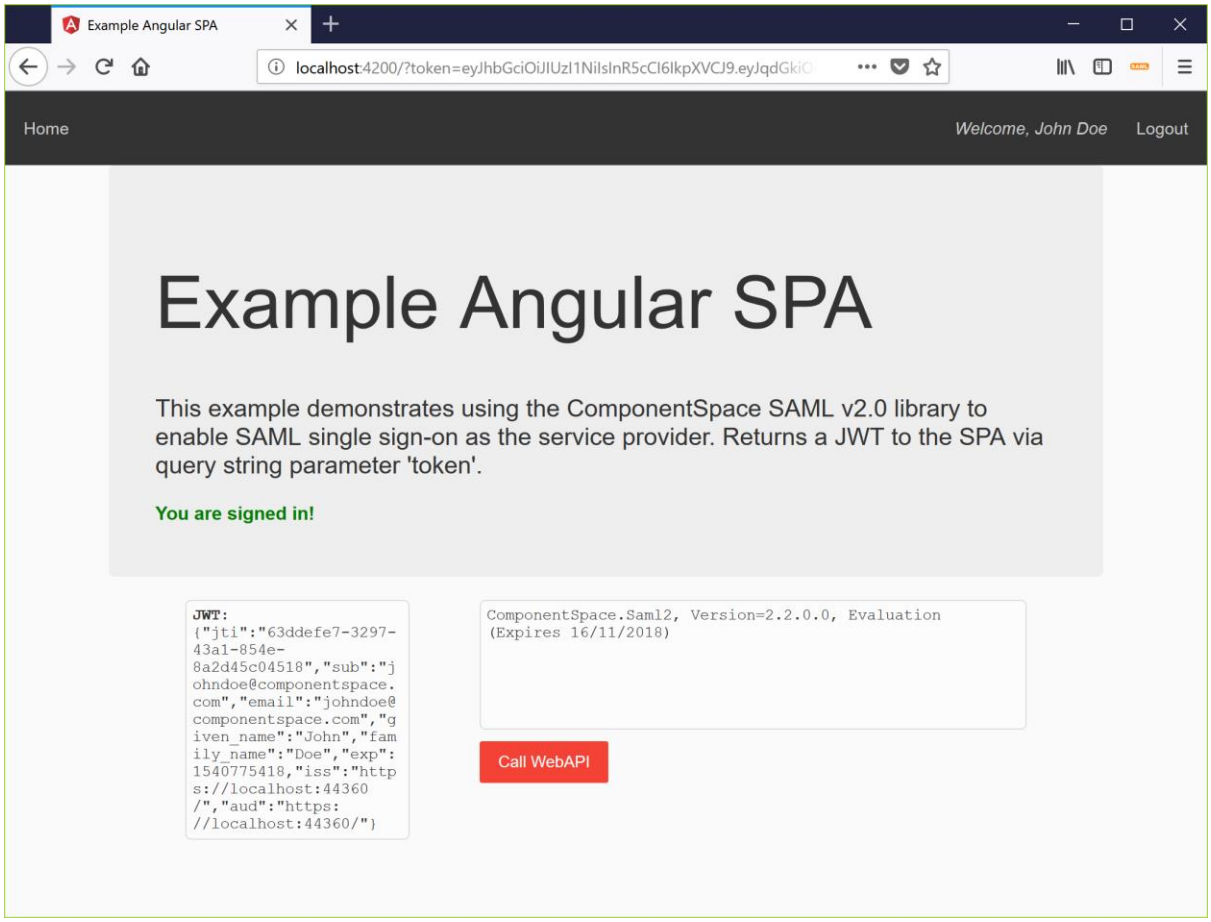
SSO completes with automatic login at the service provider. The user identity is that specified by the identity provider.

For informational purposes only, the returned JWT is displayed.



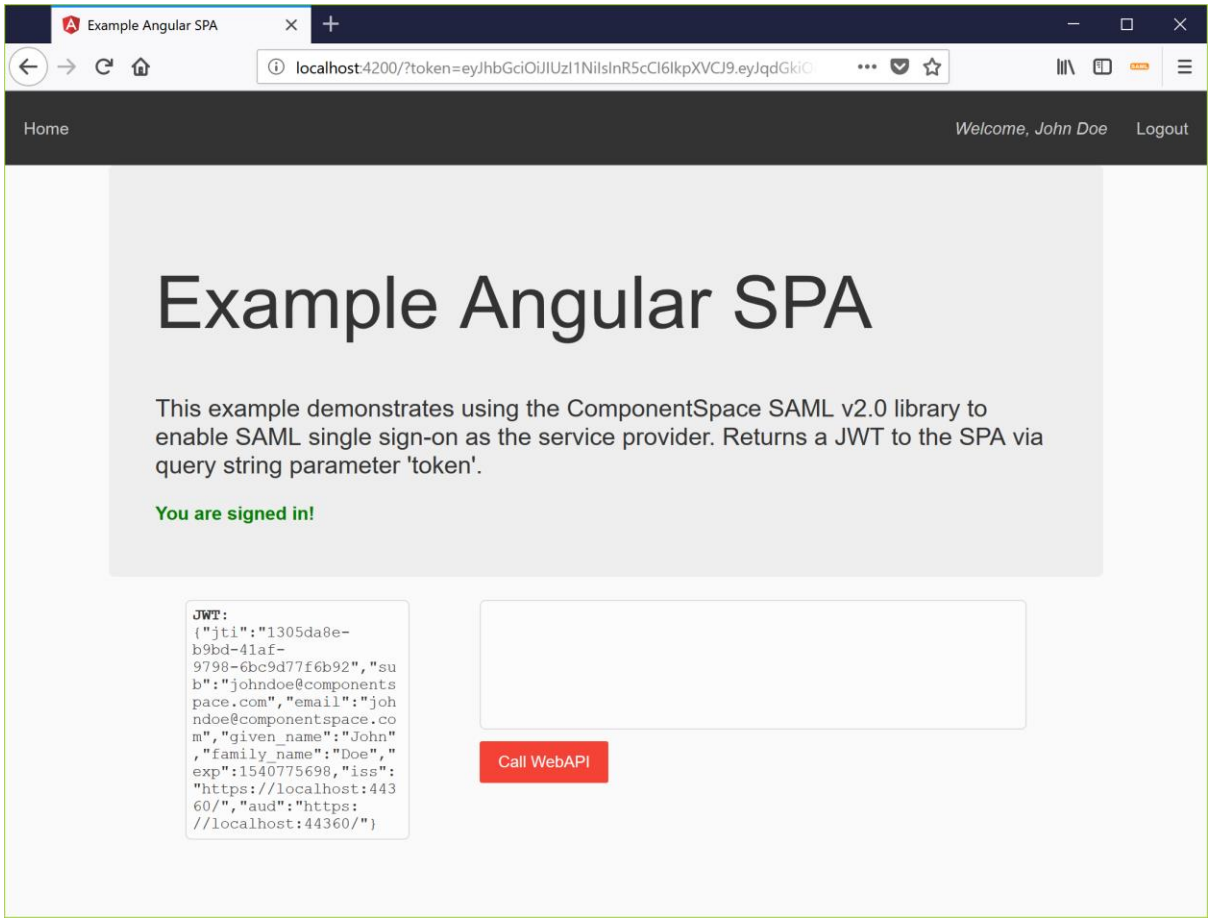
Click the Call WebAPI button.

A web API is called with the JWT bearer token and the returned SAML product information is displayed.

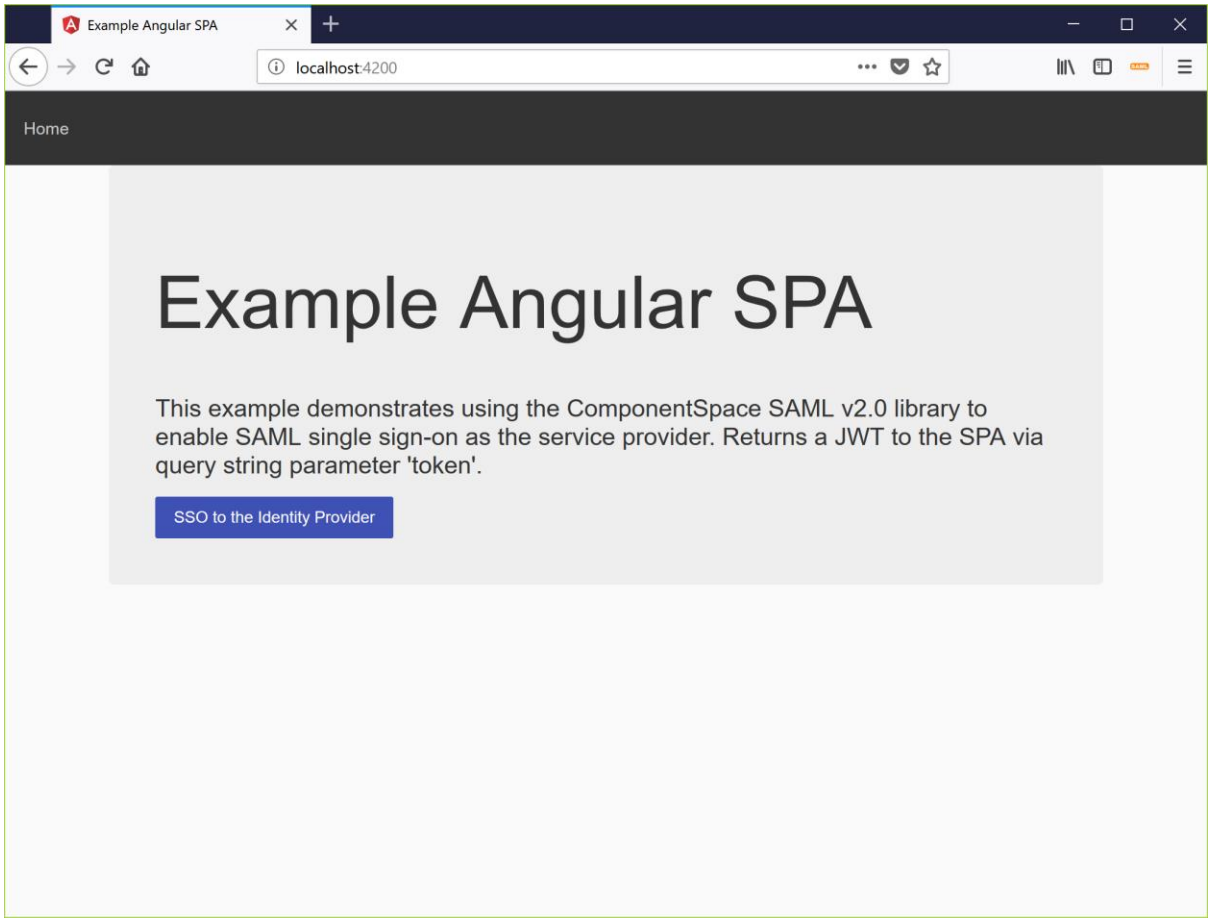


SP-initiated SLO

Having completed SSO, in the same browser window, browse to the Angular application at http://localhost:4200/.



Click the Log out link. Logout occurs at both the identity provider and service provider.



Code Walkthrough

Example Identity Provider

Configuration

The appsettings.json includes the SAML configuration.

The configuration consists of a single SAML configuration specifying a single local identity provider and multiple partner service providers.

Refer to the SAML v2.0 Configuration Guide for more information.

Startup

The ConfigureServices method includes the following code.

```
// Use a unique identity cookie name rather than sharing the cookie across applications in the domain.
services.ConfigureApplicationCookie(options =>
{
   options.Cookie.Name = "ExampleIdentityProvider.Identity";
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

To avoid the sharing of identity cookies between the ExampleIdentityProvider and ExampleServiceProvider, unique cookie names are specified.

The SAML services are added using the SAML configurations stored in the appsettings.json.

Although some of this code is specific to this example, most applications will include code to register the SAML configuration and add the SAML services.

SamlController.InitiateSingleSignOn

The SAML controller includes the following action to support IdP-initiated SSO.

```
[Authorize]
public async Task<IActionResult> InitiateSingleSignOn()
{
   // Get the name of the logged in user.
   var userName = User.Identity.Name;

   // For demonstration purposes, include some claims.
   var attributes = new List<SamlAttribute>()
   {
      new SamlAttribute(ClaimTypes.Email, User.FindFirst(ClaimTypes.Email)?.Value),
      new SamlAttribute(ClaimTypes.GivenName, User.FindFirst(ClaimTypes.GivenName)?.Value),
      new SamlAttribute(ClaimTypes.Surname, User.FindFirst(ClaimTypes.Surname)?.Value),
   };

   // Initiate single sign-on to the service provider (IdP-initiated SSO)
   // by sending a SAML response containing a SAML assertion to the SP.
   // The optional relay state normally specifies the target URL once SSO completes.
   var partnerName = _configuration["PartnerName"];
   var relayState = _configuration["RelayState"];

   await _samlIdentityProvider.InitiateSsoAsync(partnerName, userName, attributes, relayState);

   return new EmptyResult();
}
```

The authorize attribute on the method ensures only authenticated users can initiate SSO.

The user's name and some associated claims, to be included in the SAML assertion sent to the service provider, are retrieved. The user ID and attributes, if any, sent to the service provider may be different depending on your business requirements.

The partner service provider name is retrieved from the application configuration. The method for determining which service provider to select may be different depending on your business requirements.

InitiateSsoAsync is called to construct and send a SAML response to the service provider.

Control now moves to the service provider site.

Identity/Account/Logout Page

The logout page includes the following code to support IdP-initiated SLO.

```
var ssoState = await _samlIdentityProvider.GetStatusAsync();

if (await ssoState.CanSloAsync())
{
```

```
    // Request logout at the service provider(s).
    await _samlIdentityProvider.InitiateSloAsync(relayState: returnUrl);

    return new EmptyResult();
}
```

If the user clicks the log out link, a check is made to see whether the user has completed SSO and, if so, InitiateSloAsync is called to construct and send a logout request to the service provider(s).

Control now moves to the service provider site.

SamlController.SingleSignOnService

The SAML controller includes the following code to support SP-initiated SSO.

```
public async Task<IActionResult> SingleSignOnService()
{
    // Receive the authn request from the service provider (SP-initiated SSO).
    await _samlIdentityProvider.ReceiveSsoAsync();

    // If the user is logged in at the identity provider, complete SSO immediately.
    // Otherwise have the user login before completing SSO.
    if (User.Identity.IsAuthenticated)
    {
        await CompleteSsoAsync();

        return new EmptyResult();
    }
    else
    {
        return RedirectToAction("SingleSignOnServiceCompletion");
    }
}
```

ReceiveSsoAsync receives and processes the SAML authentication request from the service provider.

An internal redirect to the SingleSignOnServiceCompletion action is performed to ensure the user is logged in.

```
[Authorize]
public async Task<ActionResult> SingleSignOnServiceCompletion()
{
    await CompleteSsoAsync();

    return new EmptyResult();
}
```

The authorize attribute on the method ensures only authenticated users can respond to the SSO request.

```
private Task CompleteSsoAsync()
{
  // Get the name of the logged in user.
  var userName = User.Identity.Name;

  // For demonstration purposes, include some claims.
  var attributes = new List<SamlAttribute>()
  {
    new SamlAttribute(ClaimTypes.Email, User.FindFirst(ClaimTypes.Email)?.Value),
    new SamlAttribute(ClaimTypes.GivenName, User.FindFirst(ClaimTypes.GivenName)?.Value),
    new SamlAttribute(ClaimTypes.Surname, User.FindFirst(ClaimTypes.Surname)?.Value),
  };

  // The user is logged in at the identity provider.
  // Respond to the authn request by sending a SAML response containing a SAML assertion to the SP.
  return _samlIdentityProvider.SendSsoAsync(userName, attributes);
}
```

The user's name and some associated claims, to be included in the SAML assertion sent to the service provider, are retrieved. The user ID and attributes, if any, sent to the service provider may be different depending on your business requirements.

SendSsoAsync is called to construct and send a SAML response to the service provider.

Control now returns to the service provider site.

SamlController.SingleLogoutService

The SAML controller includes the following code to support IdP-initiated and SP-initiated SLO.

```
public async Task<IActionResult> SingleLogoutService()
{
  // Receive the single logout request or response.
  // If a request is received then single logout is being initiated by a partner service provider.
  // If a response is received then this is in response to single logout having been initiated by the identity
provider.
  var sloResult = await _samlIdentityProvider.ReceiveSloAsync();

  if (sloResult.IsResponse)
  {
    if (sloResult.HasCompleted)
    {
      // IdP-initiated SLO has completed.
      if (!string.IsNullOrEmpty(sloResult.RelayState))
      {
        return LocalRedirect(sloResult.RelayState);
      }

      return RedirectToPage("/Index");
    }
  }
  else
  {
    // Logout locally.
    await _signInManager.SignOutAsync();
```

```
        // Respond to the SP-initiated SLO request indicating successful logout.
        await _samlIdentityProvider.SendSloAsync();
    }

    return new EmptyResult();
}
```

ReceiveSloAsync is called to receive and process the logout message from the service provider.

For IdP-initiated SLO, a logout response is received. If the results indicate SLO has completed, the user is redirected to the home page.

For SP-initiated SLO, a logout request is received. The user is logged out locally and SendSloAsync is called to construct and send a SAML logout response to the service provider. Control now returns to the service provider site.

SamlController.ArtifactResolutionService

The SAML controller includes the following code to support SAML artifact resolution as part of the HTTP-Artifact binding. If the HTTP-Artifact binding is not supported, this code may be omitted.

```
public async Task<IActionResult> ArtifactResolutionService()
{
    // Resolve the HTTP artifact.
    // This is only required if supporting the HTTP-Artifact binding.
    await _samlIdentityProvider.ResolveArtifactAsync();

    return new EmptyResult();
}
```

Example Service Provider

Configuration

The appsettings.json includes the SAML configuration.

The configuration consists of a single SAML configuration specifying a single local service provider and multiple partner identity providers.

Refer to the SAML v2.0 Configuration Guide for more information.

Startup

The ConfigureServices method includes the following code.

```
// Use a unique identity cookie name rather than sharing the cookie across applications in the domain.
services.ConfigureApplicationCookie(options =>
{
    options.Cookie.Name = "ExampleServiceProvider.Identity";
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

To avoid the sharing of identity cookies between the ExampleIdentityProvider and ExampleServiceProvider, unique cookie names are specified.

The SAML services are added using the SAML configurations stored in the appsettings.json.

Although some of this code is specific to this example, most applications will include code to register the SAML configuration and add the SAML services.

SamlController.SingleSignOn

The SAML controller includes the following action to support SP-initiated SSO.

```
public async Task<IActionResult> InitiateSingleSignOn(string returnUrl = null)
{
   // To login automatically at the service provider, initiate single sign-on to the identity provider (SP-initiated SSO).
   // The return URL is remembered as SAML relay state.
   var partnerName = _configuration["PartnerName"];

   await _samlServiceProvider.InitiateSsoAsync(partnerName, returnUrl);

   return new EmptyResult();
}
```

The partner identity provider name is retrieved from the application configuration. The method for determining which identity provider to select may be different depending on your business requirements.

InitiateSSOAsync is called to construct and send a SAML authentication request to the identity provider.

Control now moves to the identity provider site.

Identity/Account/Logout Page

The logout page includes the following code to support SP-initiated SLO.

```
var ssoState = await _samlServiceProvider.GetStatusAsync();

if (await ssoState.CanSloAsync())
{
   // Request logout at the identity provider.
   await _samlServiceProvider.InitiateSloAsync(relayState: returnUrl);

   return new EmptyResult();
}
```

If the user clicks the log out link, a check is made to see whether the user has completed SSO and, if so, InitiateSloAsync is called to construct and send a logout request to the identity provider.

Control now moves to the identity provider site.

SamlController.AssertionConsumerService

The SAML controller includes the following code to support IdP-initiated and SP-initiated SSO.

```
public async Task<IActionResult> AssertionConsumerService()
{
   // Receive and process the SAML assertion contained in the SAML response.
   // The SAML response is received either as part of IdP-initiated or SP-initiated SSO.
   var ssoResult = await _samlServiceProvider.ReceiveSsoAsync();

   // Automatically provision the user.
   // If the user doesn't exist locally then create the user.
   // Automatic provisioning is an optional step.
   var user = await _userManager.FindByNameAsync(ssoResult.UserID);

   if (user == null)
   {
      user = new IdentityUser { UserName = ssoResult.UserID, Email = ssoResult.UserID };

      var result = await _userManager.CreateAsync(user);

      if (!result.Succeeded)
      {
         throw new Exception($"The user {ssoResult.UserID} couldn't be created - {result}");
      }

      // For demonstration purposes, create some additional claims.
      if (ssoResult.Attributes != null)
      {
         var samlAttribute = ssoResult.Attributes.SingleOrDefault(a => a.Name == ClaimTypes.Email);

         if (samlAttribute != null)
         {
            await _userManager.AddClaimAsync(user, new Claim(ClaimTypes.Email, samlAttribute.ToString()));
         }

         samlAttribute = ssoResult.Attributes.SingleOrDefault(a => a.Name == ClaimTypes.GivenName);

         if (samlAttribute != null)
         {
            await _userManager.AddClaimAsync(user, new Claim(ClaimTypes.GivenName,
samlAttribute.ToString()));
         }

         samlAttribute = ssoResult.Attributes.SingleOrDefault(a => a.Name == ClaimTypes.Surname);

         if (samlAttribute != null)
         {
            await _userManager.AddClaimAsync(user, new Claim(ClaimTypes.Surname,
samlAttribute.ToString()));
         }
      }
   }

   // Automatically login using the asserted identity.
   await _signInManager.SignInAsync(user, isPersistent: false);

   // Redirect to the target URL if specified.
   if (!string.IsNullOrEmpty(ssoResult.RelayState))
   {
```

```
      return LocalRedirect(ssoResult.RelayState);
   }

   return RedirectToPage("/Index");
}
```

ReceiveSsoAsync receives and processes the SAML response from the identity provider. The SAML response is either the result of IdP-initiated or SO-initiated SSO.

If the user doesn't exist in the user database, they're automatically provisioned. You may or may not support automatic provisioning depending on your business requirements.

The user is logged in automatically at the service provider using information retrieved from the SAML assertion. The user ID and attributes, if any, received by the service provider may be different depending on your business requirements.

For IdP-initiated SSO, the optional relay state sent by the identity provider specifies a URL to support deep web linking. If specified, the user is redirected to this page. Otherwise the user is redirected to the home page.

SamlController.SingleLogoutService

The SAML controller includes the following code to support IdP-initiated and SP-initiated SLO.

```
public async Task<IActionResult> SingleLogoutService()
{
   // Receive the single logout request or response.
   // If a request is received then single logout is being initiated by the identity provider.
   // If a response is received then this is in response to single logout having been initiated by the service
provider.
   var sloResult = await _samlServiceProvider.ReceiveSloAsync();

   if (sloResult.IsResponse)
   {
      // SP-initiated SLO has completed.
      if (!string.IsNullOrEmpty(sloResult.RelayState))
      {
         return LocalRedirect(sloResult.RelayState);
      }

      return RedirectToPage("/Index");
   }
   else
   {
      // Logout locally.
      await _signInManager.SignOutAsync();

      // Respond to the IdP-initiated SLO request indicating successful logout.
      await _samlServiceProvider.SendSloAsync();
   }

   return new EmptyResult();
}
```

ReceiveSloAsync is called to receive and process the logout message from the identity provider.

For SP-initiated SLO, a logout response is received. The user is redirected to the home page.

For IdP-initiated SLO, a logout request is received. The user is logged out locally and SendSloAsync is called to construct and send a SAML logout response to the identity provider. Control now returns to the identity provider site.

SamlController.ArtifactResolutionService

The SAML controller includes the following code to support SAML artifact resolution as part of the HTTP-Artifact binding. If the HTTP-Artifact binding is not supported, this code may be omitted.

```
public async Task<IActionResult> ArtifactResolutionService()
{
   // Resolve the HTTP artifact.
   // This is only required if supporting the HTTP-Artifact binding.
   await _samlIdentityProvider.ResolveArtifactAsync();

   return new EmptyResult();
}
```

ResolveArtifactAsync is called to receive and process the artifact resolve request from the identity provider.

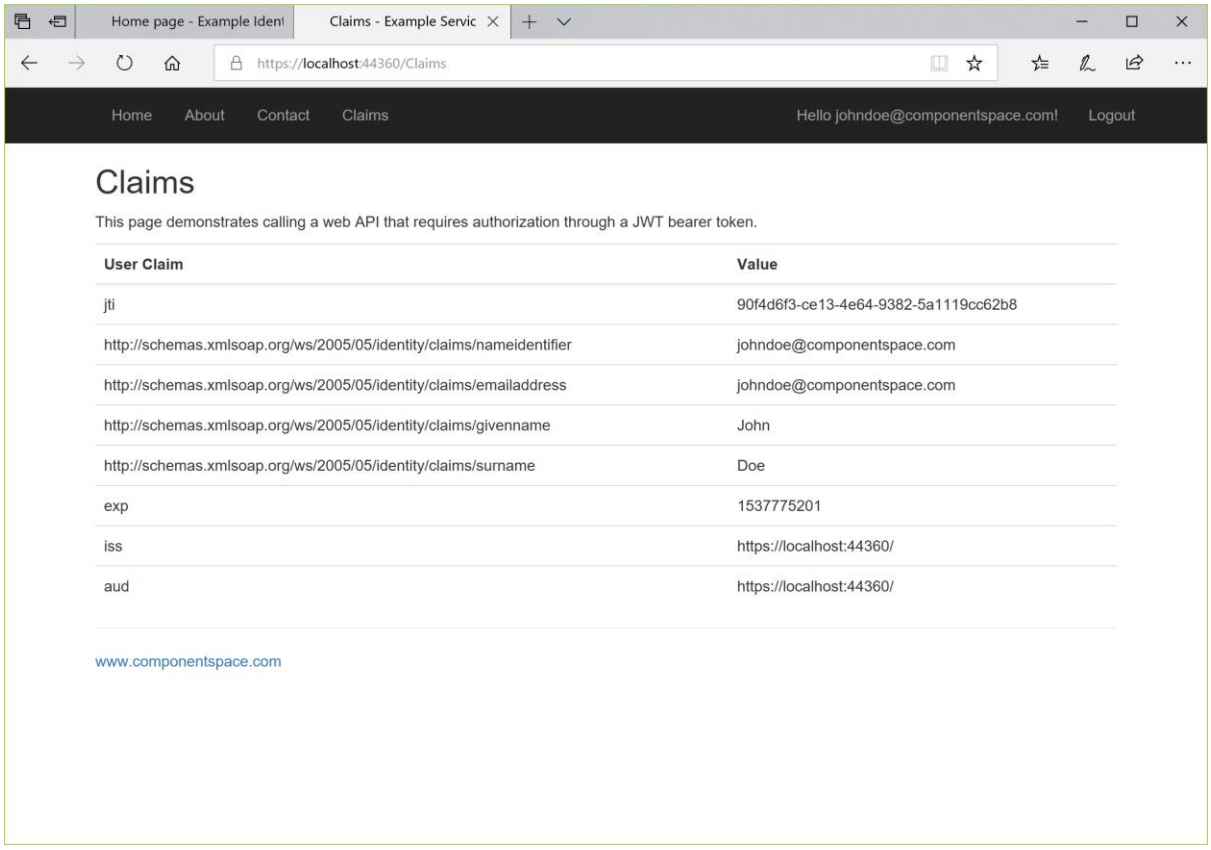
JWT Bearer Token Support

The ExampleServiceProvider project demonstrates authorizing access to a web API using JWT tokens.

The claims page calls a web API to return a JWT bearer token. This token includes claims originating from the SAML assertion received as part of SSO.

The claim page then presents the JWT bearer token when accessing a web API that returns the user's claims for display.

None of this code is SAML SSO specific but is does demonstrate using SAML SSO for authentication of the user, creating a JWT bearer token, and presenting this token for authorized access to a web API.



Middleware Identity Provider

Configuration

The appsettings.json includes the SAML configuration.

The configuration consists of a single SAML configuration specifying a single local identity provider and multiple partner service providers.

Refer to the SAML v2.0 Configuration Guide for more information.

Startup

The ConfigureServices method includes the following code.

```
// Use a unique identity cookie name rather than sharing the cookie across applications in the domain.
services.ConfigureApplicationCookie(options =>
{
   options.Cookie.Name = "MiddlewareIdentityProvider.Identity";
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));

// Add the SAML middleware services.
services.AddSamlMiddleware(options =>
{
   options.PartnerName = () => Configuration["PartnerName"];
});
```

To avoid the sharing of identity cookies between the MiddlewareIdentityProvider and ExampleServiceProvider, unique cookie names are specified.

The SAML services are added using the SAML configurations stored in the appsettings.json.

Finally, the SAML middleware is added.

The SAML options specify the name of a configured partner service provider.

The Configure method includes the following code.

```
app.UseAuthentication();

// Use SAML middleware.
app.UseSaml();
```

The authentication middleware and SAML middles are enabled.

Although some of this code is specific to this example, most applications will include code to register the SAML configuration and add the SAML services.

Index Page

The index page includes the following code to support IdP-initiated SSO.

```
public IActionResult OnGetInitiateSingleSignOn()
{
   // Pass control to the SAML middleware for IdP-initiated SSO.
   return LocalRedirect(SamlMiddlewareDefaults.InitiateSingleSignOnPath);
}
```

A redirect to the InitiateSingleSignOnPath initiates SSO.

The SAML middleware constructs a SAML response and sends it to the service provider.

Control now moves to the service provider site.

Identity/Account/Logout Page

The logout page includes the following code to support SP-initiated SLO.

```
public async Task<IActionResult> OnGet(string returnUrl = null)
{
   // If a redirect URL is included, this was invoked by the SAML middleware as part of SP-initiated SLO.
   if (returnUrl != null)
   {
      // Logout the user locally.
      await _signInManager.SignOutAsync();
      _logger.LogInformation("User logged out.");

      // Return control to the SAML middleware.
      return LocalRedirect(returnUrl);
   }

   return Page();
```

```
}
```

The user is logged out locally and control is returned to the SAML middleware to complete SLO.

Control now moves to the service provider site.

The logout page includes the following code to support IdP-initiated SLO.

```
public async Task<IActionResult> OnPost(string returnUrl = null)
{
   // Logout the user locally.
   await _signInManager.SignOutAsync();
   _logger.LogInformation("User logged out.");

   // Pass control to the SAML middleware for IdP-initiated SLO.
   return LocalRedirect(
      QueryHelpers.AddQueryString(
         SamlMiddlewareDefaults.InitiateSingleLogoutPath,
         SamlMiddlewareDefaults.ReturnUrlParameter,
         returnUrl));
}
```

A redirect to the InitiateSingleLogoutPath initiates SLO.

The SAML middleware constructs a SAML logout request and sends it to the service provider.

Control now moves to the service provider site.

Middleware Service Provider

Configuration

The appsettings.json includes the SAML configuration.

The configuration consists of a single SAML configuration specifying a single local service provider and multiple partner identity providers.

Refer to the SAML v2.0 Configuration Guide for more information.

Startup

The ConfigureServices method includes the following code.

```
// Use a unique identity cookie name rather than sharing the cookie across applications in the domain.
services.ConfigureApplicationCookie(options =>
{
   options.Cookie.Name = "MiddlewareServiceProvider.Identity";
});

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));

// Add SAML authentication middleware.
services.AddAuthentication().AddSaml(options =>
{
```

```
    options.PartnerName = () => Configuration["PartnerName"];
});
```

To avoid the sharing of identity cookies between the ExampleIdentityProvider and MiddlewareServiceProvider, unique cookie names are specified.

The SAML services are added using the SAML configurations stored in the appsettings.json.

Finally, the SAML authentication is added.

The SAML options specify the name of a configured partner identity provider.

Although some of this code is specific to this example, most applications will include code to register the SAML configuration and add the SAML services and middleware.

Identity/Account/Logout

The logout page includes the following code to support SP-initiated SLO.

```
public async Task<IActionResult> OnPost(string returnUrl = null)
{
    // Logout the user locally.
    await _signInManager.SignOutAsync();
    _logger.LogInformation("User logged out.");

    // Explicitly logout SAML as this isn't done by the SignInManager.
    await HttpContext.SignOutAsync(
        SamlAuthenticationDefaults.AuthenticationScheme,
        new AuthenticationProperties()
        {
            RedirectUri = returnUrl
        });

    return new EmptyResult();
}
```

HttpContext.SignOutAsync initiates SLO.

The SAML authentication handler constructs a SAML logout request and sends it to the identity provider.

Control now moves to the identity provider site.

Example Web API

Configuration

The appsettings.json includes the SAML configuration.

The configuration consists of a single SAML configuration specifying a single local service provider and multiple partner identity providers.

Refer to the SAML v2.0 Configuration Guide for more information.

Startup

The ConfigureServices method includes the following code.

```
// Optionally add support for JWT bearer tokens.
// This is required only if JWT bearer tokens are used to authorize access to a web API.
// It's not required for SAML SSO.
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
   .AddJwtBearer(options =>
   {
      options.TokenValidationParameters = new TokenValidationParameters()
      {
         ValidateIssuer = true,
         ValidateAudience = true,
         ValidateLifetime = true,
         ValidateIssuerSigningKey = true,
         ValidIssuer = Configuration["JWT:Issuer"],
         ValidAudience = Configuration["JWT:Issuer"],
         IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["JWT:Key"]))
      };
   });

// Add SAML SSO services.
services.AddSaml(Configuration.GetSection("SAML"));
```

JWT bearer token support is configured.

The SAML services are added using the SAML configurations stored in the appsettings.json.

Although some of this code is specific to this example, most applications will include code to register the SAML configuration and add the SAML services.

SamlController.InitiateSingleSignOn

The SAML controller includes the following action to support SP-initiated SSO.

```
public async Task<IActionResult> InitiateSingleSignOn(string returnUrl)
{
   // To login automatically at the service provider,
   // initiate single sign-on to the identity provider (SP-initiated SSO).
   var partnerName = _configuration["PartnerName"];

   await _samlServiceProvider.InitiateSsoAsync(partnerName, returnUrl);

   return new EmptyResult();
}
```

The partner identity provider name is retrieved from the application configuration. The method for determining which identity provider to select may be different depending on your business requirements.

InitiateSSOAsync is called to construct and send a SAML authentication request to the identity provider.

Control now moves to the identity provider site.

SamlController.InitiateSingleLogout

The SAML controller includes the following action to support SP-initiated SLO.

```
public async Task<IActionResult> InitiateSingleLogout(string returnUrl)
{
   var ssoState = await _samlServiceProvider.GetStatusAsync();

   if (await ssoState.CanSloAsync())
   {
      // Request logout at the identity provider.
      await _samlServiceProvider.InitiateSloAsync(relayState: returnUrl);

      return new EmptyResult();
   }

   return Redirect(returnUrl);
}
```

A check is made to see whether the user has completed SSO and, if so, InitiateSloAsync is called to construct and send a logout request to the identity provider.

Control now moves to the identity provider site.

SamlController.AssertionConsumerService

The SAML controller includes the following code to support IdP-initiated and SP-initiated SSO.

```
public async Task<IActionResult> AssertionConsumerService()
{
   // Receive and process the SAML assertion contained in the SAML response.
   // The SAML response is received either as part of IdP-initiated or SP-initiated SSO.
   var ssoResult = await _samlServiceProvider.ReceiveSsoAsync();

   // Generate a JWT from the SAML assertion.
   var jwtSecurityToken = CreateJwtSecurityToken(ssoResult);

   // Return the JWT.
   return Redirect(QueryHelpers.AddQueryString(ssoResult.RelayState, "token", new
JwtSecurityTokenHandler().WriteToken(jwtSecurityToken)));
}
```

ReceiveSsoAsync receives and processes the SAML response from the identity provider.

A JWT security token is generated and returned to the application that initiated SSO.

In this example, the token is returned as a query string parameter. However, any suitable mechanism may be used.

SamlController.SingleLogoutService

The SAML controller includes the following code to support IdP-initiated and SP-initiated SLO.

```
public async Task<IActionResult> SingleLogoutService()
{
   // Receive the single logout request or response.
   // If a request is received then single logout is being initiated by the identity provider.
```

```
    // If a response is received then this is in response to single logout having been initiated by the service
provider.
    var sloResult = await _samlServiceProvider.ReceiveSloAsync();

    if (sloResult.IsResponse)
    {
      // SP-initiated SLO has completed.
      return Redirect(sloResult.RelayState);
    }
    else
    {
      // Respond to the IdP-initiated SLO request indicating successful logout.
      await _samlServiceProvider.SendSloAsync();
    }

    return new EmptyResult();
}
```

ReceiveSloAsync is called to receive and process the logout message from the identity provider.

For SP-initiated SLO, a logout response is received. The user is redirected to the application that initiated SLO.

For IdP-initiated SLO, a logout request is received. The user is logged out locally and SendSloAsync is called to construct and send a SAML logout response to the identity provider. Control now returns to the identity provider site.

Example Angular SPA

SAML SSO is initiated by sending an HTTP Get to the example web API (e.g. https://localhost:44319/Saml/InitiateSingleSignOn?returnurl=http://localhost:4200).

The returnUrl query string parameter specifies where control should return to once SSO completes.

Similarly, SLO is initiated by sending an HTTP Get to the example web API (e.g. https://localhost:44319/Saml/InitiateSingleLogout?returnurl=http://localhost:4200).

The returnUrl query string parameter specifies where control should return to once SLO completes.

SAML SSO and SLO flows cannot be initiated through web API calls as these flows must be through the browser, as required by the SAML protocol, and typically involve prompting the user to login or logoff.

Configuration

The environment specifies the ExampleWebApi URLs for initiating SSO, initiating SLO and invoking the web API.

```
export const environment = {
  production: false,
  samlSsoUrl: 'https://localhost:44319/Saml/InitiateSingleSignOn?returnurl=http://localhost:4200',
  samlSloUrl: 'https://localhost:44319/Saml/InitiateSingleLogout?returnurl=http://localhost:4200',
  apiUrl: 'https://localhost:44319/api',
};
```

AppComponent

The AppComponent includes the following code to extract and save the JWT.

In this example, the token is saved in memory and in local storage. However, any suitable storage may be used.

```
constructor(private http: HttpClient, private router: Router) {

  // Clear an existing token
  localStorage.removeItem('token');

  // Subscribe to router events, capture token at ActivationEnd
  this.router.events.subscribe(event => {

    if (event instanceof ActivationEnd) {

      this.token = event.snapshot.queryParamMap.get('token');

      if (this.token) {
        localStorage.setItem('token', this.token);
        this.isSignedIn = true;

        // Decode the token, assign full name property
        const helper = new JwtHelperService();
        const decodedToken = helper.decodeToken(this.token);
        this.fullName = `${decodedToken.given_name} ${decodedToken.family_name}`;
        this.jwtString = JSON.stringify(decodedToken);

      } else {
        this.isSignedIn = false;
      }
    }

  });

}
```

It also includes the following code to invoke the web API including a JWT bearer token.

```
onClick() {

  this.isSpinnerVisible = true;

  const url = `${environment.apiUrl}/samllicense`;

  this.apiResult = `Calling api "${url}" with JWT...`;

  setTimeout( () => {

    // Add authorization header with the JWT
    const headers = new HttpHeaders({
      'Authorization': 'Bearer ' + this.token
    });
```

```
    // Include header in the http.get options
    this.http.get<any>(url, { headers: headers } )
      .subscribe(
        data => {
          this.isSpinnerVisible = false;
          this.apiResult = data.displayMessage;
        },
        err => this.apiResult = err.message
      );
  }
  , 2000);
}
```

Error Handling

As these are example applications, no error handling is included. Exceptions are not caught and therefore are displayed in the browser.

In a production application, exceptions should be caught and processed.

Refer to the SAML v2.0 Developer Guide for more information.

Running the Examples on IIS

Connection String

Update the connection string in appsettings.json to specify a database server such as SQL Server.

The following is one example of a connection string to SQL Server running on localhost and using Windows authentication.

```
"DefaultConnection": "Server=localhost;Database=aspnet-
ExampleIdentityProvider;Trusted_Connection=True;MultipleActiveResultSets=true"
```

Database Creation

Refer to the following tutorial for information on creating the database.

https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/migrations

From the command line, change to the project's folder and run the following commands.

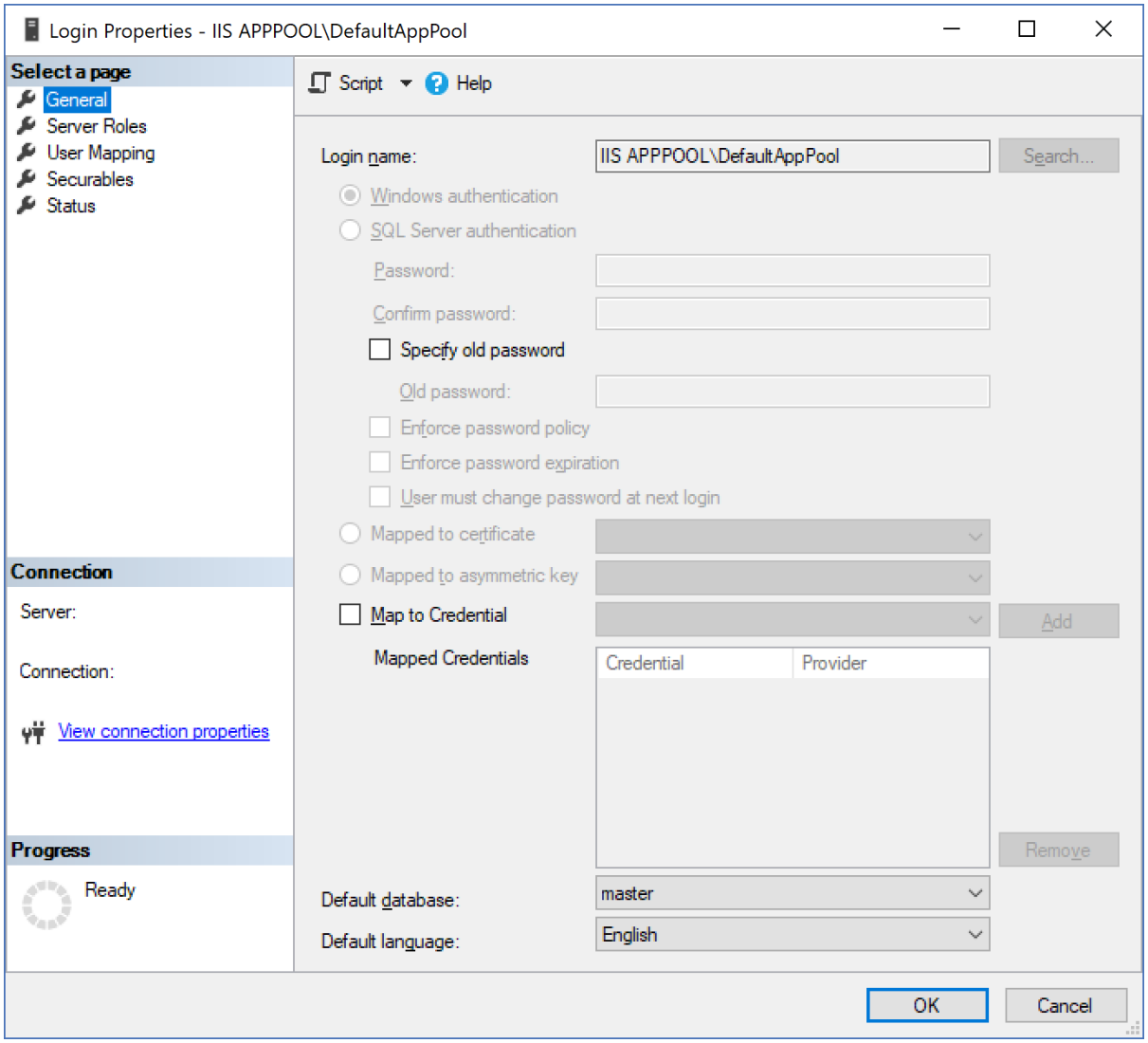
```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

Database Permissions

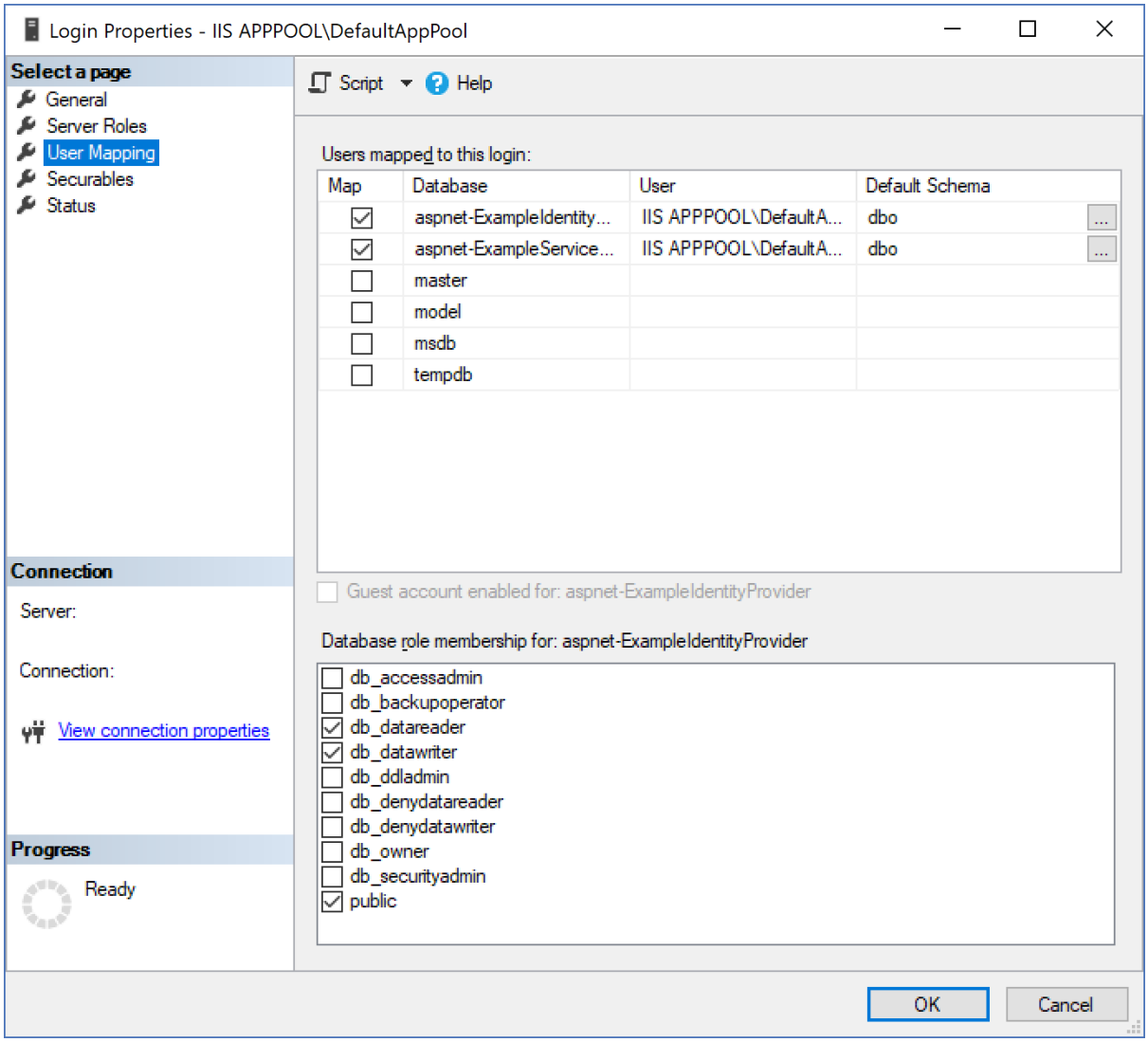
The account under which the application runs must have permission to access the database.

For a SQL Server database and if Trusted_Connection is specified in the connection string, use the Microsoft SQL Server Management Studio and create a new login by selecting the Security > Logins node and clicking New Login.

In the example below, IIS APPPOOL\DefaultAppPool is specified as this is the account under which the application runs in IIS.

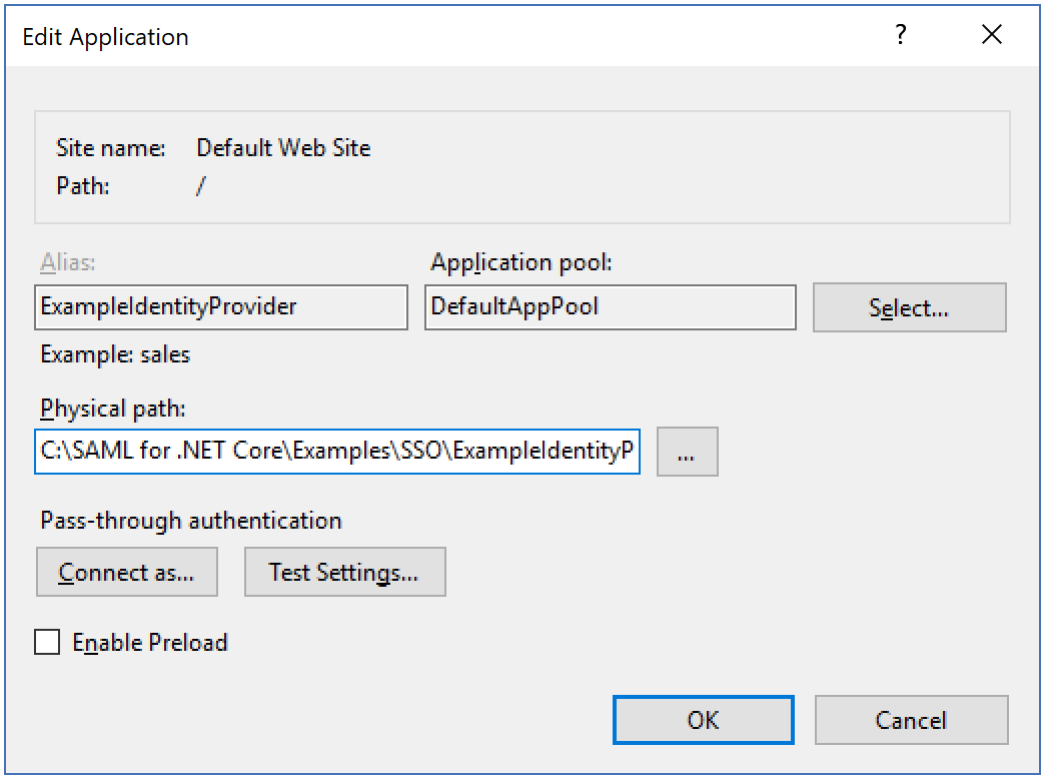


Ensure the account has read and write access to the database.



IIS Publication

Publish the application to IIS.



Update SAML Configuration

Ensure all URLs in the SAML configuration are updated as required.