|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Migration Applications en MSAL HotFix Complément** | | |  | |
| **De :** | | **Date** | **12/04/2022** | | |
| **Date Etude** |  | | |
| **A :** | | | | | |
| **Cc :** | | | | | |
| **Objet** :  **Migration Applications en MSAL HotFix** | | | | | **Importance :** Standard |

Migration des Applications en MSAL HotFix : Complément

Historique du document

|  |  |  |
| --- | --- | --- |
| Date de modification | Nom | Commentaire |
| 12/04/2022 | Anis BEN ASKER | Création du document |

Table des matières

[1 Présentation 2](#_Toc2075403800)

[2 Prérequis 3](#_Toc407041593)

[3 Frontend 3](#_Toc1274433480)

[3.1 Etapes 3](#_Toc754767863)

[4 Backend /Backend Externe 8](#_Toc1947522916)

[4.1 Etapes 9](#_Toc1771473859)

[5 Remplacement getValidToken par getValidTokenAsync 9](#_Toc752713778)

[5.1 Cas d’une application React : 10](#_Toc1087502828)

[5.2 Cas d'une requête Ajax : 13](#_Toc1312842990)

[5.3 Cas d'une grille Kendo MVC 14](#_Toc691773239)

[5.4 Cas d'une grille Kendo Javascript 16](#_Toc1342962715)

[5.5 Cas devExpress et kendoGridBinder (404) 17](#_Toc532502501)

[6 Aspose 18](#_Toc1824769805)

[7 Vérification à faire : 19](#_Toc861548389)

# Présentation

Ce document a vocation d’identifier les actions à faire permettant :

* L’installation d’un HotFix MSAL d’une application déjà migré en MSAL : Frontend, Backend, Service Windows

Les NuGets qui ont été migrés en MSAL sont :

- DomusVi.Framework.\*

# Prérequis

Avant de commencer il faut:

* Tester que l’application fonctionne correctement sans bug
* Déployer en local:
  + la MasterPage de la branche BP/master avec le profil Deploy **DEVELOPPEMENT - LOCALHOST**
  + la page d’accueil: portal.webclient et Backend\_Portal.WebApi de la branche ACC/master
  + AuthSvc.WebApi de la branche AUTH/master
* Si vous n’avez pas de developpement en cours dans le repo alors supprimez tout le Repo en local et reclonez le avant de commencer
* Créer une branche ***feature/msal-hotfix*** à partir de *feature/msal* dans laquelle on va faire le hotfix msal.
* Supprimer les dossiers **packages** et **bin** (ou nettoyer la solution) de chaque projet
* Nettoyer le cache des NuGets en local avec le BAT suivant :

\dvdpri\DOSI - Portail - Documents\DevRessources\Tools MSAL\ NugetClear.zip

Compiler et tester le fonctionnement de l’application.

Si l’application fonctionne en local alors supprimez les dossiers **packages** et **bin** (ou nettoyer la solution) de chaque projet

# Frontend

## Etapes

1 - Mettre à jour les NuGets ***DomusVi.Framework.\****dans tous les projets qui les utilisent

2 - Réécriture de la classe Startup.Auth.cs en gardant le même **namespace** de la classe de l’application courante.

Dans la majorité des applications qui se composent d’un Frontend et un Backend, le code de *Startup.Auth.cs* du Front est le suivant. Dans le cas d’une application webform (dvdnet, FicheEngagement,…) une autre version de App\_Start/*Startup.Auth.cs* est à utiliser.

* *Startup.Auth.cs* standard pour un **Frontend**

|  |
| --- |
| using DomusVi.Framework.Identity;  using DomusVi.Framework.Security.Models;  using DomusVi.Framework.Security.Owin;  using Microsoft.Identity.Client;  using Microsoft.IdentityModel.Logging;  using Microsoft.Owin;  using Microsoft.Owin.Cors;  using Microsoft.Owin.Security;  using Microsoft.Owin.Security.Cookies;  using Microsoft.Owin.Security.OpenIdConnect;  using Owin;  using System;  using System.Collections.Generic;  using System.Configuration;  using System.Linq;  using System.Security.Claims;  using System.Threading.Tasks;  **namespace portal.webclient**  {  public partial class Startup  {  private static string clientId = ConfigurationManager.AppSettings["ida:ClientId"];  private static string aadInstance = ConfigurationManager.AppSettings["ida:AADInstance"];  private static string tenantId = ConfigurationManager.AppSettings["ida:TenantId"];  private static string postLogoutRedirectUri = ConfigurationManager.AppSettings["ida:PostLogoutRedirectUri"];  public static readonly string Authority = aadInstance + tenantId;  private Dictionary<String, TokenInfos> userTokenDictionary = new Dictionary<string, TokenInfos>();  private static string backendScope = ConfigurationManager.AppSettings["ida:BackendScopeUrl"];  private static string defaultScope = ConfigurationManager.AppSettings["ida:DefaultScope"];  private string tokenValid = string.Empty;  private string NonceFront = string.Empty;  private static NLog.Logger \_logger = NLog.LogManager.GetCurrentClassLogger();  /// <summary>  /// Cette méthode permet de personnaliser l’identité en ajoutant un claims portant :  /// - l’identifiant migation récupéré ( via Microsoft Graph) dans l’annuaire AAD</summary>  /// - le token permettant de consommer des Web Api  /// <param name="identity"></param>  /// <returns></returns>  private ClaimsIdentity TransformClaims(ClaimsIdentity identity)  {  Claim clUserGuid = identity.Claims.FirstOrDefault(c => c.Type == DviClaimTypes.oid);  \_logger.Trace("TransformClaims -> Start : clUserGuid: " + clUserGuid?.Value);  if (clUserGuid != null)  {  string userUID = clUserGuid.Value;  if (!string.IsNullOrEmpty(tokenValid))  {  var employeeIdTask = Task.Run(async () => await UserIdentityHelper.GetExtendedPropertyEmployeeIdAsync(tokenValid));  var employeeId = employeeIdTask.Result;  identity.AddClaim(new Claim(DviClaimTypes.ppid, employeeId + ""));  }  string keyUserToken = "\_Token\_" + userUID;  TokenInfos tokenInfo = (userTokenDictionary.ContainsKey(keyUserToken) ? userTokenDictionary[keyUserToken] : null);  if (tokenInfo != null)  {  \_logger.Trace("TransformClaims -> tokenInfo != null ");  // .. add here token in identity to make it available throw controllers  identity.AddClaim(new Claim(DviClaimTypes.OAuth2Token, tokenInfo.AccessToken));  identity.AddClaim(new Claim(DviClaimTypes.OAuth2RefreshToken, tokenInfo.RefreshToken));  identity.AddClaim(new Claim(DviClaimTypes.OAuth2TokenExpire, tokenInfo.Expires.ToString()));  identity.AddClaim(new Claim(DviClaimTypes.OAuth2TokenIssued, tokenInfo.Issued.ToString()));  identity.AddClaim(new Claim(DviClaimTypes.NonceIdToken, NonceFront));  }  else  {  \_logger.Info("TransformClaims -> tokenInfo is Null: keyUserToken:" + keyUserToken);  }  }  return identity;  }  // Pour plus d'informations sur la configuration de l'authentification, consultez http://go.microsoft.com/fwlink/?LinkId=301864  public void ConfigureAuth(IAppBuilder app)  {  \_logger.Trace("Start ConfigureAuth");  app.UseCors(CorsOptions.AllowAll);  app.SetDefaultSignInAsAuthenticationType(CookieAuthenticationDefaults.AuthenticationType);  app.UseCookieAuthentication(new CookieAuthenticationOptions()  {  CookieManager = new InteropChunkCookieManager(),  Provider = new CookieAuthenticationProvider  {  OnResponseSignIn = ctx =>  {  ctx.Identity = TransformClaims(ctx.Identity);  }  },  CookieName = ConfigurationManager.AppSettings["cookieTokenAuthId"],  CookieSameSite = SameSiteMode.None  });  app.UseOpenIdConnectAuthentication(  new OpenIdConnectAuthenticationOptions  {  ClientId = clientId,  Authority = Authority,  PostLogoutRedirectUri = postLogoutRedirectUri,  RedirectUri = ConfigurationManager.AppSettings["ida:AppRedirectUrlAfterLogin"],  Notifications = new OpenIdConnectAuthenticationNotifications()  {  // Si la réponse OpenID Connect contient un code, échangez-le contre un jeton d'accès et actualisez le jeton, puis rangez-les.  AuthorizationCodeReceived = (context) =>  {  try  {  var code = context.Code;  \_logger.Trace("AuthorizationCodeReceived -> Start : code: " + code);  IConfidentialClientApplication clientapp;  clientapp = UserIdentityHelper.BuildConfidentialClientApplication();  string[] scopes = new string[] { defaultScope };  var myResultAuthorizationCode = Task.Run(async () => await clientapp.AcquireTokenByAuthorizationCode(scopes, code).ExecuteAsync());  var tokenResult = myResultAuthorizationCode.Result;  \_logger.Trace("AuthorizationCodeReceived -> tokenResult.UniqueId: " + tokenResult.UniqueId);  \_logger.Trace("AuthorizationCodeReceived -> tokenResult.TenantId: " + tokenResult.TenantId);  // Récupération des information du compte  var accountAsync = Task.Run(async () => await clientapp.GetAccountAsync($"{tokenResult.UniqueId}.{tokenResult.TenantId}"));  var account = accountAsync.Result;  // Obtention du token pour la récupération de l'employeeId lié au compte.  var tokenCurrentAccount = Task.Run(async () => await clientapp.AcquireTokenSilent(scopes, account).ExecuteAsync());  var tokenCurrentUser = tokenCurrentAccount.Result;  #region GET Nonce  // Nonce: La valeur à usage unique correspond au paramètre inclus dans la requête /authorize d’origine au point de distribution d’émission.  // Cette valeur est utiliser lors du renouvellement du token dans getValidTokenAsync dans master.js de la MasterPage  NonceFront = TokenManager.GetNonceFromIdToken(tokenCurrentUser.IdToken);  \_logger.Trace("AuthorizationCodeReceived -> NonceFront: " + NonceFront);  #endregion GET Nonce  // variable global définit pour récupérer le token de l'utilisateur courant.  tokenValid = tokenCurrentUser.AccessToken;  \_logger.Trace("AuthorizationCodeReceived -> Token Front: " + tokenValid);  //Obtention du token pour permettre la consommation du Backend  string[] scopesBackend = new string[] { backendScope };  var tokenSilent = Task.Run(async () => await clientapp.AcquireTokenSilent(scopesBackend, account).ExecuteAsync());  var token = tokenSilent.Result;  \_logger.Trace("AuthorizationCodeReceived -> Token Back: " + token?.AccessToken);  TokenInfos tokenInfos = new TokenInfos()  {  AccessToken = token.AccessToken,  RefreshToken = "",  Expires = token.ExpiresOn.ToLocalTime(),  ExpiresIn = (uint)(token.ExpiresOn.ToLocalTime() - DateTime.Now).Minutes,  Issued = DateTime.Now,  TokenType = "bearer"  };  string userObjectID = token.UniqueId;  userTokenDictionary["\_Token\_" + userObjectID] = tokenInfos;  }  catch (Exception er)  {  \_logger.Fatal(er, er.Message + " Stack:" + er.StackTrace);  throw;  }  return Task.FromResult(0);  },  AuthenticationFailed = (context) =>  {  \_logger.Fatal(context.Exception, " AuthenticationFailed:" + (context?.Exception?.Message ?? "") + " Stack: " + (context?.Exception?.StackTrace ?? ""));  if (context.Exception.Message.Contains("IDX21323"))  {  //Correction Bug: IDX21323: RequireNonce is '[PII of type 'System.Boolean' is hidden. For more details, see https://aka.ms/IdentityModel/PII.]'.  context.HandleResponse();  context.OwinContext.Authentication.Challenge();  }  return Task.FromResult(0);  }  }  });  IdentityModelEventSource.ShowPII = true;  }  }  } |

* *Startup.Auth.cs* pour une application **WebForms**

|  |
| --- |
| using DomusVi.Framework.Identity;  using DomusVi.Framework.Security.Models;  using DomusVi.Framework.Security.Owin;  using Microsoft.Identity.Client;  using Microsoft.IdentityModel.Logging;  using Microsoft.IdentityModel.Tokens;  using Microsoft.Owin;  using Microsoft.Owin.Cors;  using Microsoft.Owin.Security;  using Microsoft.Owin.Security.ActiveDirectory;  using Microsoft.Owin.Security.Cookies;  using Microsoft.Owin.Security.OpenIdConnect;  using Owin;  using System;  using System.Collections.Generic;  using System.Configuration;  using System.Linq;  using System.Security.Claims;  using System.Threading.Tasks;  public partial class Startup  {  private static string clientId = ConfigurationManager.AppSettings["ida:ClientId"];  private static string aadInstance = ConfigurationManager.AppSettings["ida:AADInstance"];  private static string tenantId = ConfigurationManager.AppSettings["ida:TenantId"];  private static string postLogoutRedirectUri = ConfigurationManager.AppSettings["ida:PostLogoutRedirectUri"];  public static readonly string Authority = aadInstance + tenantId;  private Dictionary<String, TokenInfos> userTokenDictionary = new Dictionary<string, TokenInfos>();  private static string backendScope = ConfigurationManager.AppSettings["ida:BackendScopeUrl"];  private static string defaultScope = ConfigurationManager.AppSettings["ida:DefaultScope"];  private string tokenValid = string.Empty;  private string NonceFront = string.Empty;  private static NLog.Logger \_logger = NLog.LogManager.GetCurrentClassLogger();  /// <summary>  /// Cette méthode permet de personnaliser l’identité en ajoutant un claims portant :  /// - l’identifiant migation récupéré ( via Microsoft Graph) dans l’annuaire AAD</summary>  /// - le token permettant de consommer des Web Api  /// <param name="identity"></param>  /// <returns></returns>  private ClaimsIdentity TransformClaims(ClaimsIdentity identity)  {  Claim clUserGuid = identity.Claims.FirstOrDefault(c => c.Type == DviClaimTypes.oid);  if (clUserGuid != null)  {  string userUID = clUserGuid.Value;  if (!string.IsNullOrEmpty(tokenValid))  {  var employeeIdTask = Task.Run(async () => await UserIdentityHelper.GetExtendedPropertyEmployeeIdAsync(tokenValid));  var employeeId = employeeIdTask.Result;  identity.AddClaim(new Claim(DviClaimTypes.ppid, employeeId + ""));  }  string keyUserToken = "\_Token\_" + userUID;  TokenInfos tokenInfo = (userTokenDictionary.ContainsKey(keyUserToken) ? userTokenDictionary[keyUserToken] : null);  if (tokenInfo != null)  {  // .. add here token in identity to make it available throw controllers  identity.AddClaim(new Claim(DviClaimTypes.OAuth2Token, tokenInfo.AccessToken));  identity.AddClaim(new Claim(DviClaimTypes.OAuth2RefreshToken, tokenInfo.RefreshToken));  identity.AddClaim(new Claim(DviClaimTypes.OAuth2TokenExpire, tokenInfo.Expires.ToString()));  identity.AddClaim(new Claim(DviClaimTypes.OAuth2TokenIssued, tokenInfo.Issued.ToString()));  identity.AddClaim(new Claim(DviClaimTypes.NonceIdToken, NonceFront));  }  else  {  \_logger.Info("TransformClaims -> tokenInfo is Null: keyUserToken:" + keyUserToken);  }  }  return identity;  }  // Pour plus d'informations sur la configuration de l'authentification, consultez http://go.microsoft.com/fwlink/?LinkId=301864  public void ConfigureAuth(IAppBuilder app)  {  app.UseCors(CorsOptions.AllowAll);  app.SetDefaultSignInAsAuthenticationType(CookieAuthenticationDefaults.AuthenticationType);  app.UseCookieAuthentication(new CookieAuthenticationOptions()  {  CookieManager = new InteropChunkCookieManager(),  Provider = new CookieAuthenticationProvider  {  OnResponseSignIn = ctx =>  {  ctx.Identity = TransformClaims(ctx.Identity);  }  },  CookieName = ConfigurationManager.AppSettings["cookieTokenAuthId"],  CookieSameSite = SameSiteMode.None  });  app.UseOpenIdConnectAuthentication(  new OpenIdConnectAuthenticationOptions  {  ClientId = clientId,  Authority = Authority,  PostLogoutRedirectUri = postLogoutRedirectUri,  RedirectUri = ConfigurationManager.AppSettings["ida:AppRedirectUrlAfterLogin"],  Notifications = new OpenIdConnectAuthenticationNotifications()  {  // Si la réponse OpenID Connect contient un code, échangez-le contre un jeton d'accès et actualisez le jeton, puis rangez-les.  AuthorizationCodeReceived = (context) =>  {  try  {  var code = context.Code;  IConfidentialClientApplication clientapp;  clientapp = UserIdentityHelper.BuildConfidentialClientApplication();  string[] scopes = new string[] { defaultScope };  var myResultAuthorizationCode = Task.Run(async () => await clientapp.AcquireTokenByAuthorizationCode(scopes, code).ExecuteAsync());  var tokenResult = myResultAuthorizationCode.Result;  // Récupération des information du compte  var accountAsync = Task.Run(async () => await clientapp.GetAccountAsync($"{tokenResult.UniqueId}.{tokenResult.TenantId}"));  var account = accountAsync.Result;  // Obtention du token pour la récupération de l'employeeId lié au compte.  var tokenCurrentAccount = Task.Run(async () => await clientapp.AcquireTokenSilent(scopes, account).ExecuteAsync());  var tokenCurrentUser = tokenCurrentAccount.Result;  #region GET Nonce  // Nonce: La valeur à usage unique correspond au paramètre inclus dans la requête /authorize d’origine au point de distribution d’émission.  // Cette valeur est utiliser lors du renouvellement du token dans getValidTokenAsync dans master.js de la MasterPage  NonceFront = TokenManager.GetNonceFromIdToken(tokenCurrentUser.IdToken);  \_logger.Trace("AuthorizationCodeReceived -> NonceFront: " + NonceFront);  #endregion GET Nonce  // variable global définit pour récupérer le token de l'utilisateur courant.  tokenValid = tokenCurrentUser.AccessToken;  //Obtention du token pour permettre la consommation du Backend  string[] scopesBackend = new string[] { backendScope };  var tokenSilent = Task.Run(async () => await clientapp.AcquireTokenSilent(scopesBackend, account).ExecuteAsync());  var token = tokenSilent.Result;  TokenInfos tokenInfos = new TokenInfos()  {  AccessToken = token.AccessToken,  RefreshToken = "",  Expires = token.ExpiresOn.ToLocalTime(),  ExpiresIn = (uint)(token.ExpiresOn.ToLocalTime() - DateTime.Now).Minutes,  Issued = DateTime.Now,  TokenType = "bearer"  };  string userObjectID = token.UniqueId;  userTokenDictionary["\_Token\_" + userObjectID] = tokenInfos;  }  catch (Exception er)  {  \_logger.Fatal(er, er.Message + " Stack:" + er.StackTrace);  throw;  }  return Task.FromResult(0);  },  AuthenticationFailed = (context) =>  {  \_logger.Fatal(context.Exception, " AuthenticationFailed:" + (context?.Exception?.Message ?? "") + " Stack: " + (context?.Exception?.StackTrace ?? ""));  if (context.Exception.Message.Contains("IDX21323"))  {  //Correction Bug: IDX21323: RequireNonce is '[PII of type 'System.Boolean' is hidden. For more details, see https://aka.ms/IdentityModel/PII.]'.  context.HandleResponse();  context.OwinContext.Authentication.Challenge();  }  return Task.FromResult(0);  }  }  });  IdentityModelEventSource.ShowPII = true;  app.UseWindowsAzureActiveDirectoryBearerAuthentication(  new WindowsAzureActiveDirectoryBearerAuthenticationOptions  {  Tenant = ConfigurationManager.AppSettings["ida:Tenant"],  TokenValidationParameters = new TokenValidationParameters()  {  ValidateIssuer = false,  ValidAudience = ConfigurationManager.AppSettings["ida:BackendScopeUrl"]  }  });  // This makes any middleware defined above this line run before the Authorization rule is applied in web.config  //app.UseStageMarker(PipelineStage.Authenticate);  }  } |

Compilez, testez en local et ensuite déployer en DEV

# Backend /Backend Externe

## Etapes

1 – Nettoyer la solution : supprimer le dossier packages et bin

2- Mettre à jour les NuGets ***DomusVi.Framework.\**** MSAL dans tous les projets qui les utilisent :

|  |
| --- |
|  |

# Remplacement getValidToken par getValidTokenAsync

Si vous avez le message d’erreur ci-dessous, dans votre application cela veut dire que l’application utilise la fonction Javascript l’ancienne fonction Javascript *getValidToken* qui est dépréciée et il faut la corriger comme cela.

|  |
| --- |
| MSAL-ERROR : getValidToken ne fonctionne plus sous MSAL si le token est expiré. Mettez à jour l'application avec getValidTokenAsync. |

**Le Token se récupère ainsi maintenant :**

|  |
| --- |
| master.auth.getValidTokenAsync().then((authToken) => console.log(authToken)) |

## Cas d’une application React :

Remplacer la classe *helpers\ApiHelper.jsx* par :

|  |
| --- |
| import axios from 'axios';  //const timeout = 3600000;  const timeout = 360000;  class ApiHelper {      //Alternative Ajax avec l'outil axios : les methodes methodes axios renvoient des Promises (https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser\_les\_promesses      //Permet de d'obtenir le token et configurer axios pour une requete get en mode asynchrone      //url : url de la ressource      static async getAsync(url) {          const auth\_token = await master.auth.getValidTokenAsync();          axios.defaults.headers.common['Authorization'] = 'Bearer ' + auth\_token;          axios.defaults.timeout = timeout;          return await axios.get(url);      }      static async postCrossDomainAsync(url, data) {          axios.defaults.timeout = timeout;          axios.defaults.headers.common['Access-Control-Allow-Origin'] = '\*';          return await axios.post(url, data);      }      //Permet de d'obtenir le token et configurer axios pour une requete post en mode asynchrone      //url: url à envoyer la ressource      //data: données (ressources) à envoyer (optionel)      // ! axios convertit les dates au format ISO 8601 lors de la methode stringify,      // on ajoute un workaround pour obtenir la bonne heure      static async postAsync(url, data) {          const auth\_token = await master.auth.getValidTokenAsync();          axios.defaults.headers.common['Authorization'] = 'Bearer ' + auth\_token;          axios.defaults.timeout = timeout;          return await axios.post(url, data);      }      // api must return HttpReponse with byteArray as Content      static async postGetFileAsync(url, data, fileName) {          const auth\_token = await master.auth.getValidTokenAsync();          axios.defaults.headers.common['Authorization'] = 'Bearer ' + auth\_token;          axios.defaults.timeout = timeout;          let config = {              responseType: 'arraybuffer'          };          await axios.post(url, data, config).then((response) => {              const url = window.URL.createObjectURL(new Blob([response.data]));              const link = document.createElement('a');              link.href = url;              link.setAttribute('download', fileName); //or any other extension              document.body.appendChild(link);              link.click();          });      }      //Permet de d'obtenir le token et configurer axios pour une requete post en mode asynchrone      //url: url à envoyer la ressource      //data: données (ressources) à envoyer (optionel)      static async putAsync(url, data) {          const auth\_token = await master.auth.getValidTokenAsync();          axios.defaults.headers.common['Authorization'] = 'Bearer ' + auth\_token;          axios.defaults.timeout = timeout;          return await axios.put(url, data);      }      //Permet de d'obtenir le token et configurer axios pour une requete delete en mode asynchrone      //url: url à envoyer la ressource      //data: données (ressources) à envoyer (optionel)      static async deleteAsync(url, data) {          const auth\_token = await master.auth.getValidTokenAsync();          axios.defaults.headers.common['Authorization'] = 'Bearer ' + auth\_token;          axios.defaults.timeout = timeout;          return await axios.delete(url, data);      }        //Permet de d'obtenir le token et configurer axios pour une requete get permettant de télécharger un fichier en mode asynchrone      //url : url de la ressource      static async getFileAsync(url) {          const auth\_token = await master.auth.getValidTokenAsync();          axios.defaults.headers.common['Authorization'] = 'Bearer ' + auth\_token;          axios.defaults.timeout = timeout;          let config = {              responseType: 'arraybuffer',              headers: { 'Content-Type': 'application/octet-stream' }          };          return await axios.get(url, config);      }      static async postFormDataAsync(url, formData, onUploadProgress) {          const auth\_token = await master.auth.getValidTokenAsync();          axios.defaults.headers.common['Authorization'] = 'Bearer ' + auth\_token;          axios.defaults.timeout = timeout;          let config = {              headers: { 'Content-Type': 'application/octet-stream' },              onUploadProgress: onUploadProgress          };          return await axios.post(url, formData, config);      }  }  export default ApiHelper; |

Dans le *helpers\DataManager.jsx* il faut modifier les fonctions :

* Ajouter un async devant les méthodes qui utilisent ApiHelper
* Ajouter un await devant ApiHelper

|  |
| --- |
|  |

Appelant

|  |
| --- |
| DataManager.getPlinkByModuleAsync().then((response) => {              //TODO          });  //OU  DataManager.getPlinkByModuleAsync() ; |

## Cas d'une requête Ajax :

Pour faire un appel ajax il faut appeler avant master.auth.**getValidTokenAsync**(). Voir exemple ci-dessous.

|  |
| --- |
| master.auth.**getValidTokenAsync**().then(function(**authToken**) {  $.ajax({  type: 'GET',  url: monUrl,  contentType: 'application/json; charset=utf-8',  beforeSend: function (xhr) {  **xhr.setRequestHeader('Authorization', 'Bearer ' + authToken);**  },  success: data => {...}  });  }) |

* Si l’appel ajax contient le paramètre async : false, c’est que, probablement, la fonction qui l’utilise est appelée dans d’autres fonctions en synchrone .
* La solution est de retourner une promise et de transformer les fonctions qui les appellent en async puis récupérer la variable avec un .then().

Exemple :

function CheckProfile() {

    return new Promise((resolve, reject) => {

        if (ProfilGSUser) {

            resolve(ProfilGSUser)

            return;

        }

        master.auth.getValidTokenAsync().then(function (authToken) {

            $.ajax({

                url: api.apiUserProfile,

                type: 'GET',

                contentType: "application/json; charset=utf-8",

                dataType: "json",

                //async: false,

                beforeSend: function (xhr) {

                    xhr.setRequestHeader('Authorization', 'Bearer ' + authToken);

                },

                success: function (GSProfile) {

                    ProfilGSUser = GSProfile;

                     resolve(ProfilGSUser);

                },

                error: function (error) {

                    reject(error)

                    console.error("User GS : la ressource API est injoignable");

                }

            });

        });

        return;

    });

}

//appeler la fonction en async

async function IsUserSuperAdminGS() {

    var data = await CheckProfile();

    return data.IS\_SUPER\_ADMIN == 1;

}

// chercher où est appelé la fonction async et remplacer par ceci

IsUserSuperAdminGS().then((data)=> {

          if (data) {

          $("#liClassification").show();

      }

      else {

          $("#liClassification").hide();

      }})

  });

## Cas d'une grille Kendo MVC

Voici la méthode pour initialiser le Token dans une Grille Kendo MVC:

Voir cette page : <https://www.telerik.com/forums/asp-net-core-custom-datasource-pass-bearer-token>

Tout d'abord dans l'initialisation de la grille kendo MVC dans le cshtml: mettre la propriété Autobind à false comme ceci :

|  |
| --- |
| .Scrollable() .Groupable() **.AutoBind(false)** |

Ceci pour empêcher le déclenchement du 1er appel dès l'ouverture de la page (sans Token à ce moment-là)

Puis dans la partie JS :

|  |
| --- |
| $(document).ready(function () {  master.auth.getValidTokenAsync()  .then(function(authToken) {  var grid = $(`#gridModalContactAdmis`).data("kendoGrid");  grid.dataSource.transport.options.read.beforeSend = function (xhr) {  xhr.setRequestHeader('Authorization', 'Bearer ' + authToken);  }  grid.dataSource.read();  // make a DataSource read, which will now use the authorization header  })  }); |

Si plusieurs grilles sur la même page (solution faite sur le projet prescripteur) :

|  |
| --- |
| $(document).ready(function () {  master.auth.getValidTokenAsync()  .then(function (auth\_token) {  initTokenAndReadGrid(auth\_token, 'gridActionFuture');  initTokenAndReadGrid(auth\_token, 'gridActionLate');  initTokenAndReadGrid(auth\_token, 'gridActionToday');  initTokenAndReadGrid(auth\_token, 'gridPrescripteurSleeping');  initTokenAndReadGrid(auth\_token, 'gridPrescripteurActif');  initTokenAndReadGrid(auth\_token, 'gridPrescripteurNouveau');  })  }); |

Pour avoir le Token, on initialise le *beforeSend* de la grille en JavaScript puis on fait le Read nous-même (*AutoBind* à false dans la Kendo Grid MVC)

|  |
| --- |
| function initTokenAndReadGrid(authToken, gridName) {  var grid = $(`#${gridName}`).data("kendoGrid");  grid.dataSource.transport.options.read.beforeSend = function (xhr) {  xhr.setRequestHeader('Authorization', 'Bearer ' + authToken);  }  // make a DataSource read, which will now use the authorization header  grid.dataSource.read();  } |

Selon l'OPTION que l'on a besoin :

|  |
| --- |
| grid.dataSource.transport.options.read.beforeSend = function (xhr) {  xhr.setRequestHeader('Authorization', 'Bearer ' + authToken);  }  grid.dataSource.transport.options.update.beforeSend = function (xhr) {  xhr.setRequestHeader('Authorization', 'Bearer ' + authToken);  }  grid.dataSource.transport.options.create.beforeSend = function (xhr) {  xhr.setRequestHeader('Authorization', 'Bearer ' + authToken);  } |

Remplacer **SetRequestHeader(xhr)**

|  |  |
| --- | --- |
| Ancien | var urlpost = $("#urlPostNewPreuve").val();  xhr.open('POST', urlpost, true);  SetRequestHeader(xhr);  xhr.send(formData); |
| Nouveau | master.auth.getValidTokenAsync()  .then(function (authToken) {  var urlpost = $("#urlPostNewPreuve").val();  xhr.open('POST', urlpost, true);  xhr.setRequestHeader('Authorization', 'Bearer ' + authToken);  xhr.send(formData);  }); |

## Cas d'une grille Kendo Javascript

Dans le cas dune Kendo grid Js il faut transformer l’appel le read en une fonction avec un appel Ajax (faire une recherche sur toute la solution avec le mot « transport »). Voici la référence suivie d’un exemple. <https://docs.telerik.com/kendoui/api/javascript/data/datasource/configuration/transport.read>





Exemple : // datasource Grille Tarifaire (Tous les éléments)

      datasourcePresents = new kendo.data.DataSource({

          autoSync: false,

          transport: {

              read: function (options) {

                  master.auth.getValidTokenAsync().then(function (authToken) {

                      $.ajax({

                          url: api.apiHebergementGT + idresidence,

                          dataType: "json",

                          beforeSend: function (xhr) {

                              xhr.setRequestHeader('Authorization', 'Bearer ' + authToken);

                          },

                          success: function (data) {

                              if (data.length != null) {

                                  options.success(data);

                              }

                              else

                                  options.error();

                          },

                          error: function (error) {

                              console.error("erreur recuperation grille tarifaire : " + error);

                          }

                      });

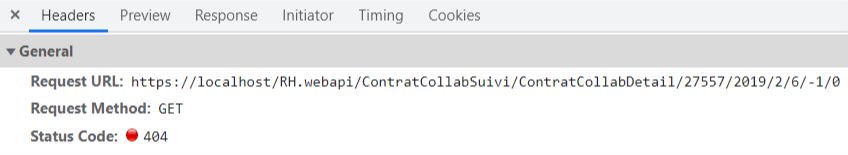
                  });

## Cas devExpress et kendoGridBinder (404)

Dans le cas de l’utilisation de DynamicQuery et KendoGridBinder, il peut y avoir des paramètres ajoutés en query string et qui sont absent dans la route (dans le controller côté webapi)

La mise à jour du .net vers la version 4.8 nous fera perdre cette fonctionnalité, on se retrouvera donc avec une erreur 404 vu que la route et la requête ajax reçu ne correspondent plus au nombre de paramètre de la fonction au sein du controller.

L'erreur peut ressemblé à ceci:

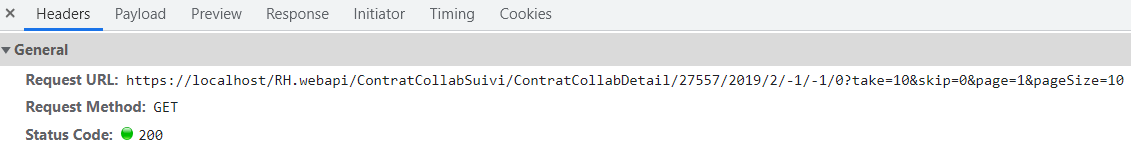


Il faudra donc ajouter les paramètres manquants de manière explicite dans l’ajax et en les récupérant lors de l’instanciation de la dataSource, exemple ci-dessous:

Exemple : // datasource RH (

       g\_dataSourceGridContratsDet = new kendo.data.DataSource({  
 type: "json",  
 autoSync: false,  
 // batch: true,  
 transport: {  
 read: function (options) {  
   
 master.auth.getValidTokenAsync().then(function (authToken) {  
   
 $.ajax({  
 url: new String(`${urlContratsDetGet}?take=${this.g\_dataSourceGridContratsDet.\_take}&skip=${this.g\_dataSourceGridContratsDet.\_skip}&page=${this.g\_dataSourceGridContratsDet.\_page}&pageSize=${this.g\_dataSourceGridContratsDet.\_pageSize}`).format(''),  
 datatype: "json",

Après correction:



## Cas d’un kendoUpload dans une grille Kendo

function onDataBoundRelances(e) {  
 //console.log('databound Relance Objets', e.sender.\_data);  
   
 master.auth.getValidTokenAsync().then(function (authToken) {  
   
 //initialisation kendo upload  
 $('#GridLitigeRelance input[type="file"]').kendoUpload({  
 async: {  
 saveUrl: "save",  
 autoUpload: true,  
 token: authToken  
 },  
 success: onSuccessRelance,  
 error: data => data,  
 upload: onUpload,  
 cancel: onCancel,  
 complete: onComplete,  
 error: onError,  
 progress: onProgress,  
 remove: onRemove,  
 select: onSelect,  
 multiple: false  
 });  
   
 });  
   
 //INFOBULLE ds databound car le dom ne connait pas les element de la grille avant qu'elle n'ait ete bindé  
   
 initPopover();  
 //popover trace  
 $('[data-toggle="popover"]').popover();  
}

// Lors du Chargement des fichiers de relance  
function onUpload(e) {  
   
 //reecrit le save pour set idrelance et idpreuve dynamiquement ;)  
 var url = api.urlApiSavePreuveFile + "/" + idrelance + "/" + idpreuve + "/" + idresidence;  
 e.sender.options.async.saveUrl = url;  
   
 console.log("url : ", url);  
   
 const authToken = e.sender.options.async.token;  
 //generation token  
 var xhr = e.XMLHttpRequest;  
   
 if (xhr) {  
   
 xhr.addEventListener("readystatechange", function onReady(e) {  
 if (xhr.readyState == 1 /\* OPENED \*/) {  
 xhr.setRequestHeader('Authorization', 'Bearer ' + authToken);  
   
 xhr.removeEventListener("readystatechange", onReady);  
 }  
 });  
 }  
   
   
   
}

On ne doit plus avoir ces instructions dans le code :

* master.auth.setRequestHeader
* SetRequestHeader(xhr)
* master.auth.getValidToken()

# Aspose

**Nettoyage**

Les Nugest *DomusVi.Framework.Aspose.\** sont installés généralement dans le backend

* Désinstaller les NuGets *DomusVi.Framework.Aspose.\** des projets **où ils ne sont pas utilisés** (Model, Entities, Dto par exemple).
* Supprimer tous les fichiers *Aspose.Total.lic* inutiles.

Pour trouver tous les fichiers *Aspose.Total.lic,* recherchez ce texte dans toute la solution « *<LicensedTo>DOMUSVI</LicensedTo>* »

* Enlever le fichier *Aspose.Total.lic* de la racine du projet FrontEnd (webclient), Backend(webapi), console, webforms,…
* Mettre à jour les NuGets *DomusVi.Framework.Aspose.\** (dernière version ~4.0.1) dans le projet où ils ont utilisés (généralement dans le projet Services).
* Vérifier la dernière licence Aspose dans tous les fichiers *Aspose.Total.lic,* on doit avoir la valeur 20230127 dans la baliseSubscriptionExpiry *(*<SubscriptionExpiry>**20230127**</SubscriptionExpiry>*)*
* Ensuite aller dans propriétés (clic droit sur le fichier ensuite Propriétés) de chaque fichier *Aspose.Total.lic* et sélectionner **Toujours copier**

|  |
| --- |
|  |

# Vérification à faire :

**Compiler et tester l’application en local et en DEV :**

* **Générer un document pour tester Aspose**
* **Ne plus afficher le message d’erreur MSAL-ERROR**

On ne doit plus avoir ces instructions dans le code :

* master.auth.setRequestHeader
* SetRequestHeader(xhr)
* master.auth.getValidToken()
* $.ajaxSetup