Lab 2: Extend the Azure Api App to pass through authentication to Azure SQL Database.

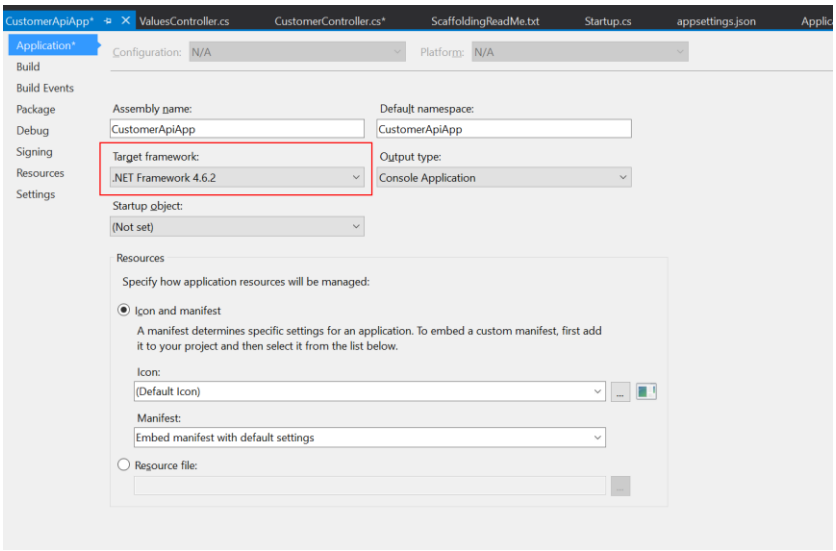
In this lab you will extend the Azure Api Application created in the previous lab. At the end of this lab you will have a Api application build on .Net Core secured with Azure Active Directory that is able to query a SQL Database with the logged in user of the application.

The following activities are going to be done:

1. Change Framework version of the project
2. Install NuGet Package
3. Adding application settings
4. Add an Authentication Helper to the Azure Api App.
5. Add a Customer Controller

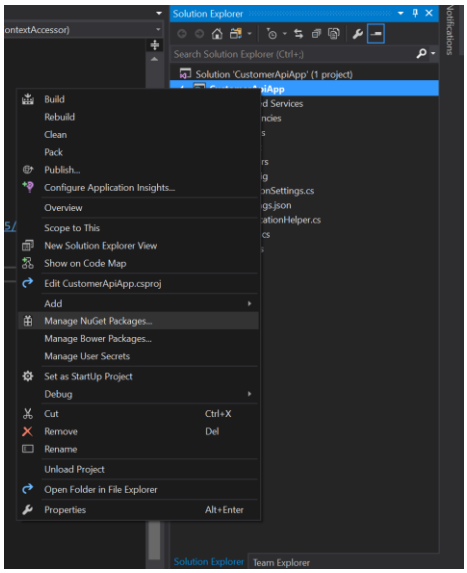
Change framework version of the project

1. Open Visual Studio 2017 as Administrator
2. Open the solution created in Lab 1.
3. Right click the project and select “Properties”
4. In the “Properties” make sure the “Target Framework” is set to “.Net Framework 4.6.2” and save the configuration.

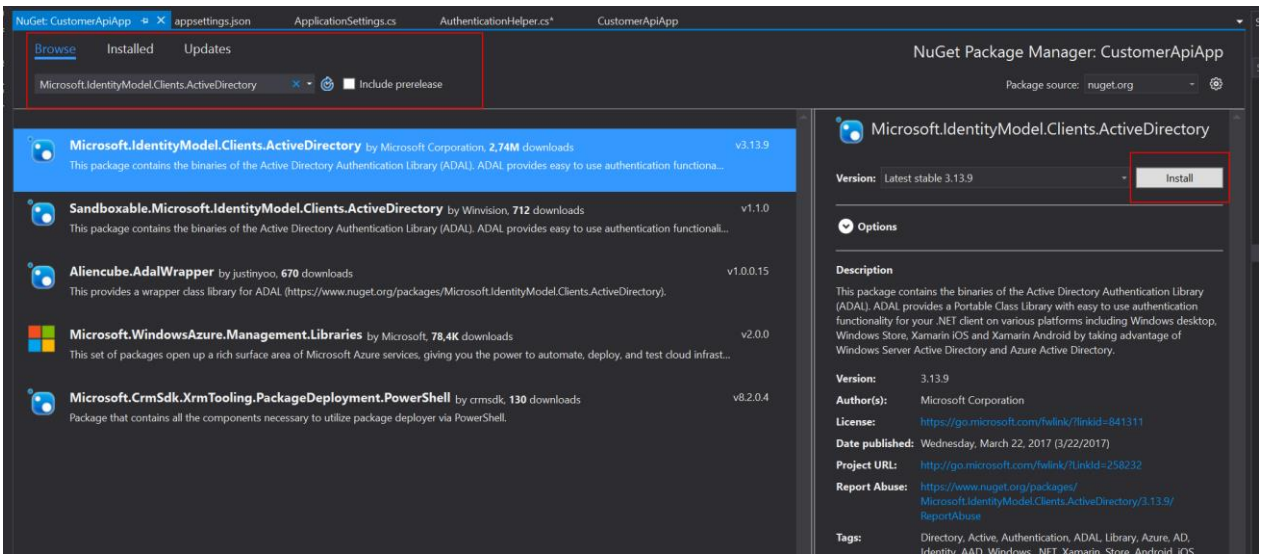


Install NuGet Package

1. Right click the project and select “Manage NuGet Packages…”

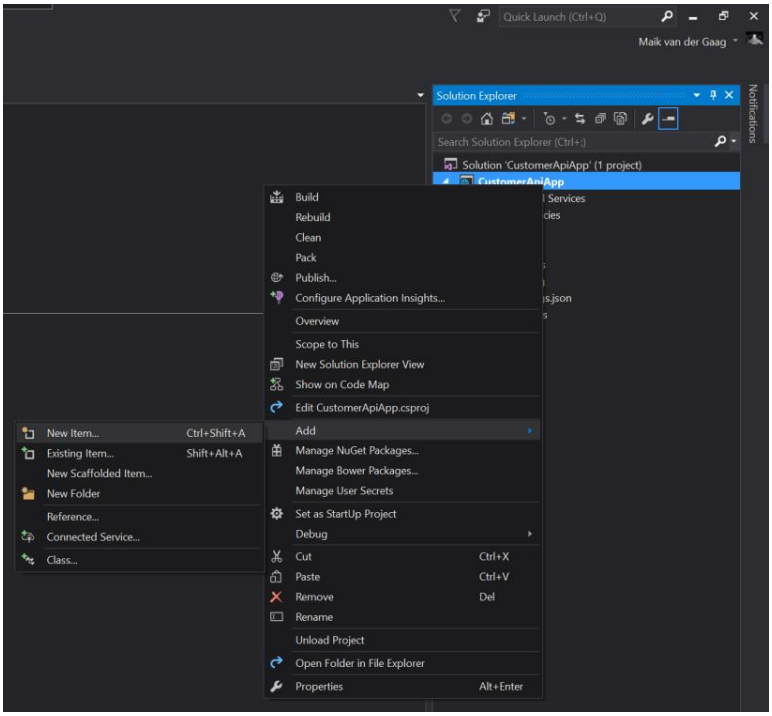


2. In the NuGet Packages window click on browse and make sure you install the following NuGet Package:
    - Microsoft.IdentityModel.Clients.ActiveDirectory

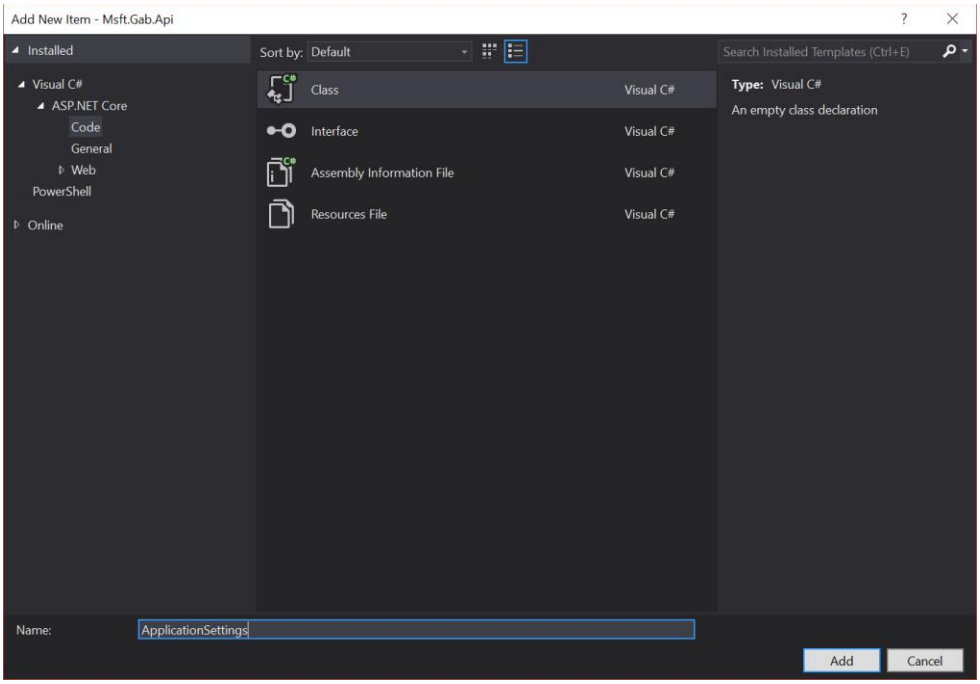


Adding application settings

1. Open Visual Studio 2017 as Administrator if not already opened.
2. Open the solution created in Lab 1.
3. Right click the project and select “Add” – “Class”.



4. Select “Visual C#”- “ASP.NET Core” – “Code” then select the Class type and give it a name like ApplicationSettngs.



5. In this class add application settings properties that are needed for the Api. Add the properties below.

```
public string ClientId { get; set; }

public string ClientSecret { get; set; }

public string AadInstance { get; set; }

public string Domain  { get; set; }

public string Audience { get; set; }

public string TenantId { get; set; }

public string AzureSqlResource { get; set; }

public string ConnectionString { get; set; }
```

Property	Description
ClientId	The client id of the current application registered in Active Directory.
ClientSecret	The secret of the current client application.
AadInstance	The login URL of Azure Active Directory
Domain	The Domain of Azure Active Directory
Audience	The application resource App ID URL
TenantId	The Tenant Id of Azure Active Directory
AzureSqlResource	The resource Id of Azure SQL Databases
ConnectionString	The Connection String to out Azure SQL Database

6. To load the settings within our application we first need to add the settings to the settings file ASP.Net Core uses. Open the “appsettings.json” file and add the following section:

```
"AuthenticationSettings": {
  "AadInstance": "https://login.microsoftonline.com/",
  "Audience": "{App ID URL Application}",
  "ClientId": "{ClientId}",
  "ClientSecret": "{ClientSecret}",
  "Domain": "{Azure Active Directory Domain}",
  "TenantId": "{Domain Tenant Id}",
  "ConnectionString": "Data Source={Azure SQL Server};Initial Catalog={Azure SQL
Database};Pooling=False;MultipleActiveResultSets=False;Encrypt=True;TrustServerCerti
ficate=False;Connection Timeout=30;",
  "AzureSqlResource": "https://database.windows.net/"
}
```

Note: *The values will be partially added in this lab and partially in the next one.*

7. With the help of dependency injection, the settings will get loaded into our application. To configure this correctly open the file: “Startup.cs”
8. Replace the “ConfigureServices” method with the below one.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    //Loads the configuration section into out Authentication Settings Object

services.Configure<ApplicationSettings>(Configuration.GetSection("AuthenticationSett
ings"));
    //Make sure that we have a context within out ASP.Net Core Project
    services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
}
```

Note: *As you can see within the comment, the second line makes sure the configuration section within "appsettings.json" get loaded into a AuthenticationSettings object. The third line adds HttpContext to our ASP.Net Core project.*

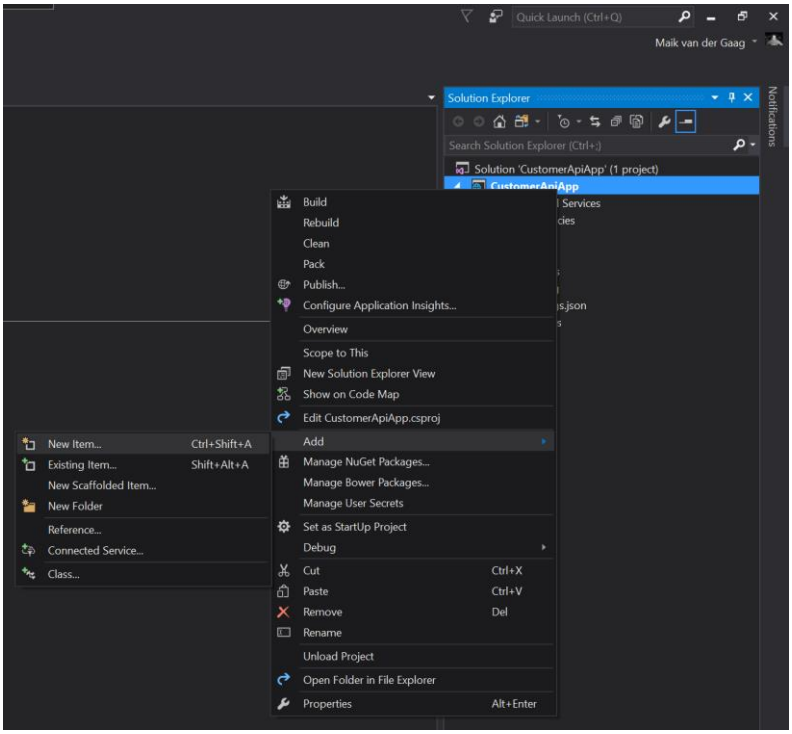
9. Because we do not want duplicate settings within out application we also need to replace some values within the "Configure" method. Make sure that the method looks like the below one when ready.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    app.UseJwtBearerAuthentication(new JwtBearerOptions
    {
        Authority = Configuration["AuthenticationSettings:AadInstance"] +
Configuration["AuthenticationSettings:TenantId"],
        Audience = Configuration["AuthenticationSettings:Audience"]
    });

    app.UseMvc();
}
```

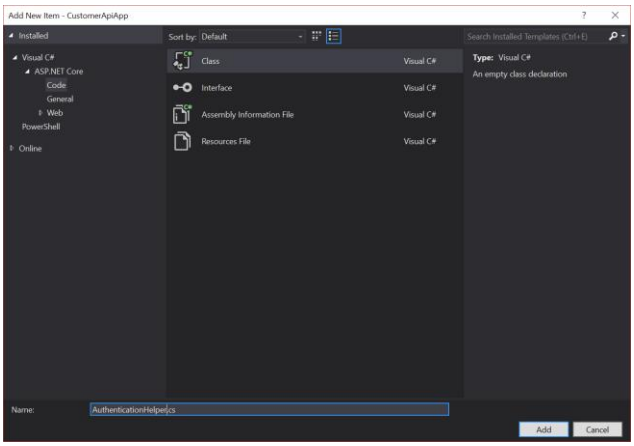
10. Move the values from the "Authentication" – "AzureAd" section in the "appsettings.json"file to our "AuthenticationSettings" section.
11. Delete the complete "Authentication" section and save the file.

Add an Authentication Helper to the Azure Api App

1. Open Visual Studio 2017 as Administrator if not already opened.
2. Open the solution created in Lab 1.
3. Right click the project and select “Add” – “Class”.



4. Select “Visual C#”- “ASP.NET Core” – “Code” then select the Class type and give it a name like AuthenticationHelper.



5. Make the class a static class, this is because we will note initiate objects from this class and add the following usings:

```
using Microsoft.AspNetCore.Http;
using Microsoft.IdentityModel.Clients.ActiveDirectory;
using System.Linq;
using System.Security.Claims;
```

6. In the newly created class add a method that will create a UserAssertion that can be used to login to the Azure SQL database, by retrieving the token of the currently logged in user from the authorization header.

Note: The UserAssertion object is a credential type representing user credentials.

```
private static UserAssertion GetUserAssertion(IHttpContextAccessor httpContextAccessor)
{
    UserAssertion retVal = null;

    string accessToken =
httpContextAccessor.HttpContext.Request.Headers["Authorization"][0];
    string userAccessToken = accessToken.Substring(accessToken.LastIndexOf('
')).Trim();

    Claim upn = httpContextAccessor.HttpContext.User.Claims.First(c => c.Type ==
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn");

    if (upn != null) {
        retVal = new UserAssertion(userAccessToken, "urn:ietf:params:oauth:grant-
type:jwt-bearer", upn.Value);
    }

    return retVal;
}
```

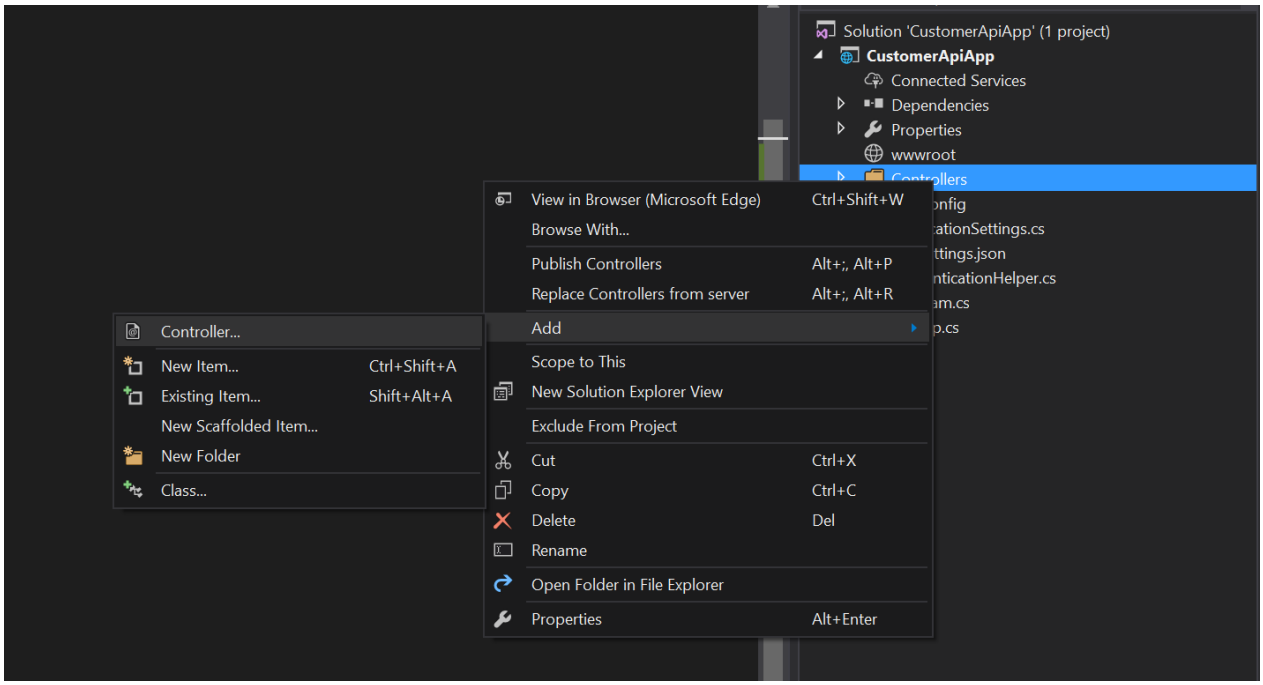
7. With the UserAssertion object we can retrieve an Access Token from Azure Active Directory to authenticate against our Azure SQL Database by adding the resource: https://database.windows.net/ to authenticate against.

```
public static AuthenticationResult GetAuthenticationResult(IHttpContextAccessor
httpContextAccessor, ApplicationSettings authSettings) {

    AuthenticationResult retVal = null;

    UserAssertion userInfo = GetUserAssertion(httpContextAccessor);
    ClientCredential clientCred = new ClientCredential(authSettings.ClientId,
authSettings.ClientSecret);
    AuthenticationContext authContext = new
AuthenticationContext(authSettings.AadInstance + authSettings.Domain);

    bool retry = false;
    int retryCount = 0;

    do {
        retry = false;
        try {
            retVal = authContext.AcquireTokenAsync(authSettings.AzureSqlResource,
clientCred, userInfo).Result;

        } catch (AdalException ex) {
            if (ex.ErrorCode == "temporarily_unavailable") {
                retry = true;
                retryCount++;
            }
        }
    } while ((retry == true) && (retryCount < 1));

    return retVal;
}
```

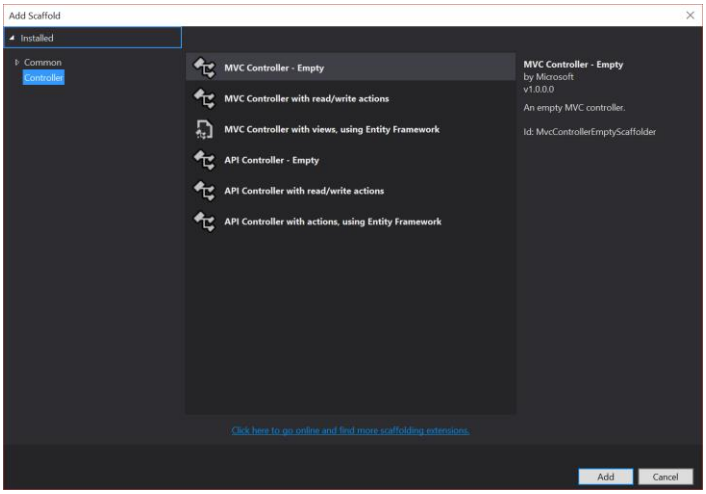
8. The token retrieved by this method will be used in the request to Azure SQL Server.

Add a Customer Controller

1. Right click the “Controllers” folder and select “Add” – “Controller”.



2. In the dialog that appears select “MVC Controller – Empty” click “Add” and in the next dialog name it CustomerController.cs and click on “Ok”.



3. In the Customer Controller add the below usings.

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Clients.ActiveDirectory;
using Newtonsoft.Json;
using System.Data;
using System.Data.SqlClient;
```

4. Add the below attributes add class level for the routing within ASP.Net Core and to make sure the user who calls this controller is authenticated.

```
[Authorize]
[Route("api/[controller]")]
```

5. Add the following fields and constructor to the controller. The default dependency injection within ASP.Net Core will make sure the objects will be initiated.

```
private IHttpContextAccessor httpContextAccessor;

private ApplicationSettings authSettings { get; set; }

public CustomerController(IHttpContextAccessor httpContextAcc,
IOptions<ApplicationSettings> settings) {
    httpContextAccessor = httpContextAcc;
    authSettings = settings.Value;
}
```

6. Add a Get Method ass below to your controller. This Get methods will get a token based on the currently signed in user via the "AuthenticationHelper" we created and pass this token to the SQL Data Connection and performs a select all on the specified table (Make sure you replace the value in the square brackets with your own table name).

```
[HttpGet]
public JsonResult Get() {
    JsonResult retVal = null;

    AuthenticationResult authResult =
AuthenticationHelper.GetAuthenticationResult(httpContextAccessor, authSettings);

    if (authResult != null) {
        string queryString = "SELECT * FROM SalesLT.Product";

        using (SqlConnection connection = new
SqlConnection(authSettings.ConnectionString)) {
            connection.AccessToken = authResult.AccessToken;
            try {
                connection.Open();
                SqlCommand command = new SqlCommand(queryString, connection);
                SqlDataAdapter adapter = new SqlDataAdapter(command);

                DataTable table = new DataTable();
                adapter.Fill(table);

                retVal = new JsonResult(table);

            } catch (SqlException ex) {
            }
        }
    }
    return retVal;
}
```

Note: *Your project needs to run on .Net Framework 4.6.2 to be able to use Azure Active Directory authentication on the Azure SQL Database.*