|  |
| --- |
|  |

17/3/2015

|  |
| --- |
|  |
| https://uknowit.uwgb.edu/images/group91/41011/Office365.jpg |
|  |



|  |  |
| --- | --- |
|  | Laboratorio 7 Uso de las API’s de Office 365 (II) y Publicación de Aplicaciones en Windows Azure Active Directory |

**Indice**

[1 Introducción 2](#_Toc414306285)

[2 Ejercicio 1: Crear una Aplicación ASP.NET MVC 5 3](#_Toc414306286)

[2.1 Tarea 1 Crear una lista de Memorias de Projectos 3](#_Toc414306287)

[2.2 Tarea 2 Subir una Memoria de Proyecto a One Drive 5](#_Toc414306288)

[2.3 Tarea 3 : Crear algunos contactos 6](#_Toc414306289)

[2.4 Tarea 4: Crear el Proyecto en Visual Studio 8](#_Toc414306290)

[2.5 Tarea 5 Conectar los servicios 11](#_Toc414306291)

[2.6 Tarea 6 Añadir paquete ADAL 13](#_Toc414306292)

[2.7 Tarea 7: Añadir Ficheros auxiliaries al Proyecto 14](#_Toc414306293)

[2.8 Tarea 8: Modificar el código del controlador Home 14](#_Toc414306294)

[2.9 Tarea 9 Código del Discover Service 17](#_Toc414306295)

[2.10 Tarea 10: Obtener los Contactos 21](#_Toc414306296)

[2.11 Tarea 11 Obtener los Ficheros 25](#_Toc414306297)

[2.12 Crear Nuevo Proyecto 29](#_Toc414306298)

[3 Ejercicio 2: Crear Aplicación Apache Cordova 36](#_Toc414306299)

[3.1 Tarea 1: Crear el Proyecto en Visual Studio 36](#_Toc414306300)

[3.2 Conectamos los servicios de Office 365 37](#_Toc414306301)

[3.3 Agregamos Angular JS a la solución 37](#_Toc414306302)

[3.4 Creación del servicio de acceso a las API’s de Office 365 37](#_Toc414306303)

[3.5 Creación de la Pantalla de Login 37](#_Toc414306304)

[3.6 Creación de la Pantalla de mostrar los contactos 37](#_Toc414306305)

[3.7 Creación del Inicio de la App 37](#_Toc414306306)

[4 Ejercicio 3: Registrar Aplicación en AAD 37](#_Toc414306307)

[5 Ejercicio 4: Crear Liberia Portable 38](#_Toc414306308)

[5.1 Creación del Proyecto de Visual Studio 38](#_Toc414306309)

# Introducción

**Tiempo estimado para este laboratorio:**

45 a 60 minutos

**Objetivos:**

Después de completar este laboratorio usted será capaz de:

* Comprender como registrar aplicaciones en el Directorio Activo de Azure.
* Comprender como dar permisos a nuestras aplicaciones.

**Prerequisitos**

Visual Studio 2013 Update 3

Tools API Office 365

Debes de tener la cuenta de Office 365 vinculada con la subscripción de Microsoft Azure

**Descripción del laboratorio**

En este laboratorio vamos a introducir la manera de poder utilizar las API’s de Office 365. La finalidad de estas API’s es poder utilizar estos servicios en cualquier de nuestros desarrollos, independientemente de la plataforma en la que decidamos desarrollar, independiente del dispositivo y de la plataforma.

En primer lugar vamos a ver como registrar una Aplicación en el Azure Active Directory para nuestra organización, seremos capaz de distinguir los tipos de Aplicaciones disponibles.

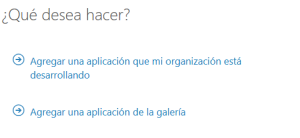
A continuación, veremos cómo poder crear librerías portables para poder código entre diversas aplicaciones.

# Ejercicio 3: Registrar Aplicación en AAD

1.- Entramos en el portal de Administración de Azure

2.- Hacemos clic en el icono de Active Directory en el menú de la izquierda y, a continuación, hacemos clic en el directorio deseado.

3.- En el menú de arriba, hacemos clic sobre “Aplicaciones”. Si no hay ninguna aplicación, se nos mostrará un enlace con Agregar  una Aplicación. Otra posible opción es pulsar sobre el icono de Agregar de la barra interior[](http://blogs.encamina.com/desarrollandosobresharepoint/wp-content/uploads/sites/8/2014/11/Mensaje.png)

4.- Al pulsar sobre el botón nos preguntará el tipo de aplicación que deseamos agregar: una aplicación de nuestra organización o una aplicación de la galería de Azure.[](http://blogs.encamina.com/desarrollandosobresharepoint/wp-content/uploads/sites/8/2014/11/Selecci%C3%B3n-Aplicacion.png)

5.- En caso de seleccionar una aplicación de nuestra organización, deberemos indicar si es una aplicación web y/o API web (por defecto) o la aplicación cliente nativo que representa una aplicación que se instala en un dispositivo, como un teléfono o un ordenador.

6.- En la página de propiedades de la aplicación, proporcionamos la URL de sesión y App ID URI para la aplicación Web (o simplemente la URI de redirección para una aplicación de cliente nativo). A continuación, hacemos clic en la casilla de verificación en la esquina inferior derecha de la página.

 7.- Una vez se agrega la aplicación, ya se puede comenzar la actualización de su aplicación para permitir a los usuarios iniciar sesión, las API de acceso web en otras aplicaciones, o configurar aplicaciones multi-tenant (que permite a otras organizaciones a acceder a su solicitud).

# Ejercicio 4: Crear Liberia Portable

## Creación del Proyecto de Visual Studio

Siguiendo los siguientes pasos crearemos un Nuevo Proyecto Class Library (Portable

1. Arrancamos “Visual Studio 2013”.

2. Seleccionamos “Archivo/Nuevo/Proyecto” del menu principal.

3. En el diálogo de “Nuevo Proyecto”:

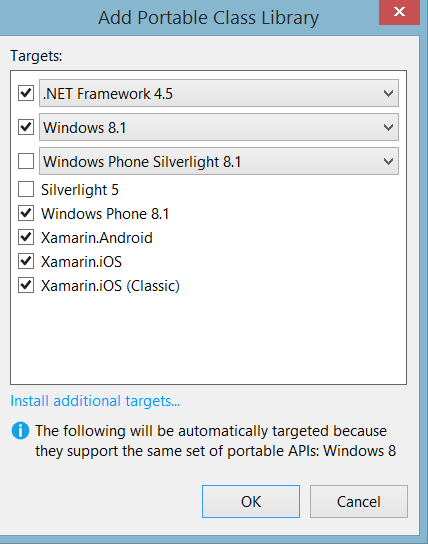
1. Seleccionamos “C#/ Windows Descktop/”.

2. Click “Portable Class (Library)”.

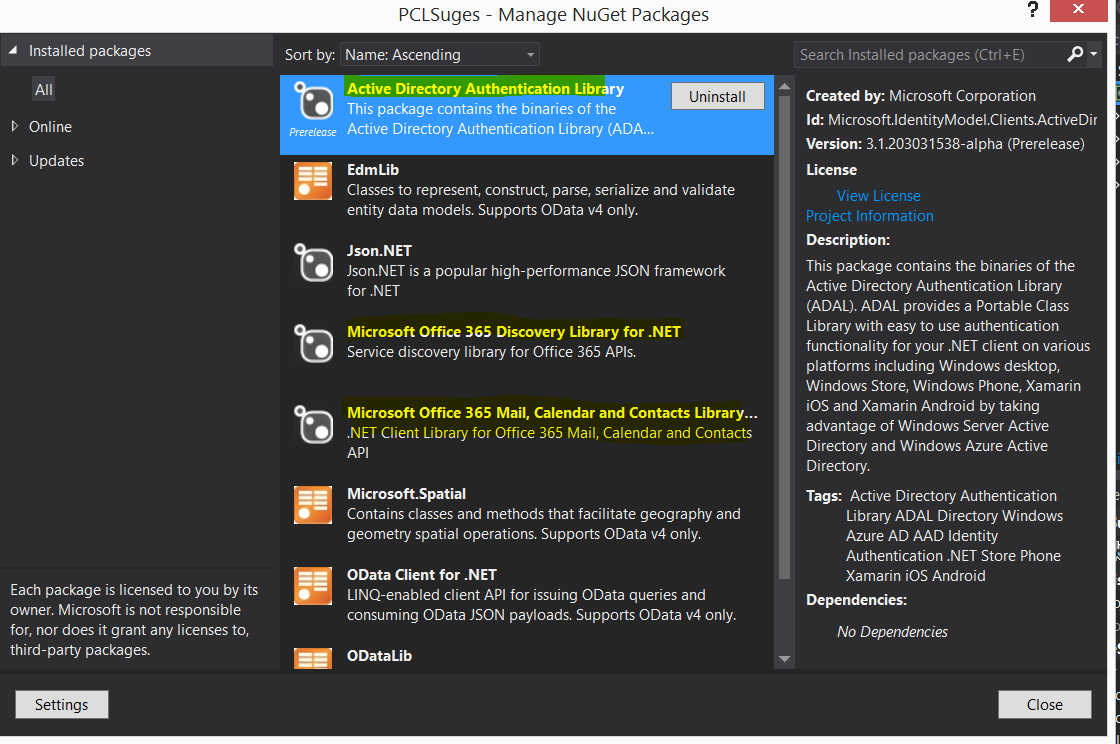
3. Llamamos al proyecto “PCLSuges”.

4. Click “OK”.

5. Seleccionamos los destinos donde nuestra librería se podrá ejecutar, tal y como se muestra en la siguiente imagen (en caso de no tener instalado Xamarin no se muestran las opciones: Xamarin.Android, Xamarin.iOS, Xamarin.IOS (Classic):



5.- Instalamos los siguientes paquetes de Nuget:



6.- Creamos una clase “MyContacts.cs” con el siguiente código:

public class MyContacts

{

public string Id { get; set; }

public string GivenName { get; set; }

public string Surname { get; set; }

public string DisplayName { get; set; }

public string CompanyName { get; set; }

public string BusinessPhone1 { get; set; }

public string HomePhone1 { get; set; }

public string EmailAddress1 { get; set; }

}

7.- Creamos una clase Contacts donde se dispondrán los métodos necesarios para autenticarse contra Office 365 y mostrar los Contactos, para ello utilizamos el siguiente código:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.IdentityModel.Clients.ActiveDirectory;

using Microsoft.Office365.Discovery;

using Microsoft.Office365.OutlookServices;

namespace PCLSuges

{

public class Contacts

{

public static string ClientId = "[Client ID]";

public static Uri ReturnUri = new Uri("URI");

public const string CommonAuthority = "https://login.windows.net/Common";

public const string DiscoveryResourceId = "https://api.office.com/discovery/";

public static AuthenticationContext \_authenticationContext { get; set; }

private static async Task<string> GetTokenHelperAsync(AuthenticationContext context, string resourceId, IAuthorizationParameters parent)

{

string accessToken = null;

AuthenticationResult result = await context.AcquireTokenAsync(resourceId, ClientId, ReturnUri, parent);

accessToken = result.AccessToken;

return accessToken;

}

public static async Task<OutlookServicesClient> CreateOutlookClientAsync(string capability, IAuthorizationParameters parent)

{

try

{

string authority = null;

authority = CommonAuthority;

\_authenticationContext = new AuthenticationContext(authority);

var discoveryClient = new DiscoveryClient(

async () => await GetTokenHelperAsync(\_authenticationContext, DiscoveryResourceId, parent));

var result = await discoveryClient.DiscoverCapabilityAsync("Contacts");

var client = new OutlookServicesClient(

result.ServiceEndpointUri,

async () =>

await GetTokenHelperAsync(\_authenticationContext, result.ServiceResourceId, parent));

return client;

}

catch (Exception)

{

if (\_authenticationContext != null && \_authenticationContext.TokenCache != null)

\_authenticationContext.TokenCache.Clear();

return null;

}

}

public static async Task<List<MyContacts>> GetContacts(IAuthorizationParameters parent)

{

var outlookClient = await CreateOutlookClientAsync("Contacts", parent);

var contactList = new List<MyContacts>();

var contactsResults = await outlookClient.Me.Contacts.ExecuteAsync();

foreach (var contact in contactsResults.CurrentPage.OrderBy(c => c.Surname))

{

contactList.Add(new MyContacts

{

Id = contact.Id,

GivenName = contact.GivenName,

Surname = contact.Surname,

DisplayName = contact.Surname + ", " + contact.GivenName,

CompanyName = contact.CompanyName,

EmailAddress1 = contact.EmailAddresses.FirstOrDefault().Address,

BusinessPhone1 = contact.BusinessPhones.FirstOrDefault(),

HomePhone1 = contact.HomePhones.FirstOrDefault()

});

}

return contactList;

}

}

}

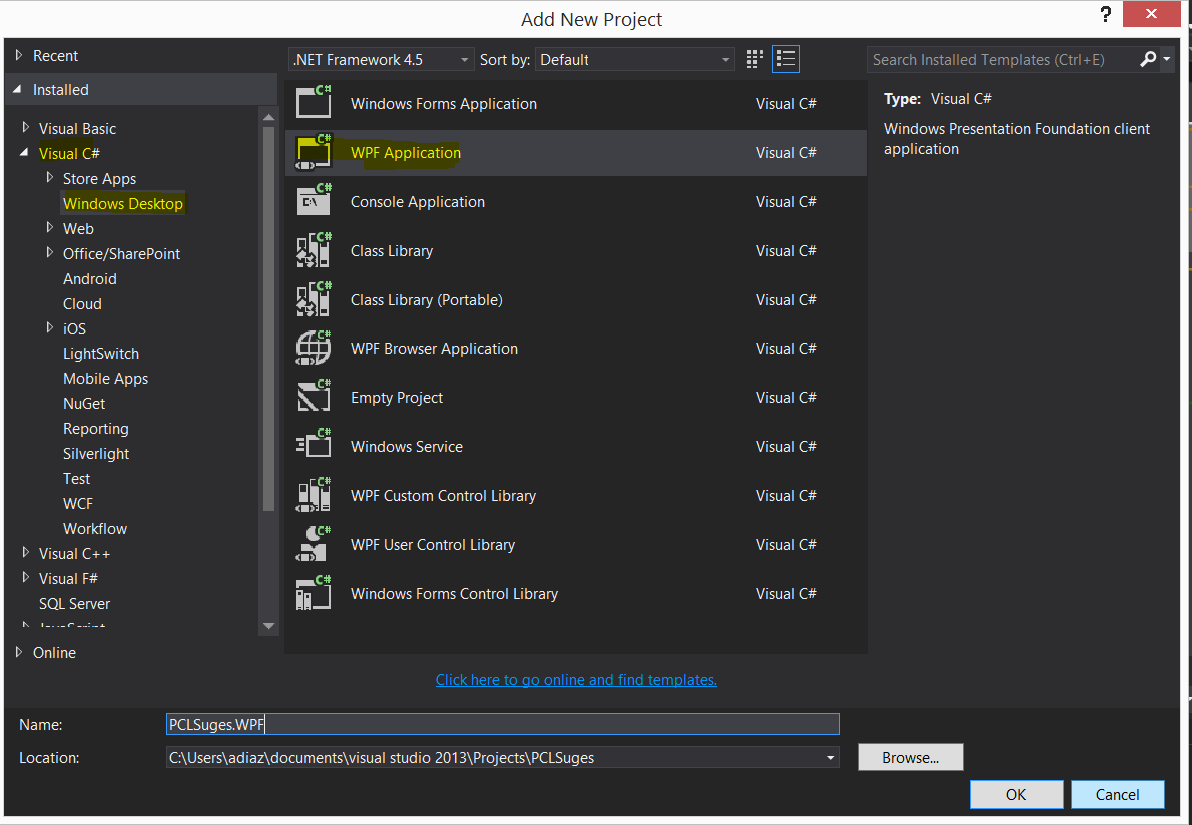
Modificaremos los valores ClientId y ReturnUri serán los valores de nuestra aplicación que previamente hemos registrado en el Azure Active Directory.

## Tarea 2 Creación del Proyecto de WPF

1.- En el Explorador de Soluciones, botón derecho, agregrar Nuevo Proyecto

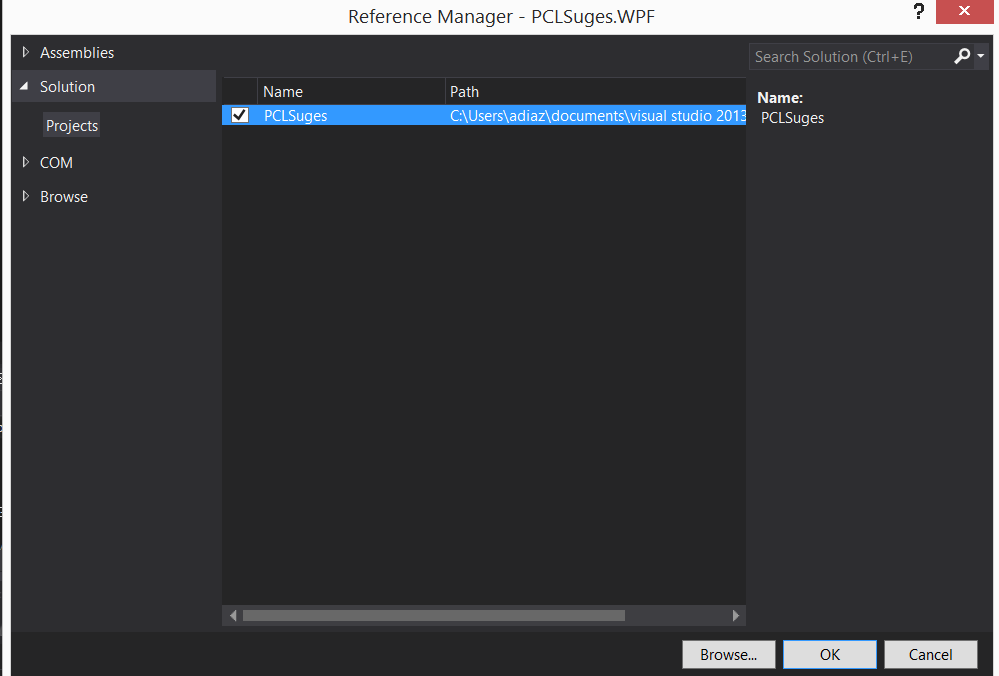
2.- En la pantalla que se muestra seleccionaremos “Visual C#/ Windows Desktop/”

3.- Seleccionamos “WPF Application”, introducimos “PCLSuges.WPF” y pulsamos OK

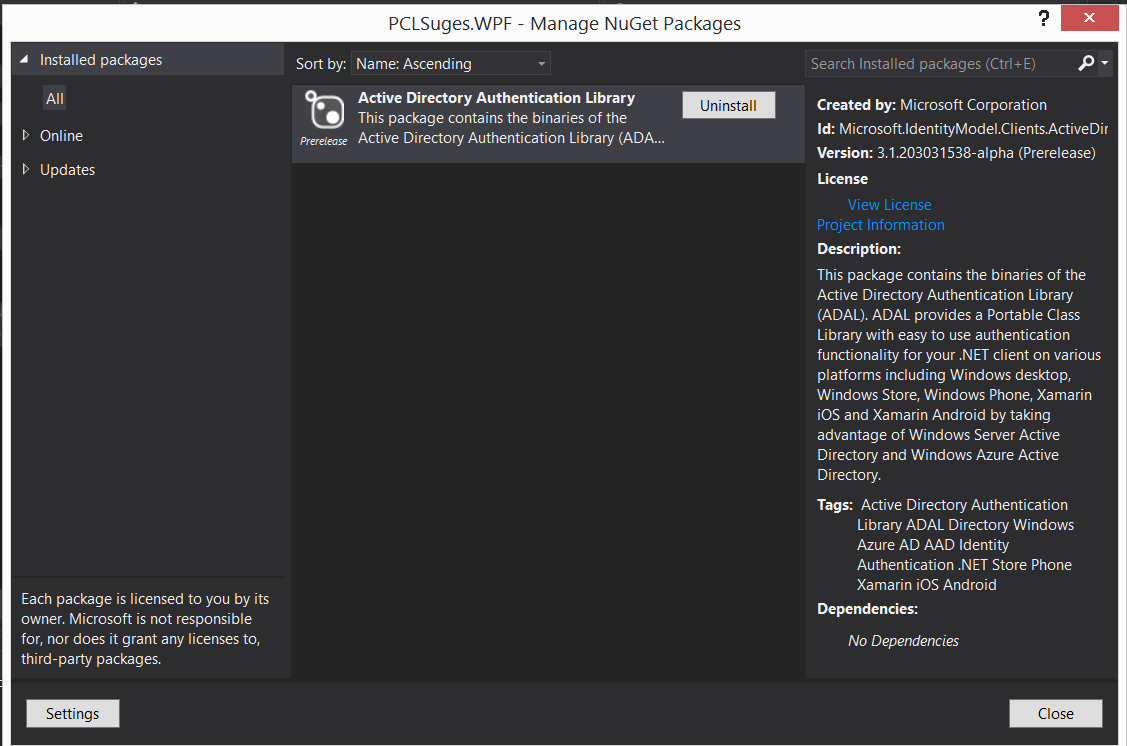


4.- Agregamos referencia a la biblioteca anterior, pulsar botón derecho en el explorador de soluciones, Add-> Reference

5.- En la pantalla que se muestra, seleccionar Solution-> Proyecto y marcar la opción disponible:



6.- Agregar el paquete de Nuget de Active Directory Authentication Library



7.- Ir a MainWindows.xaml y substituir todo el contenido que hay entre “Grid” por el siguiente código:

<Grid Margin="12,12,12,0">

<Grid.Resources>

<VisualBrush x:Key="SearchHint" Stretch="None" AlignmentX="Left" AlignmentY="Top">

<VisualBrush.Transform>

<TranslateTransform X="4" Y="7" />

</VisualBrush.Transform>

<VisualBrush.Visual>

<Grid>

<TextBlock Text="Enter a user's alias" FontWeight="Normal" Foreground="Gray"

FontSize="12" TextAlignment="Justify"/>

</Grid>

</VisualBrush.Visual>

</VisualBrush>

</Grid.Resources>

<StackPanel Grid.Row="0" Margin="5,5,5,0" VerticalAlignment="Top" Height="295">

<Button Grid.Column="1" Content="Cargar Contactos" HorizontalAlignment="Right" Margin="0,10,-0.2,0" VerticalAlignment="Top" Width="201" RenderTransformOrigin="-0.013,0.15" Click="Search" Height="34" IsDefault="True" TabIndex="2"/>

<DataGrid x:Name="SearchResults" Margin="0,0,0,0" AutoGenerateColumns="False" ScrollViewer.CanContentScroll="True" ScrollViewer.HorizontalScrollBarVisibility="Disabled"

IsReadOnly="True" MinWidth="350" CanUserAddRows="False" CanUserDeleteRows="False" Height="180" GridLinesVisibility="None" Background="#FFFFFFFF">

<DataGrid.Columns>

<DataGridTextColumn Header="First Name" Binding="{Binding GivenName}" Width="1\*"/>

<DataGridTextColumn Header="Last Name" Binding="{Binding Surname}" Width="1\*"/>

<DataGridTextColumn Header="UPN" Binding="{Binding DisplayName}" Width="3\*"/>

<DataGridTextColumn Header="Phone" Binding="{Binding BusinessPhone1}" Width="1\*"/>

</DataGrid.Columns>

</DataGrid>

</StackPanel>

</Grid>

8.- Ahora en el code-behind de ese fichero añadiremos el siguiente código:

public MainWindow()

{

InitializeComponent();

List<MyContacts> results = new List<MyContacts>();

SearchResults.ItemsSource = results;

}

private async void Search(object sender, RoutedEventArgs e)

{

await LoadData();

}

private async Task LoadData()

{

List<MyContacts> results = await Contacts.GetContacts(new AuthorizationParameters(PromptBehavior.Auto, this.Handle));

SearchResults.ItemsSource = results;

}

public IntPtr Handle

{

get

{

var interopHelper = new WindowInteropHelper(this);

return interopHelper.Handle;

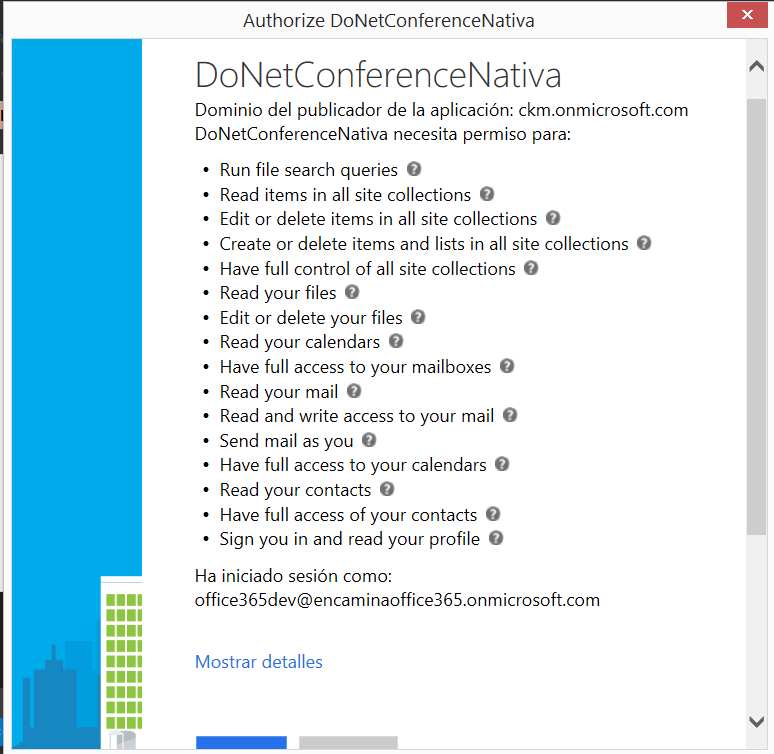
}

}

9.- Ahora pulsamos F5

10.- Apretamos sobre el bóton de “Cargar Contactos”

11.- Al mostrar la pantalla, pulsaremos Ok



12.- Si todo ha ido correctamente se mostrarán los Contactos en el Grid:

