How to migrate PowerServer projects from the discontinued OAuth server authorization

1 Overview

The "Built-in OAuth server" auth template uses IdentityServer4 but IdentityServer4's support ends at the end of the year 2022. For this reason, starting from PowerServer 2022 R2, the "Built-in OAuth server" auth template will no longer be available. This article suggested three migration paths for you to take if you have implemented the "Built-in OAuth server" authorization in your project. For more authorization guidance, please refer to the tutorial https://docs.appeon.com/ps2022/Authenticating_your_apps.html.

2 (Recommended) Migration Path #1: Switching to the "Use built-in JWT server" auth template

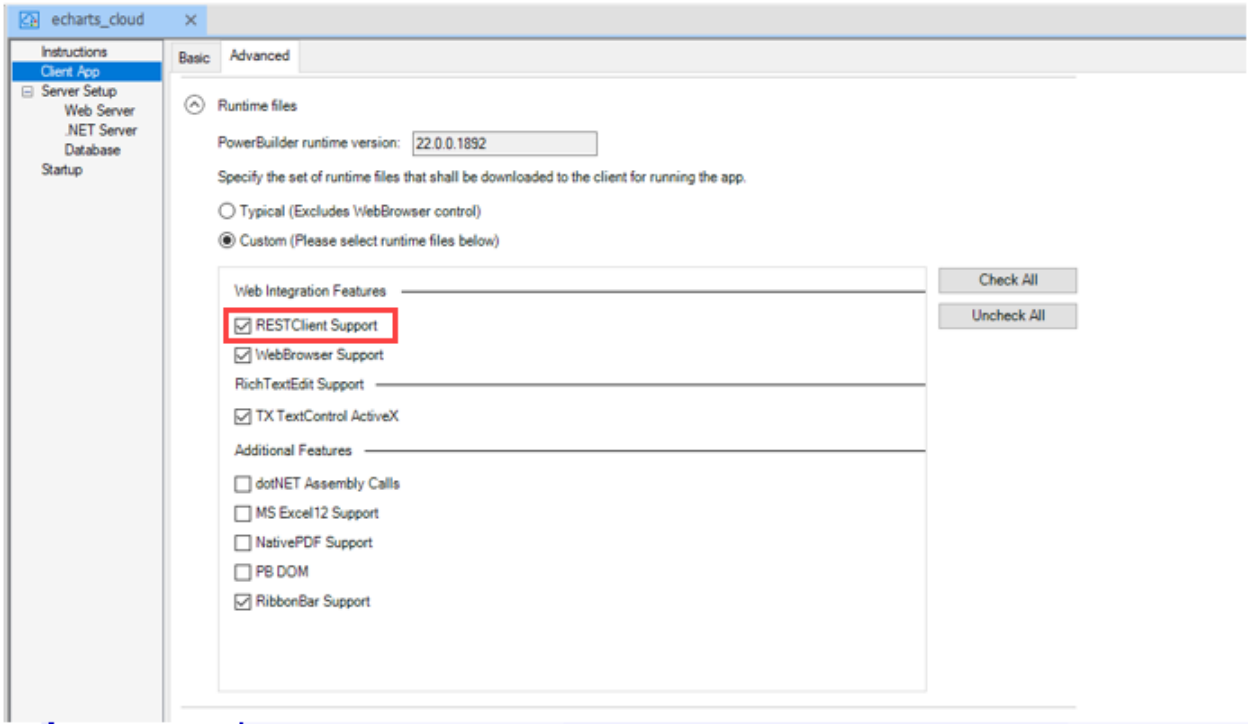
The steps provided in this section assumes that you have properly implemented the "Built-in OAuth server" authorization in your project, and covers the minimum changes you need to make when switching to the "Use built-in JWT server".

2.1 Updating the f_Authorization

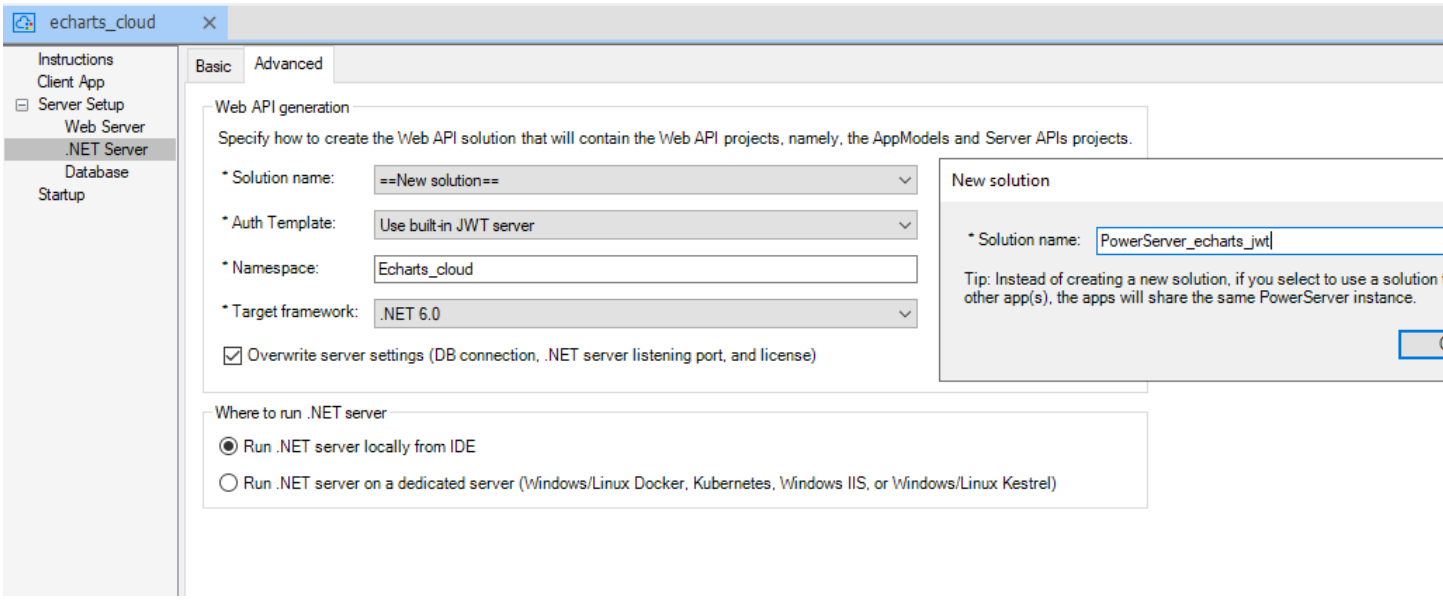
The f_Authorization scripts for JWT authorization are different from the scripts for OAuth. Please copy the scripts provided at https://docs.appeon.com/ps2022/JWT_client_app.html#Add_scripts_for_JWT to replace the scripts in the f_Authorization function.

2.2 Updating the PowerServer project configuration

a. Make sure to select the "RESTClient Support" option in the Client App > Runtime files settings.



b. Create a new solution, and select the “Use built-in JWT server” as the auth template.



2.3 Updating the scripts in the C# solution

After changes in the above two sections, please build and deploy the PowerServer project.

When the username and password are passed from the application to the built-in JWT server, the JWT server will by default authenticate them against users defined in the DefaultUserStore.cs file. If you want to authenticate users against an authentication database, follow the instructions at https://docs.appeon.com/ps2022/Appendix_for_JWT.html#Validate_against_database_for_JWT to modify the Authentication\JWT\Impl\DatabaseUserStore.cs file in the generated C# solution.

3 Migration Path #2: Switching to the "Use external Azure Active Directory service" auth template

The steps provided in this section assumes that you have properly implemented the "Built-in OAuth server" authorization in your project, and covers the minimum changes you need to make when switching to the "Use external Azure Active Directory service".

3.1 Updating the f_Authorization

The f_Authorization scripts for Azure AD authorization are different from the scripts for OAuth. Please copy the scripts provided at https://docs.appeon.com/ps2022/Azure_Active_Directory.html#Add_scripts_for_Azure_AD to replace the scripts in the f_Authorization function.

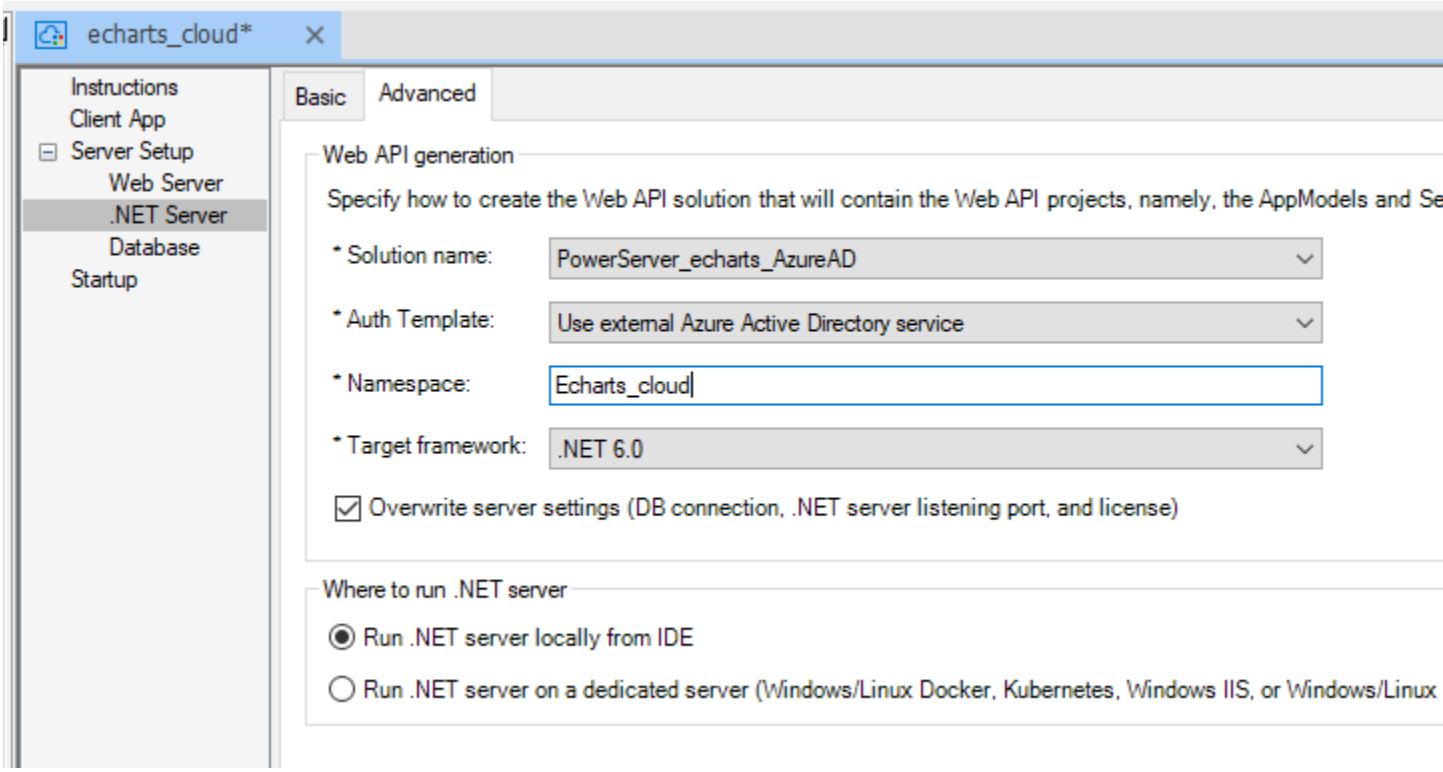
3.2 Updating the CloudSetting.ini file

Locate the CloudSetting.ini file that you have created for OAuth in the same location as the PBT file, and change the content to the following:

```
[Setup]
TokenURL=https://login.microsoftonline.com/0ffb9ae0-c080-4913-aa94-
ed08b5de4d40/oauth2/v2.0/token
```

3.3 Updating the PowerServer project configuration

In the .NET Server -> Advanced tab, create a new solution, and select the "Use external Azure Active Directory service" as the auth template.



3.4 Updating the scripts in the C# solution

After changes in the above two sections, please build and deploy the PowerServer project.

Then, follow the instructions at https://docs.appeon.com/ps2022/Azure_Active_Directory.html#Modifying_the_authentication_template_for_Azure_AD to modify the authentication template in the Authentication.json file in the C# solution. It is because the Azure AD server address must be provided so that the PowerServer Web APIs can use it to validate the token passed from the client.

4 Migration Path #3: Switching to use Duende.IdentityServer for OAuth

Important: Duede.Identity requires commercial license for production use. Please check their license agreement before continuing.

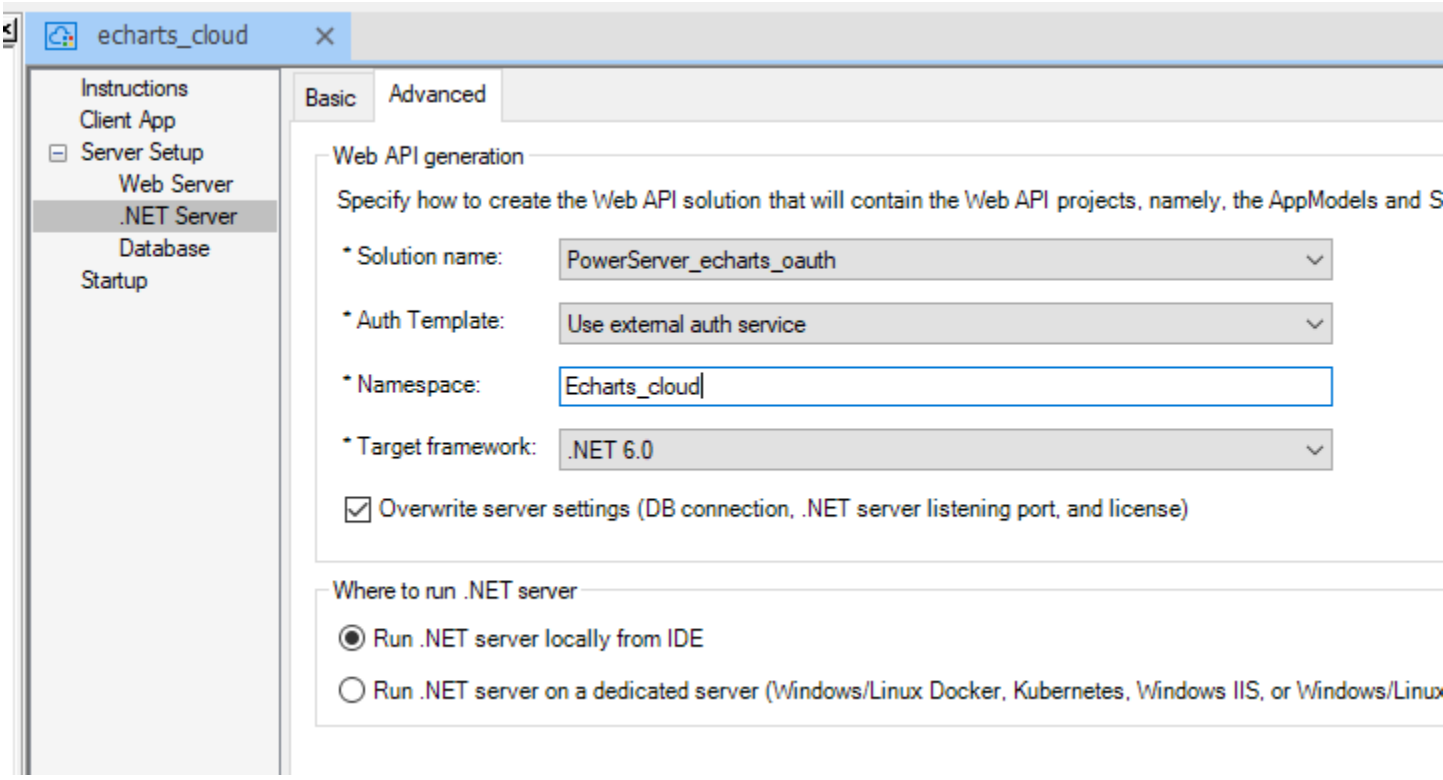
Before you start, be aware of the equivalent Duende Nuget package for each of the IdentityServer4 package.

IdentityServer4 Package	Equivalent Duende Package
IdentityServer4	Duende.IdentityServer
IdentityServer4.AspNetIdentity	Duende.IdentityServer.AspNetIdentity

IdentityServer4.EntityFramework	Duende.IdentityServer.EntityFramework
IdentityServer4.Storage	Duende.IdentityServer.Storage

4.1 Updating the PowerServer project configuration

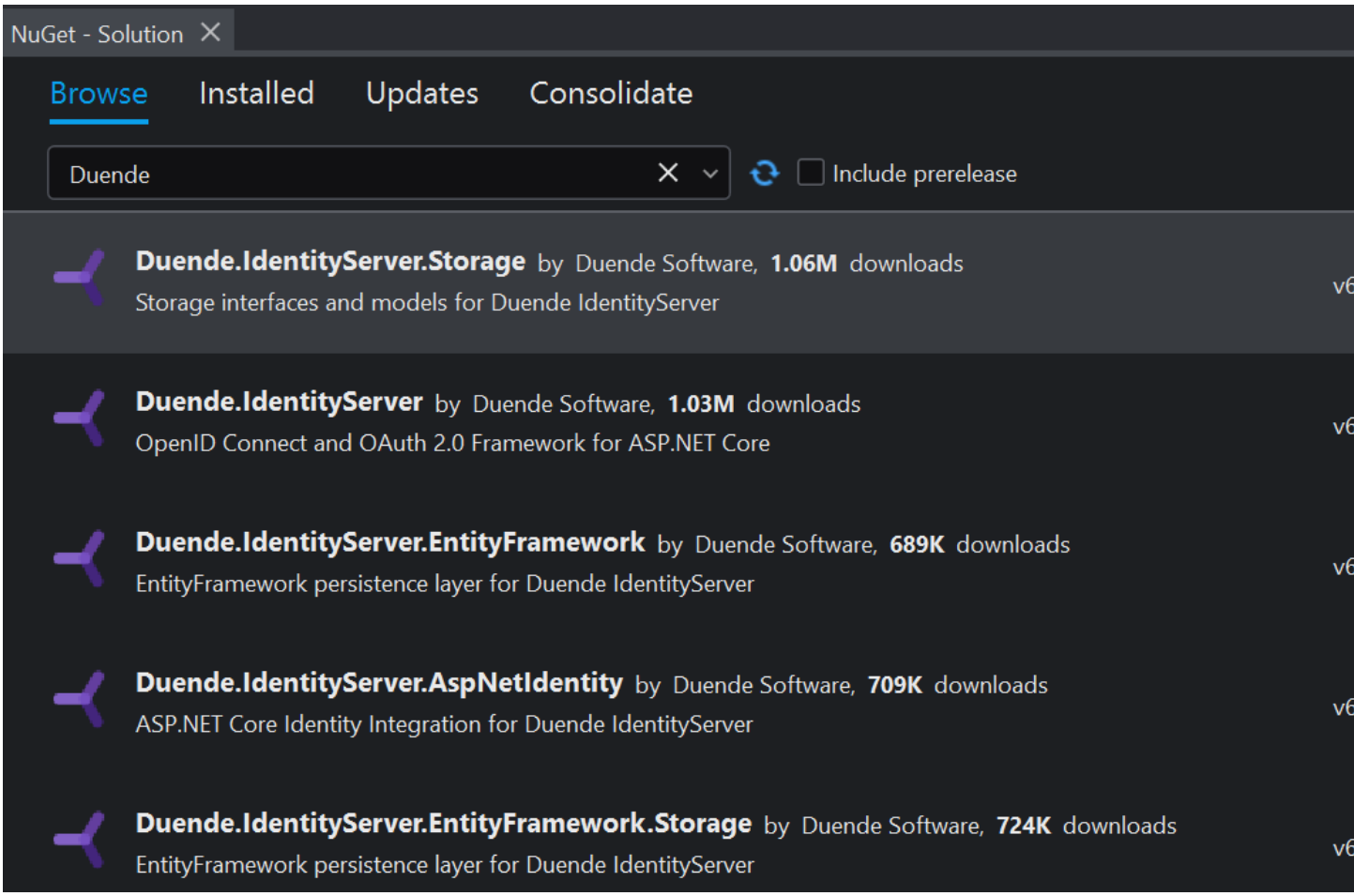
In the .NET Server -> Advanced tab, create a new solution, and select the “Use external auth service” as the auth template.



4.2 Installing Duende.IdentityServer packages to work with the C# solution

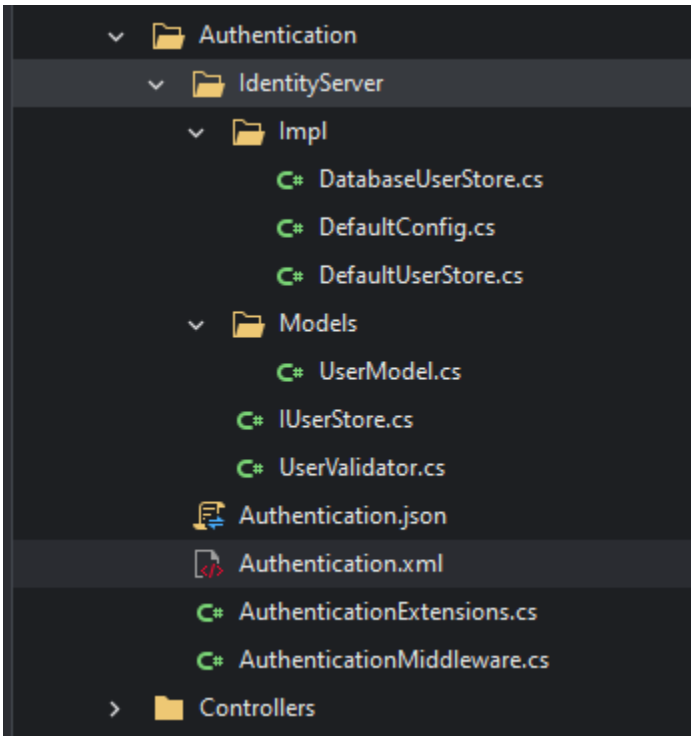
After changes in the above two sections, please build and deploy the PowerServer project. Then, in the generated C# solution, these Duende.IdentityServer packages shall be installed:

- Duende.IdentityServer
- Duende.IdentityServer.AspNetIdentity
- Duende.IdentityServer.EntityFramework
- Duende.IdentityServer.Storage



4.3 Adding auth template files to the C# solution

The following screenshot shows the auth template files that shall be added to the C# solution:



6.1.4 UserModel.cs

UserModel.cs

```
using System;
using System.Collections.Generic;
using System.Security.Claims;

namespace ServerAPIs
{
   public class UserModel
   {
      // User name, it shall be unique for each user
      public string Username { get; set; } = String.Empty;

      // User password
      public string Password { get; set; } = String.Empty;

      // User claim, will be included in id_token
      public IEnumerable<Claim> Claims { get; set; } = new List<Claim>();
   }
}
```

6.1.5 IUserStore.cs

IUserStore.cs

```
namespace ServerAPIs
{
   public interface IUserStore
   {
      // Returns the user info after the user is validated successfully
      UserModel ValidateCredentials(string username, string password);
   }
}
```

6.1.6 UserValidator.cs

UserValidator.cs

```
using System.Threading.Tasks;
using IdentityModel;
using Duende.IdentityServer.Models;
using Duende.IdentityServer.Validation;

namespace ServerAPIs
{
   // Defines the IdentityServer user validator, will be called when validate users using
ResourceOwnerPassword
   public class UserValidator : IResourceOwnerPasswordValidator
   {
      private readonly IUserStore _users;

      public UserValidator(IUserStore users)
      {
         _users = users;
      }

      public Task ValidateAsync(ResourceOwnerPasswordValidationContext context)
      {
         // Calls IUserStore to validate the user
         if (_users.ValidateCredentials(context.UserName, context.Password) is UserModel user)
         {
            // When the validation is successful, returns the user information
            context.Result = new GrantValidationResult(
               user.Username, OidcConstants.AuthenticationMethods.Password, user.Claims);
         }
         else
         {
            // When the validation fails, returns the error
            context.Result = new GrantValidationResult(
               TokenRequestErrors.InvalidGrant, "Incorrect username or password.");
         }
```

```
            return Task.CompletedTask;
        }
    }
}
```

6.1.7 DatabaseUserStore.cs

DatabaseUserStore.cs

```
using Microsoft.Extensions.Logging;

namespace ServerAPIs
{
    public class DatabaseUserStore : IUserStore
    {
        private readonly ILogger _logger;

        public DatabaseUserStore(ILogger<DatabaseUserStore> logger)
        {
            _logger = logger;
        }

        public UserModel ValidateCredentials(string username, string password)
        {
            // Adds code here to get user by user name and password from the database
            UserModel user = null;

            if (user != null)
            {
                _logger.LogInformation($"User <{username}> logged in.");

                return user;
            }
            else
            {
                _logger.LogError($"Invalid login attempt.");

                return default;
            }
        }
    }
}
```

6.1.8 DefaultConfig.cs

DefaultConfig.cs

```
using System.Collections.Generic;
using Duende.IdentityServer.Models;

namespace ServerAPIs
{
   public static class DefaultConfig
   {
      // Configures the user identity information in IdentityServer4
      // To be used by UserInfo and included in the returned IdToken
      // see https://identityserver4.readthedocs.io/en/latest/reference/identity_resource.html
      public static IEnumerable<IdentityResource> IdentityResources => new IdentityResource[]
      {
         new IdentityResources.OpenId(),
         new IdentityResources.Profile(),
         new IdentityResources.Email(),
      };

      // Configures ApiScope data
      // see https://identityserver4.readthedocs.io/en/latest/reference/api_scope.html
      public static IEnumerable<ApiScope> ApiScopes => new ApiScope[]
      {
         new ApiScope("serverapi", "PowerServer Api"),
      };

      // Configures the authentication client data
      // see https://identityserver4.readthedocs.io/en/latest/reference/client.html
      public static IEnumerable<Client> Clients => new Client[]
      {
         // client credentials flow client
         new Client
         {
            ClientName = "Client Credentials Client",
            // Authentication method used by the client
            // see https://identityserver4.readthedocs.io/en/latest/topics/grant_types.html
            AllowedGrantTypes = GrantTypes.ClientCredentials,

            // Client ID, shall be unique for each client
            ClientId = "client",
            // Client password, to make sure the client is valid when authenticated
            ClientSecrets = { new Secret("511536EF-F270-4058-80CA-1C89C192F69A".Sha256()) },

            // Allows the client to specify the scope, and the scope will be included in access_token
            AllowedScopes = { "serverapi" },
```

```
            // Allows the token refresh mechanism
            // see https://identityserver4.readthedocs.io/en/latest/topics/refresh_tokens.html
            AllowOfflineAccess = true,
        },

        // resource owner password grant client
        new Client
        {
            ClientName = "Resource Owner Password Client",
            // Authentication method used by the client
            // see https://identityserver4.readthedocs.io/en/latest/topics/grant_types.html
            AllowedGrantTypes = GrantTypes.ResourceOwnerPassword,

            // Client ID, shall be unique for each client
            ClientId = "ro.client",
            // Client password, to make sure the client is valid when authenticated
            ClientSecrets = { new Secret("08692CED-944D-4DA9-BFEF-0FE503C203AC".Sha256()) },

            // Allows the client to specify the scope, and the scope will be included in access_token
            AllowedScopes = { "openid", "profile", "serverapi" },

            // Allows the token refresh mechanism
            // see https://identityserver4.readthedocs.io/en/latest/topics/refresh_tokens.html
            AllowOfflineAccess = true,
        },
    };
  }
}
```

6.1.9 DefaultUserStore.cs

DefaultUserStore.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Text.Json;
using IdentityModel;
using Duende.IdentityServer;
using Microsoft.Extensions.Logging;

namespace ServerAPIs
{
  // Default IUserStore implementation
  public class DefaultUserStore : IUserStore
```

```
  {
    private readonly ILogger _logger;
    private readonly List<UserModel> _users;

    public DefaultUserStore(ILogger<DefaultUserStore> logger)
    {
      _logger = logger;

      var address = JsonSerializer.Serialize(new
      {
        country = "Germany",
        locality = "Heidelberg",
        street_address = "One Hacker Way",
        postal_code = 69118,
      });

      _users = new List<UserModel>
      {
        new UserModel
        {
          Username = "alice",
          Password = "alice",
          Claims = new[]
          {
            new Claim(JwtClaimTypes.Name, "Alice Smith"),
            new Claim(JwtClaimTypes.GivenName, "Alice"),
            new Claim(JwtClaimTypes.FamilyName, "Smith"),
            new Claim(JwtClaimTypes.WebSite, "http://alice.com"),
            new Claim(JwtClaimTypes.Email, "AliceSmith@email.com"),
            new Claim(JwtClaimTypes.EmailVerified, "true", ClaimValueTypes.Boolean),
            new Claim(JwtClaimTypes.Address, address,
IdentityServerConstants.ClaimValueTypes.Json)
          }
        },
        new UserModel
        {
          Username = "bob",
          Password = "bob",
          Claims = new[]
          {
            new Claim(JwtClaimTypes.Name, "Bob Smith"),
            new Claim(JwtClaimTypes.GivenName, "Bob"),
            new Claim(JwtClaimTypes.FamilyName, "Smith"),
            new Claim(JwtClaimTypes.WebSite, "http://bob.com"),
            new Claim(JwtClaimTypes.Email, "BobSmith@email.com"),
            new Claim(JwtClaimTypes.EmailVerified, "true", ClaimValueTypes.Boolean),
```

```
                    new Claim(JwtClaimTypes.Address, address,
IdentityServerConstants.ClaimValueTypes.Json)
                }
            }
        };
    }

    public UserModel ValidateCredentials(string username, string password)
    {
        // Logs in with the default user in the script
        // If you want to integrate an existing user management system (such as database), uses
DatabaseUserStore instead
        var user = _users.FirstOrDefault(x => x.Username == username && x.Password == password);

        if (user != null)
        {
            // Logs the authentication-success information
            _logger.LogInformation($"User <{username}> logged in.");

            return user;
        }
        else
        {
            // Logs the authentication-failed information
            _logger.LogError($"Invalid login attempt.");

            return default;
        }
    }
  }
}
```

6.1.10 AuthenticationMiddleware.cs

AuthenticationMiddleware.cs

```
using System;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;

namespace ServerAPIs
{
  public class AuthenticationMiddleware : IStartupFilter
  {
    //  To be called before Startup.ConfigureServices, for configuring the HTTP request pipeline.
```

```
    // To easily switch between templates, see https://docs.microsoft.com/en-
us/aspnet/core/fundamentals/startup#extend-startup-with-startup-filters
    public Action<IApplicationBuilder> Configure(Action<IApplicationBuilder> next)
    {
      return builder =>
      {
        // Injects the IdentityServer endpoint
        builder.UseIdentityServer();

        next(builder);
      };
    }
  }
}
```

6.1.11 AuthenticationExtensions.cs

AuthenticationExtensions.cs

```
using System;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.IdentityModel.Logging;
using PowerServer.Core;

namespace ServerAPIs
{
  public static class AuthenticationExtensions
  {
    // To be called in Startup.ConfigureServices, for adding services related with the authentication
module
    public static IServiceCollection AddPowerServerAuthentication(
      this IServiceCollection services, IConfiguration configuration)
    {
      // Uses IdentityServer as the built-in authentication server
      // Injects the IdentityServer services (request handler, validation, and storage)
      var builder = services.AddIdentityServer(options =>
      {
        options.LicenseKey = ""; //your license key
        options.Events.RaiseSuccessEvents = true;
        options.Events.RaiseFailureEvents = true;
        options.Events.RaiseInformationEvents = true;
        options.Events.RaiseErrorEvents = true;
        options.EmitStaticAudienceClaim = true;
```

```
        });

        builder.AddInMemoryIdentityResources(DefaultConfig.IdentityResources);
        builder.AddInMemoryApiScopes(DefaultConfig.ApiScopes);
        builder.AddInMemoryClients(DefaultConfig.Clients);
        builder.Services.AddSingleton<IUserStore, DefaultUserStore>();
        builder.AddResourceOwnerValidator<UserValidator>();
        builder.AddDeveloperSigningCredential();
        builder.Services.AddTransient<IStartupFilter, AuthenticationMiddleware>();

        // Adds the PowerServer PowerServer authentication configuration
        // see https://docs.microsoft.com/en-us/aspnet/core/security/authentication
        // Adds PowerServer authentication configuration
        // see https://docs.microsoft.com/en-us/aspnet/core/security/authentication
        services.AddAuthentication()
           .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, options =>
           {
              // Gets the authentication server from Authentication.json
              options.Authority = configuration["Authentication:Authority"];

              options.RequireHttpsMetadata =
                 !String.IsNullOrWhiteSpace(options.Authority) &&
                 options.Authority.StartsWith("https://", StringComparison.OrdinalIgnoreCase);

              options.TokenValidationParameters.ValidateAudience = false;

              options.IncludeErrorDetails = true;

              // It is recommended to enable this option only in development environment, and disable
it in production environment for better compliance with GDPR
              // see https://aka.ms/IdentityModel/PII
              IdentityModelEventSource.ShowPII = true;
           });

        // Enables PowerServer authorization
        // see https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction
        services.AddAuthorization(options =>
        {
           // Rewrite the defaultPowerServer authorization strategy
           options.AddPolicy(PowerServerConstants.DefaultAuthorizePolicy, policy =>
           {
              // Requires that the user has passed authentication
              policy.RequireAuthenticatedUser();

              // Requires the authentication method is JWTBearer
              policy.AddAuthenticationSchemes(JwtBearerDefaults.AuthenticationScheme);
```

```
                // Defines additional authentication logics if needed
                // For example, requires that access_token contains "serverapi" scope
                //policy.RequireClaim("scope", "serverapi");
                // ...
            });
        });

        return services;
    }
  }
}
```

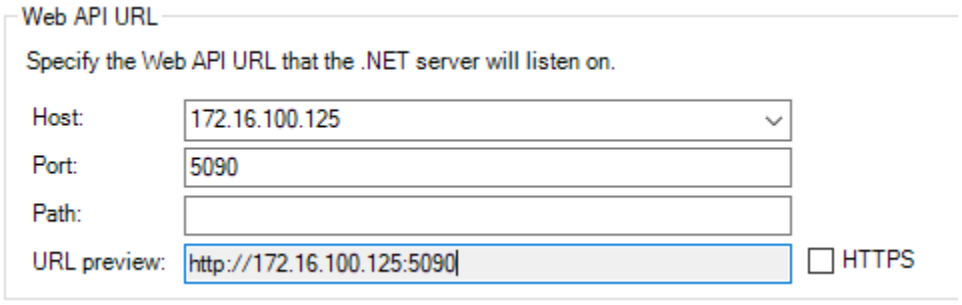
6.2 Specifying the Duende.IdentityServer license key

The production use of Duende.IdentityServer requires the license key. Please remember to specify the key in the ServerAPIs\Authentication\AuthenticationExtensions.cs.

```
        var builder = services.AddIdentityServer(options =>
        {
            options.LicenseKey = ""; //enter your Duende.IdentityServer license key
            options.Events.RaiseSuccessEvents = true;
            options.Events.RaiseFailureEvents = true;
            options.Events.RaiseInformationEvents = true;
            options.Events.RaiseErrorEvents = true;
            options.EmitStaticAudienceClaim = true;
        });
```

6.3 Updating the Authentication.json file

The Authentication: Authority value in the Authentication.json file shall be set the same as the URL preview setting for the PowerServer project.




```
"Authentication": {
    "Authority": "http://172.16.100.125:5090/"
}
```