Azure Fast Start for Mobile Application Development

Module 09: Azure Mobile Apps

Student Lab Manual

Instructor Edition (Book Title Hidden Style)

Version 1.0

**Conditions and Terms of Use**

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2015 Microsoft Corporation. All rights reserved.

**Copyright and Trademarks**

© 2015 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at  
<http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx>

Active Directory, Azure, Microsoft, Internet Explorer, Visual Studio, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Contents

[Lab 1: Backend As a Service 1](#_Toc433038946)

[Exercise 1: Create a Mobile App Backend 3](#_Toc433038947)

[Exercise 2: Send push notification 27](#_Toc433038948)

[Lab 2: Authentication 39](#_Toc433038949)

[Exercise 1: Configure authentication for your Mobile App 40](#_Toc433038950)

[Exercise 2: Add authentication in your Mobile App 46](#_Toc433038951)

[Lab 3: Offline Data Sync in Mobile App 54](#_Toc433038952)

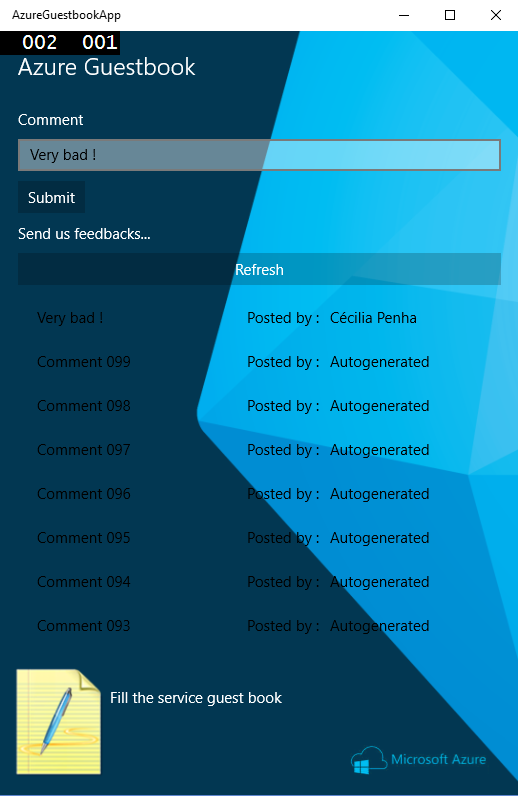
[Exercise 1: Add offline sync data 55](#_Toc433038953)

# Lab 1: Backend As a Service

#### Introduction

In this lab, you will learn how to add a cloud-based backend service to a Windows Universal App by using a Microsoft Azure Mobile App backend, and push notifications.

The following are the screen shots from the completed app:



#### Objectives

After completing this lab, you will be able to:

* Create a simple Windows Universal App.
* Create a new Azure Mobile App backend as a service.

#### Estimated Time to Complete This Lab

60 minutes

#### Prerequisites

All resources allocated in Azure will follow these Naming Rules:

* Prefix: **W10AZFS-GB**: Windows 10 Azure Fast Start – Guestbook.
* You can use your own or ignore it as long as you stay consistent.

Before starting:

* Ensure you have an active Azure account. If you do not have an account, you can sign up for an Azure trial and get up to ten free mobile apps that you can keep using even after your trial ends. For details, see Azure Free Trial.
* Copy the AzureGuestbook snippets to the Visual Studio code snippets folder: Documents\Visual Studio\Code Snippets
* Ensure you have a Visual Studio 2015 with the latest Azure SDK installed.

Exercise 1: Create a Mobile App Backend

#### Objectives

In this exercise, you will learn:

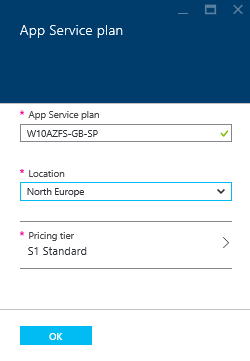
* How to create a new Azure Mobile App.
* How to create and publish a new Mobile App Server project (as **Backend**).
* How to create and run a Mobile App Client project.

#### Scenario

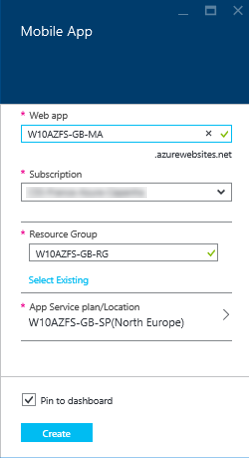
In this scenario, we will create a guestbook App that will store user comments. This simple application will be used as a base and enhanced using Mobile Services features. We will take advantage of this to build from the very beginning and analyze in detail with the audience about a full app lab from start to end.

Task 1: Create a New Azure Mobile App in Azure

1. Log onto the [Azure Portal](https://portal.azure.com/).
2. On the top-left side of the window, click the **+NEW** button, navigate to **Web + Mobile** > **Mobile App**.
3. Provide the **App web** **W10AZFS-GB-MA.azurewebsites.net** for the Mobile App backend.
4. In the **Resource Group** box, enter the name **W10AZFS-GB-RG**.
5. The default **App Service plan** will be selected. Change your **App Service plan** by clicking the **App Service Plan** > **+ Create New**.
6. Provide the name **W10AZFS-GB-SP** of the new **App Service plan**.
7. Select the **Location**: **North Europe**.
8. Click the **Pricing tier** and select an appropriate pricing tier free or shared for the service. After you have selected the **Pricing tier**, click the **Select button**.
9. Back on the **App Service plan** blade, click **OK**.



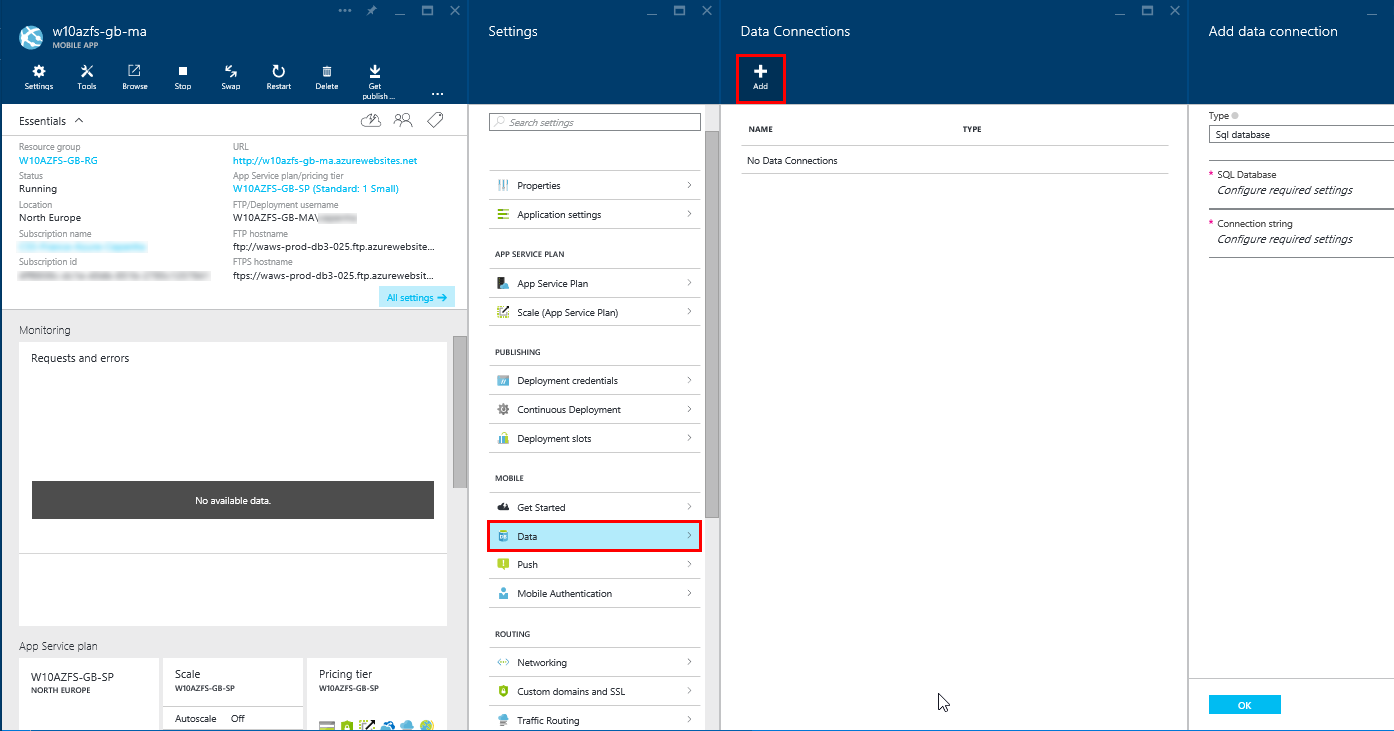
1. Back on the **Mobile App** blade, click **Create**.



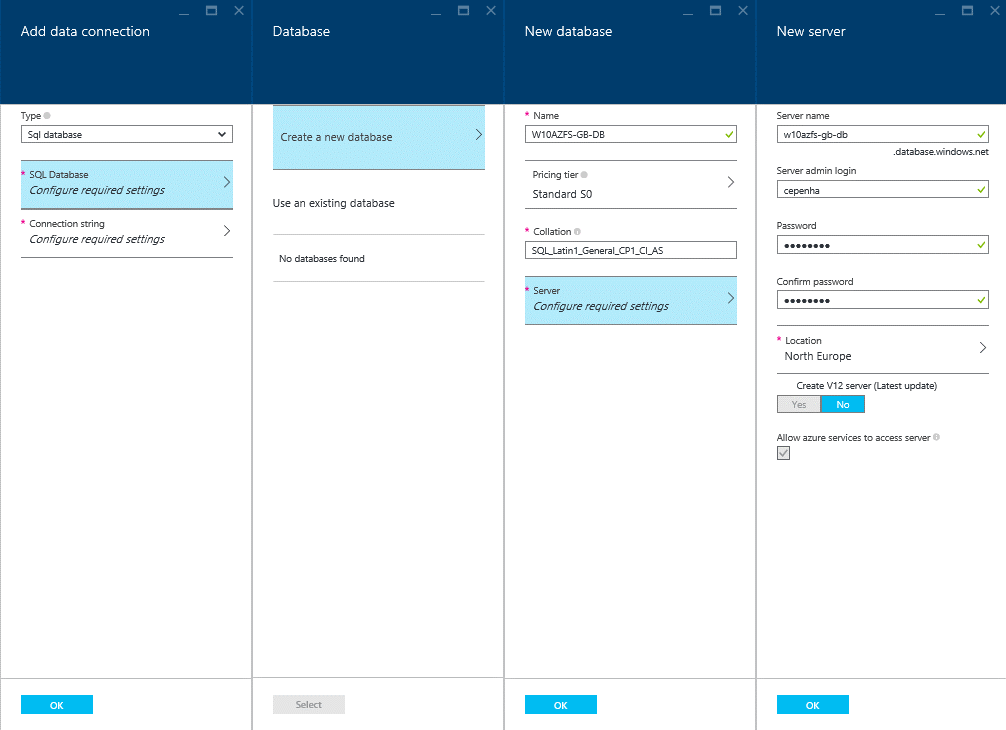
**Note** This creates a Mobile App backend where you will later deploy your server project. Provisioning a Mobile App backend can take some time. After the Mobile App backend is provisioned, the portal will open the Settings blade for the Mobile App backend.

**Note** As part of this lab, you will create a new SQL Database instance and server for your Azure Mobile App. You can reuse this new database and administer it as you would do for any other SQL Database instance. If you already have a database in the same location as the new mobile app backend, you can instead choose **Use an existing database** and then select that database. The use of a database in a different location is not recommended because of additional bandwidth costs and higher latencies.

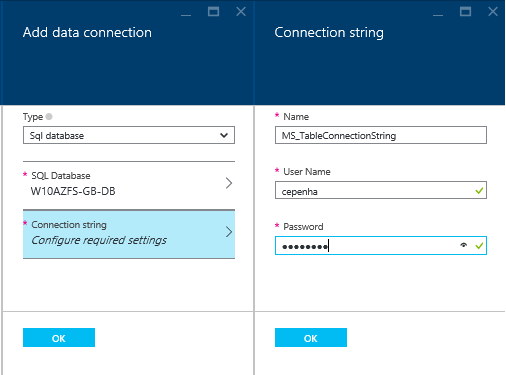
1. From the new Mobile App **W10AZFS-GB-MA** blade, click **Settings** > **Data** > **+Add**.



1. In the **Add data connection** blade, click **SQL Database** > **Configure required settings** > **Create a new database**. Enter the name **W10AZFS-GB-DB** of the new database in the **Name** field, and then click **Server**.
2. In the **New server** blade, enter a unique Server name **W10AZFS-GB-DB**, and provide a suitable Server admin login and Password. Ensure the **Allow azure services to access server** check box is selected.
3. Then, click **OK**.
4. On the **New database** blade, click **OK**.



1. Back on the **Add data connection** blade, select **Connection string**, enter the **User Name** and **Password** that you just provided while creating the database. If you use an existing database, provide the credentials for that database.
2. Then, click **OK**.

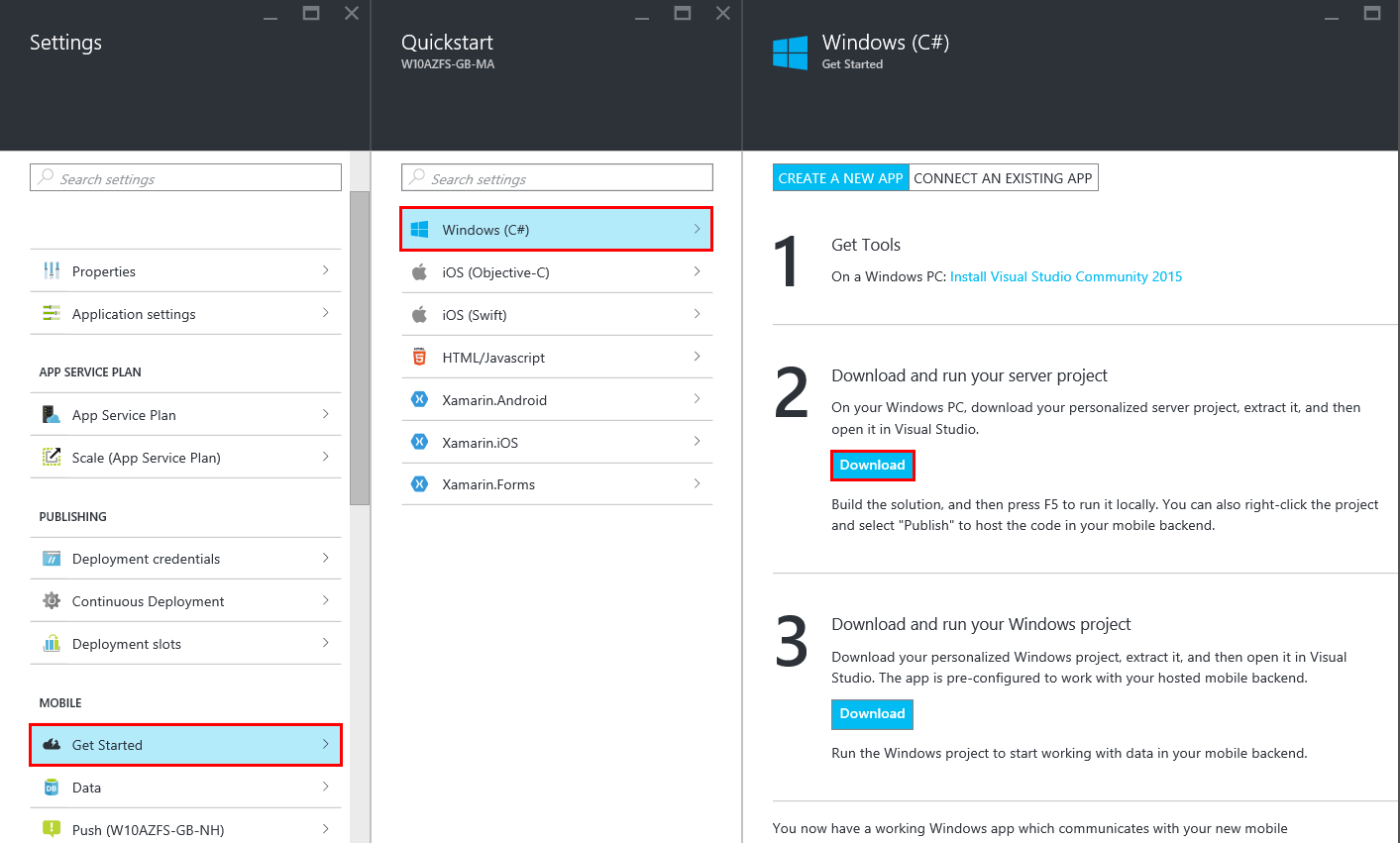


1. To add the connection, click **OK**.

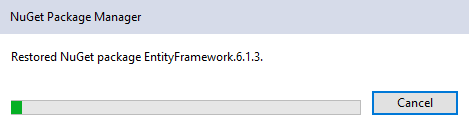
You have now provisioned an Azure Mobile App in Azure that can be used by your mobile client applications. Next, you will create a server project for a simple **Comments list** backend and publish it to Azure.

Task 2: Create a New Azure Mobile App Server Project

1. Log onto the [Azure Portal](https://portal.azure.com/).
2. On the top-left side of the window, click **Browse** > **Mobile App**.
3. From the Mobile App **W10AZFS-GB-MA** blade, click **Settings** > **Get Started** > **Windows (C#)**, download thestarter server project.

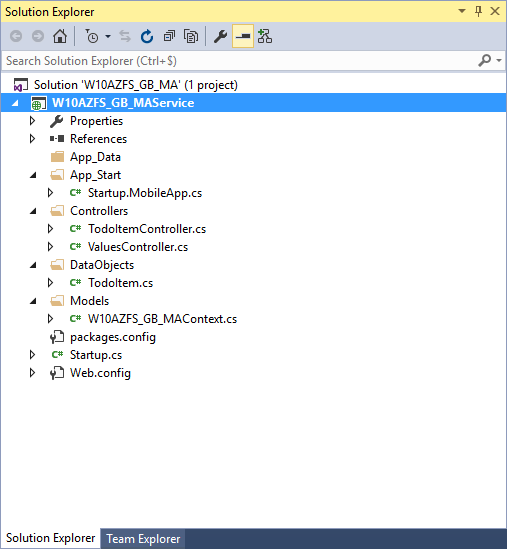


1. Open the server project **W10AZFS\_GB\_MAService**, and build to restore all NuGet packages.



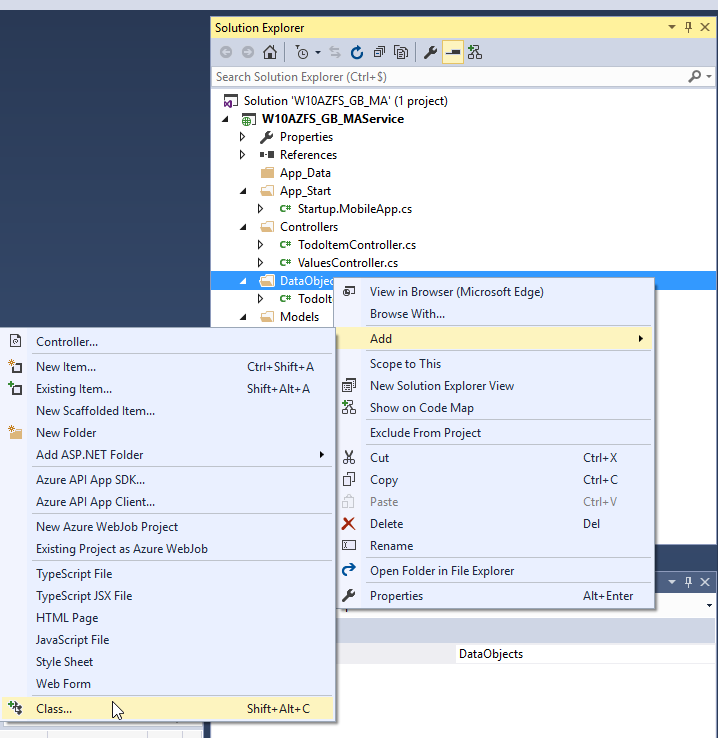
**Note** The project is provisioned with the necessary NuGet packages and personalized with the Azure Mobile App.

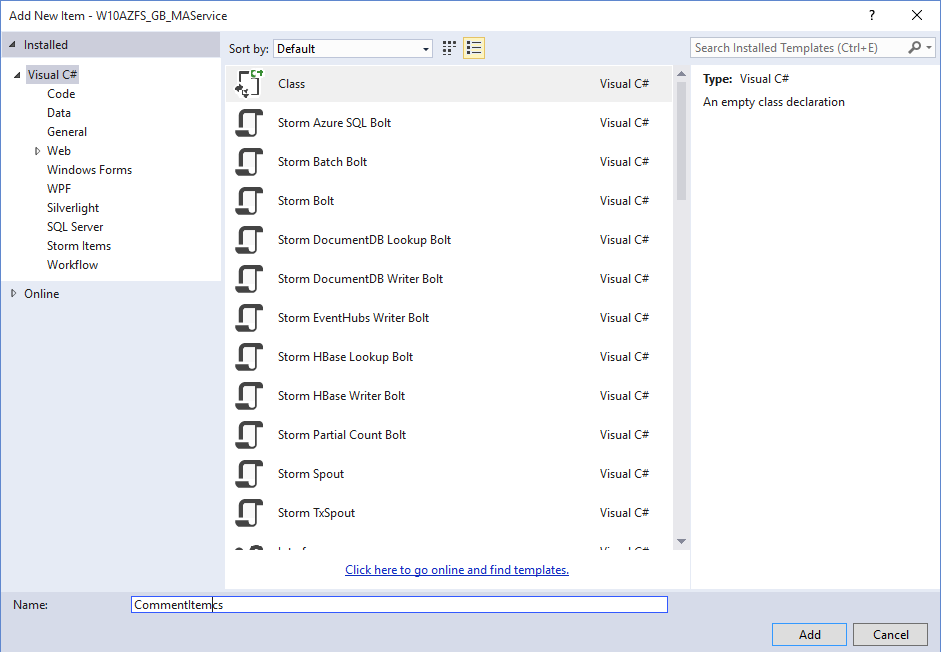
1. Examine the project.



**Note** The project contains starter code for the TodoItem mobile service. We will add new data in the **DataObjects** folder.

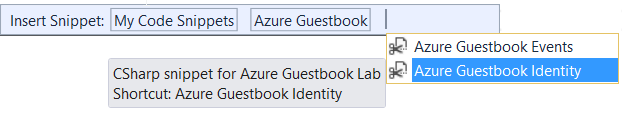
1. Create a new class **CommentItem** in the **DataObjects** folder.





1. Replace the whole code of the **CommentItem** class using the snippet **Azure Guestbook Identity**.

To insert the snippet, press Ctrl+K, then X, and then navigate to the **Azure Guestbook Identity** snippet shown as follows:



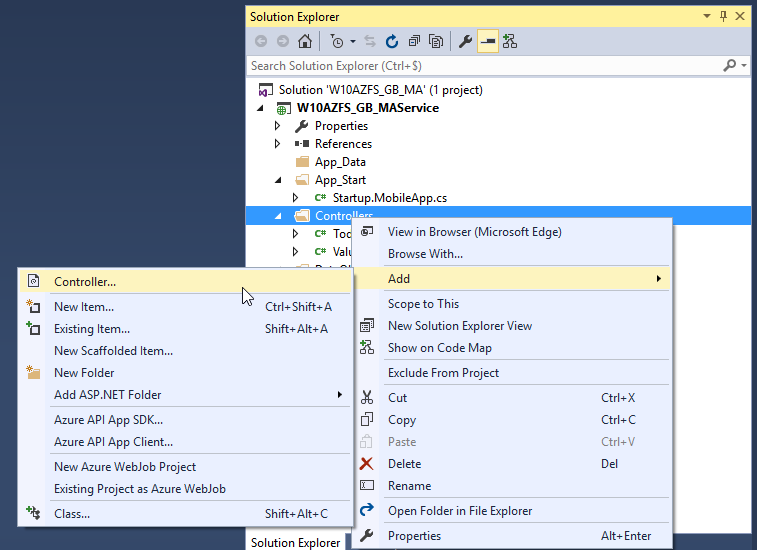
The code should be as follows:

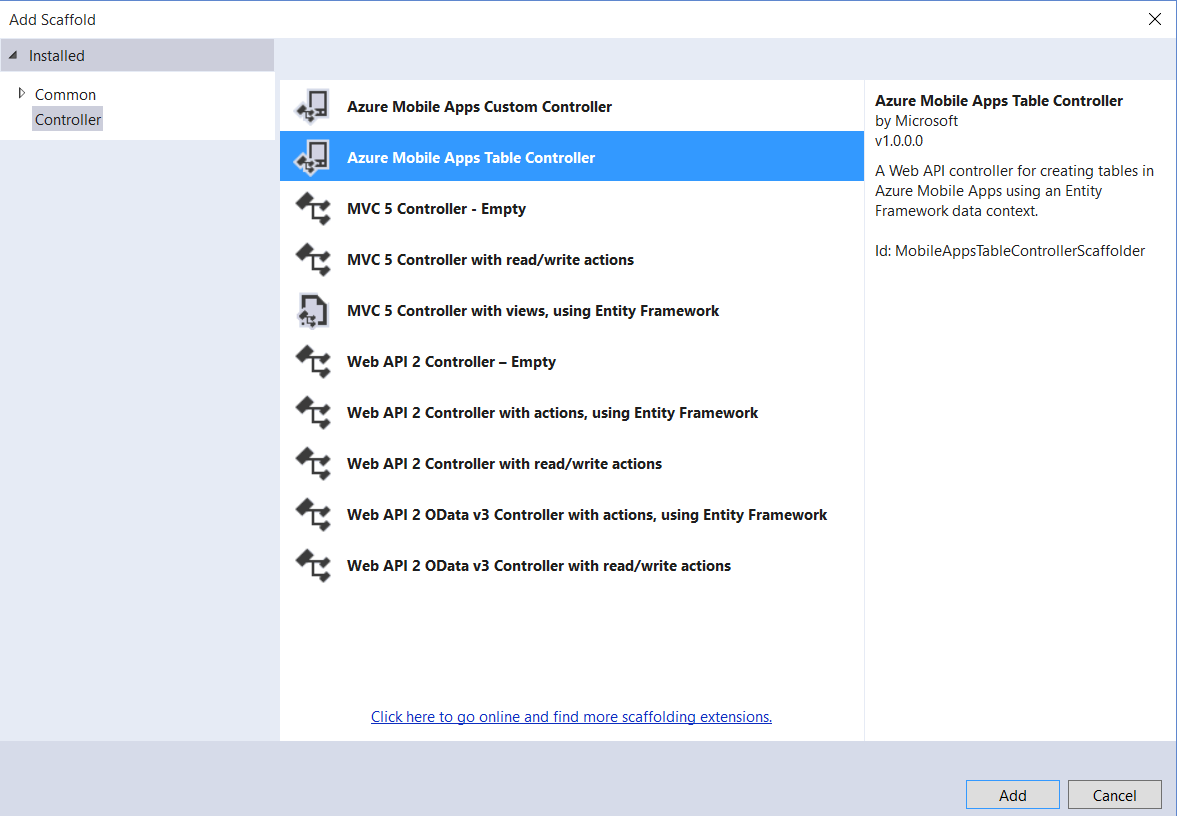
|  |
| --- |
| using Microsoft.Azure.Mobile.Server;  namespace W10AZFS\_GB\_MAService.DataObjects  {  public class CommentItem : EntityData  {  public string Description { get; set; }  public string Name { get; set; }  }  } |

The **CommentItem** class represents the data type and maps onto a single database table. The class inherits from **EntityData**, which means this data object maps onto a mobile service table and can be used as a template type with the table controller, TableController<TEntityData>. Entity Framework is a useful data access library that promotes code-first data development. For more information, see: [Entity Framework](http://msdn.microsoft.com/en-us/data/ef.aspx)

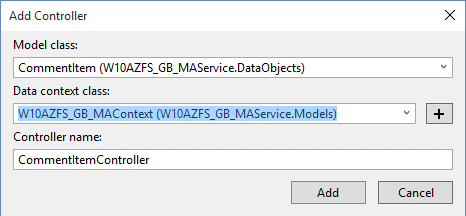
It works well with Web API and is the default data access framework for mobile services with a .NET backend.

1. Add a new Controller (**Azure Mobile Apps Table Controller**).





1. Fill the wizard by selecting the newly created entity (Comment), and the existing **Data context class** shown as follows:



1. In the **Controllers** node, open the **CommentItemController** class and analyze. Correct the following issue when adding Controller.

**Important** There is a known issue on the **Initialize** method. Remove the third parameter of the instantiation of EntityDomainManager. The code should be as follows:

|  |
| --- |
| protected override void Initialize(HttpControllerContext controllerContext)  {  base.Initialize(controllerContext);  W10AZFS\_GB\_MAContext context = new W10AZFS\_GB\_MAContext();  DomainManager = new EntityDomainManager<CommentItem>(context, Request);  } |

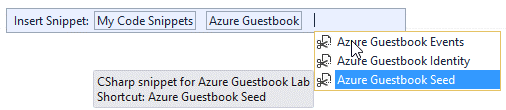
The **CommentItemController** class is a Web API controller. It is a special kind of Web API controller because it inherits from TableController that is designed to work with mobile services tables. The class contains methods that implement a Representational State Transfer (REST) API, which your mobile service clients call to create, update, read, or delete data. REST is a preferred design pattern for web services and is the design pattern used by mobile services table controllers. The comment lines indicate the type of HTTP requests that this Web API accepts, all of which are standard to a REST API. The class also contains an Initialize method that sets up some context objects and creates the entity domain manager for Entity Framework.

1. Open the **MobileServiceContext** class in the **Models** folder and analyze.

This is a class that is required to enable Entity Framework. You would not need to modify this class in this lab. You can have at most one data context class in each mobile service.

1. Provision some sample data. Open the **Startup** file in the folder **App\_Start**, locate the class **MobileAppInitializer**, and in **Seed** method, add the following code:

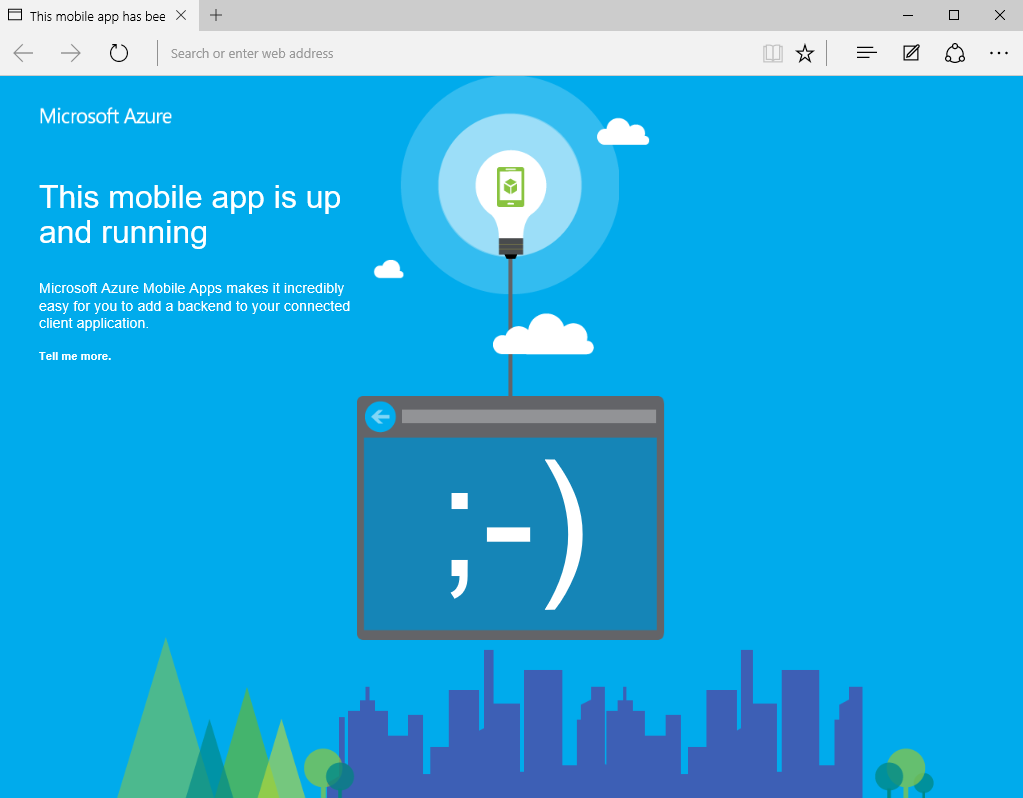
To insert the snippet, press Ctrl+K, then X, and then navigate to the **Azure Guestbook Seed** snippet shown as follows:



The code should be as follows:

|  |
| --- |
| protected override void Seed(W10AZFS\_GB\_MAContext context)  {  // Autogenerating 100 comments  for (int i = 0; i < 100; i++)  {  context.Set<CommentItem>().Add(new CommentItem()  {  Id = i.ToString("D3"),  Name = "Autogenerated",  Description = "Comment " + i.ToString("D3")  });  }  List<TodoItem> todoItems = new List<TodoItem>  {  new TodoItem { Id = Guid.NewGuid().ToString(), Text = "First item", Complete = false },  new TodoItem { Id = Guid.NewGuid().ToString(), Text = "Second item", Complete = false },  };  foreach (TodoItem todoItem in todoItems)  {  context.Set<TodoItem>().Add(todoItem);  }  base.Seed(context);  } |

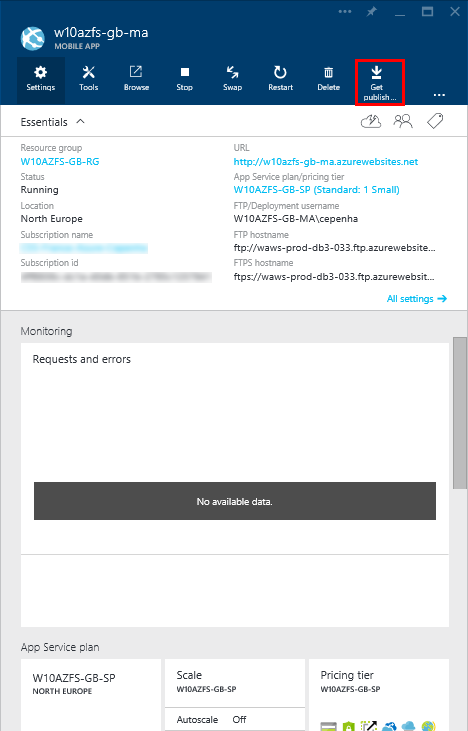
1. Build and run the Mobile App.



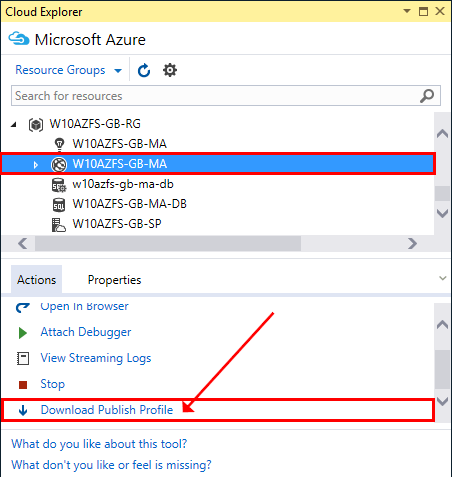
1. Check the creation of the table **CommentItem** from the local URL path: **/tables/CommentItem**

Task 3: Publish the Azure Mobile App Server Project

1. From Azure Portal, get the Publish profile.

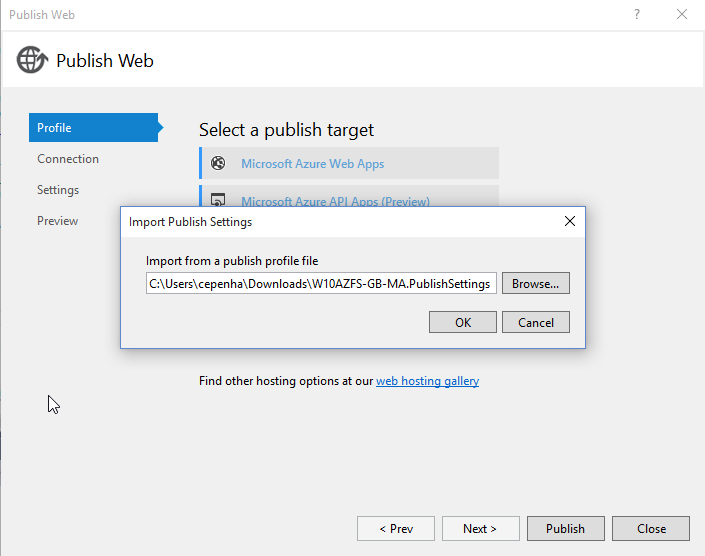


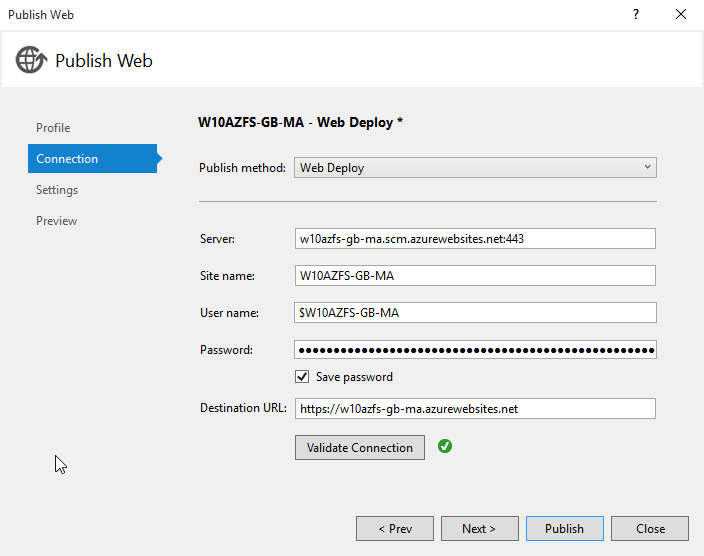
Or, you can download it from the Cloud Explorer view shown as follows:



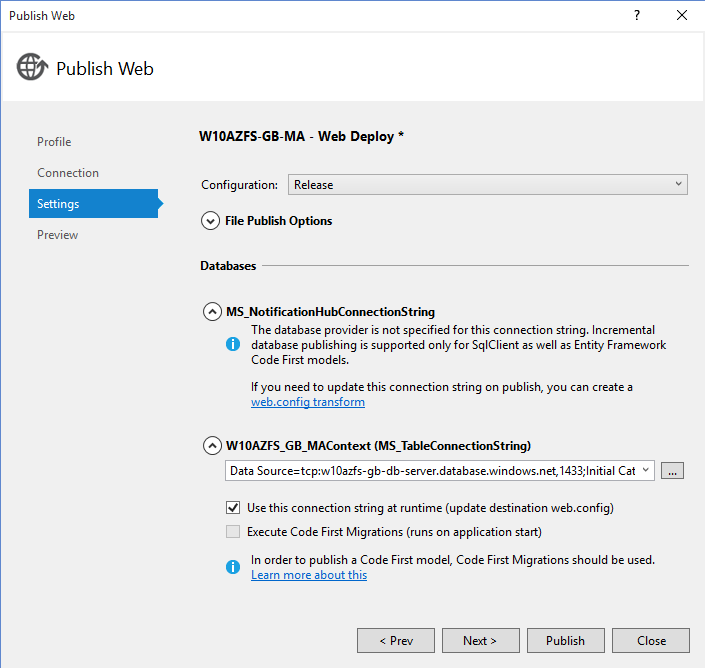
1. In the Visual Studio, right-click the project, click **Publish** > **Publish Web** > **Import**

Then, import the **Publish** profile in the **Publish** web settings of your project shown as follows:





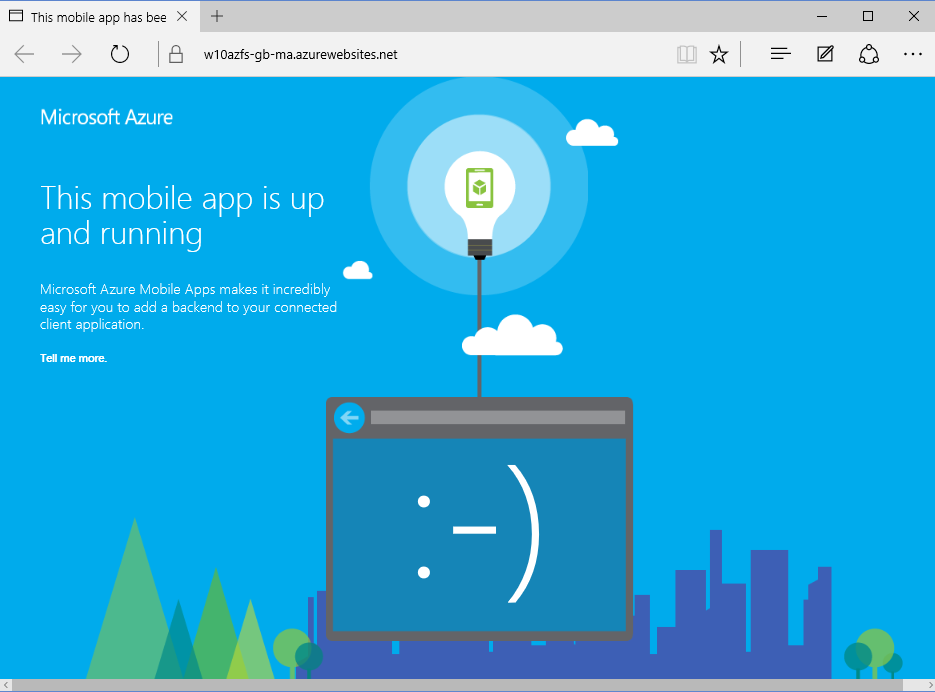
1. Choose the **Validate Connection** button. If the connection succeeds, a green check mark appears.
2. Click **Next**.



1. If you plan to debug your mobile App, choose a debug configuration, and then click **Next**.
2. Click **Publish**.

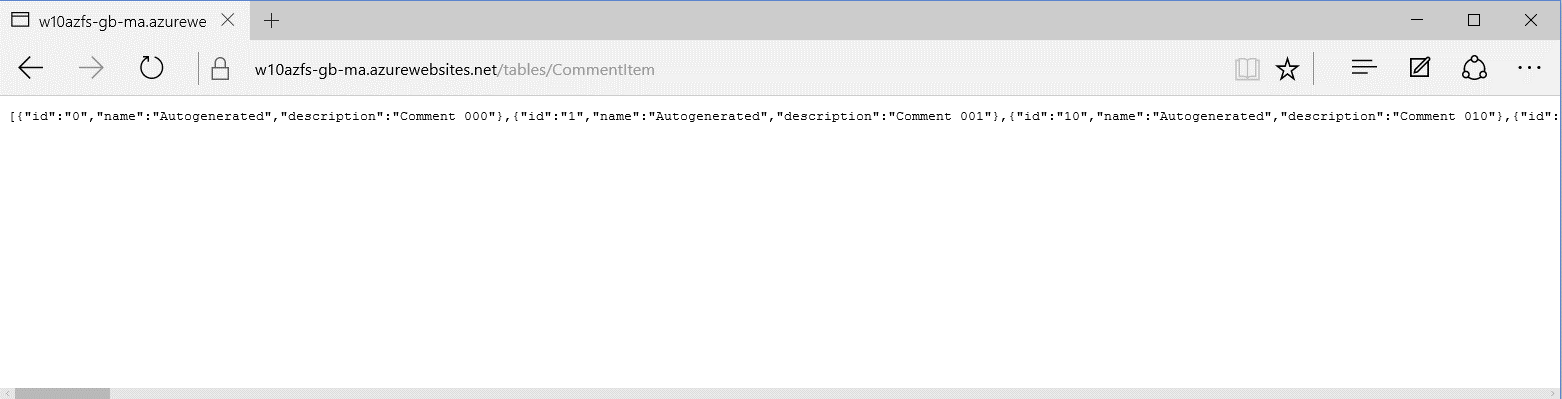
When your Mobile App backend has published successfully, you will see a landing page indicating success.





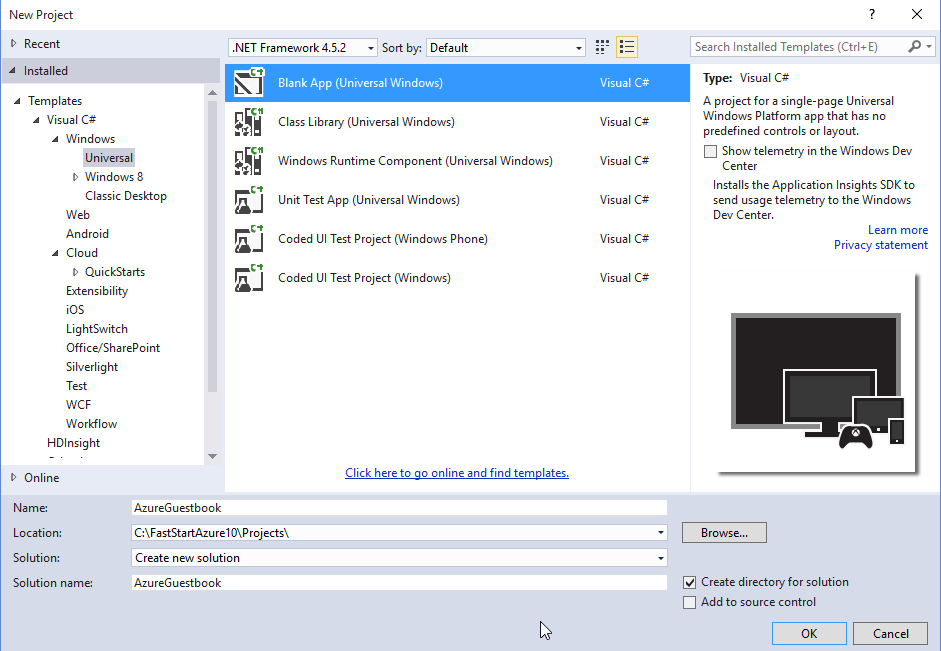
**Tip** When you publish a Mobile App project, the browser opens with the same welcome page that appears when you start the mobile service project on your local machine.

1. Check the creation of the table **CommentItem** from the link: <https://w10azfs-gb-ma.azurewebsites.net/tables/CommentItem>



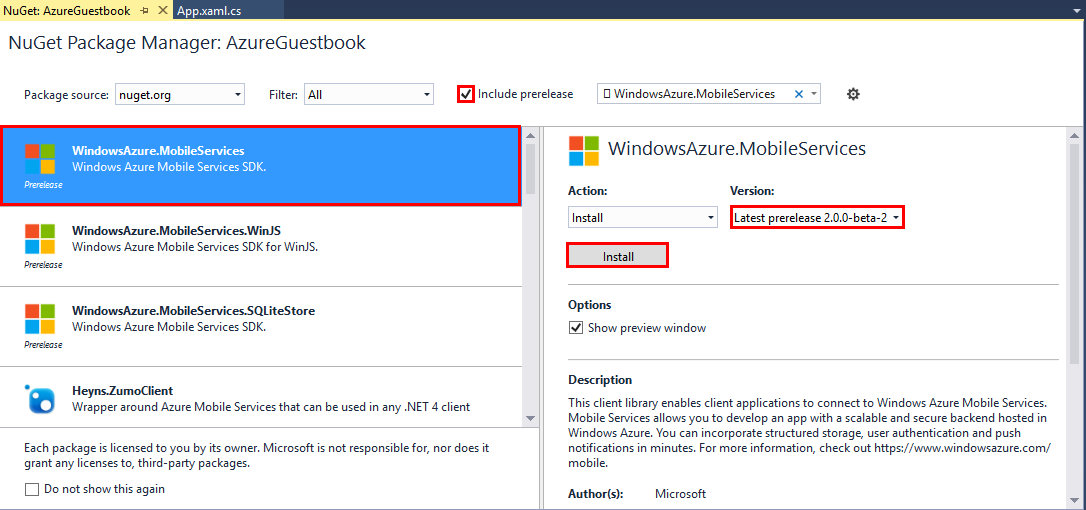
Task 4: Create and Run a New Azure Mobile App Client Project

1. In Visual Studio, create a new **Blank App** project under **Visual C#** > **Windows** > **Universal** templates, named **AzureGuestbook**.

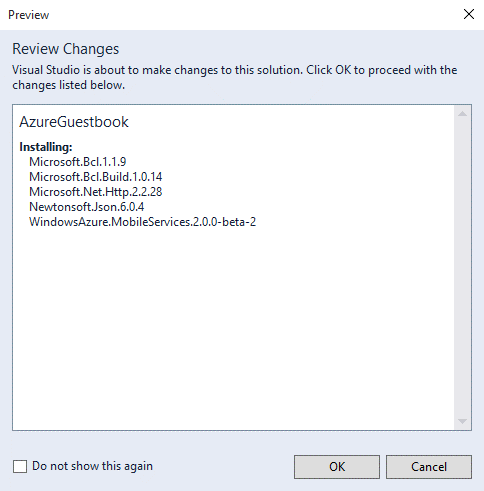


1. Install the **NuGet Packages** and the dependencies.

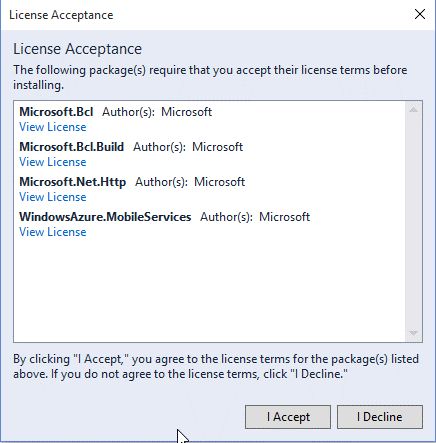
* WindowsAzure.MobileServices (SDK).



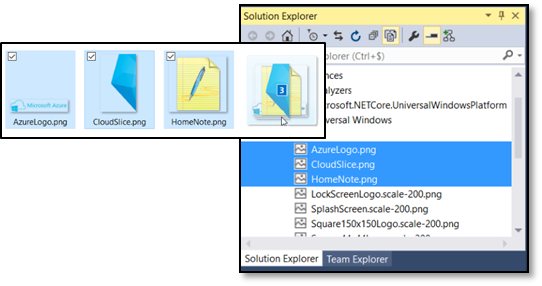
1. You will be prompted to install dependencies.



1. Click **OK** andacceptthe license.



1. Drag pictures from the **Asset** folder of the lab to the **Asset** folder on the solution.



1. Open the **App.xaml.cs** file.
2. Add a namespace **Microsoft.WindowsAzure.MobileServices**

|  |
| --- |
| using Microsoft.WindowsAzure.MobileServices; |

1. Copy and paste the following code at the beginning of the class **App**:

|  |
| --- |
| // This MobileServiceClient has been configured to communicate with the Azure Mobile Service and  // Azure Gateway using the application key. You're all set to start working with your Mobile Service!  public static MobileServiceClient MobileService = new MobileServiceClient(  "https://w10azfs-gb-ma.azurewebsites.net",  "",  ""  ); |

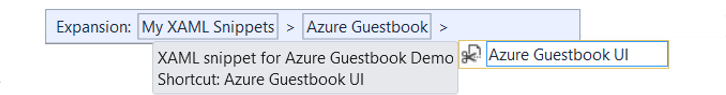
This MobileServiceClient will be configured to communicate with the Azure Mobile Service and Azure Gateway using the application key: <https://w10azfs-gb-ma.azurewebsites.net>

You are all set to start working with your Mobile Service.

1. Open or double-click the **MainPage.xaml**.
2. Replace the code shown as follows using the snippet XAML **Azure Guestbook UI**:



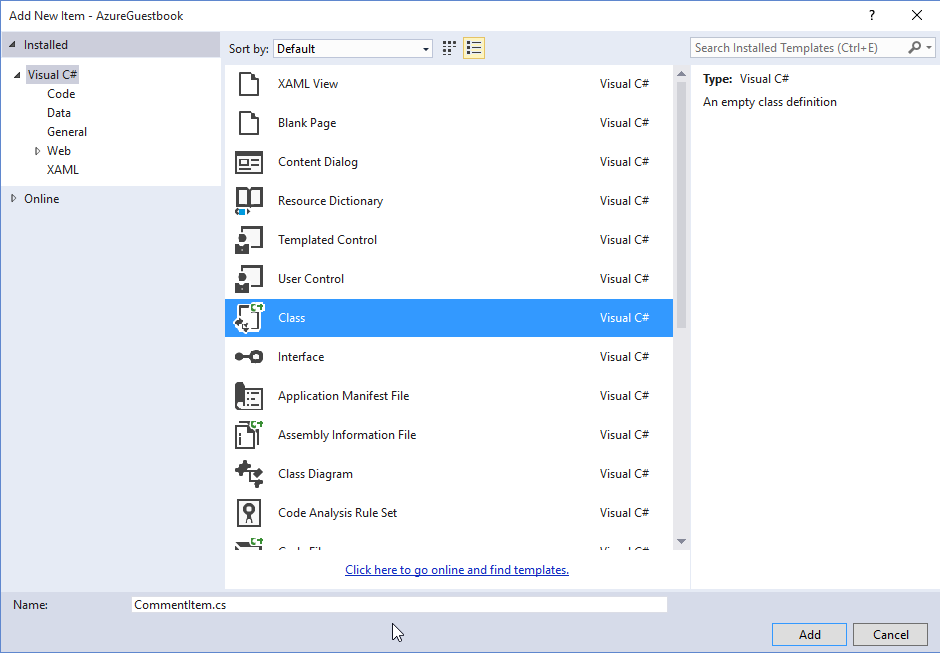
To insert the snippet, press Ctrl+K, then X, and then navigate to the **Azure Guestbook UI** snippet shown as follows:



The code should be as follows:

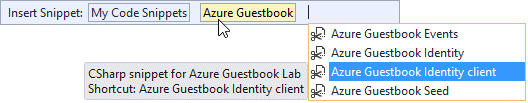
|  |
| --- |
| <!--LayoutRoot is the root grid where all page content is placed-->  <Grid x:Name="LayoutRoot" Background="#FF023752">  <Grid.RowDefinitions>  <RowDefinition Height="Auto"/>  <RowDefinition Height="\*"/>  <RowDefinition Height="Auto"/>  </Grid.RowDefinitions>  <!--TitlePanel contains the name of the application and page title-->  <Image HorizontalAlignment="Right" VerticalAlignment="Top" Source="Assets/CloudSlice.png" Grid.RowSpan="3"/>  <Image Source="Assets/AzureLogo.png" Grid.Row="2" Height="30" Width="150" HorizontalAlignment="Right" Margin="20,20,30,20" VerticalAlignment="Bottom"/>  <Image Source="Assets/HomeNote.png" Grid.Row="2" Height="110" Width="115" HorizontalAlignment="Left" VerticalAlignment="Bottom" Margin="0,18"/>  <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="0,18,0,18" >  <TextBlock Text="Azure Guestbook" Grid.ColumnSpan="2" Style="{StaticResource TitleTextBlockStyle}" Foreground="White" Margin="18,0"/>  </StackPanel>  <!--ContentPanel - place additional content here-->  <Grid x:Name="ContentPanel" Grid.Row="1" Margin="18,0,18,0">  <Grid.RowDefinitions>  <RowDefinition Height="Auto" />  <RowDefinition Height="Auto" />  <RowDefinition Height="Auto" />  <RowDefinition Height="Auto" />  <RowDefinition Height="Auto" />  <RowDefinition Height="\*" />  </Grid.RowDefinitions>  <Grid.ColumnDefinitions>  <ColumnDefinition Width="4\*" />  <ColumnDefinition Width="2\*" />  </Grid.ColumnDefinitions>  <TextBlock Grid.Row="0" Grid.ColumnSpan="2" Style="{StaticResource BodyTextBlockStyle}" Text="Comment" Foreground="White" Margin="0,10" TextWrapping="Wrap"/>  <TextBox Grid.Row="1" Grid.ColumnSpan="2" Name="CommentTextBox" TextWrapping="Wrap" BorderBrush="#FF7A7A7A" />  <Button Grid.Row ="2" Grid.Column="0" Name="SubmitButton" Click="SubmitButton\_Click" Foreground="White" Margin="0,10">Submit</Button>  <TextBlock Grid.Row="3" Grid.ColumnSpan="2" Style="{StaticResource BodyTextBlockStyle}" Text="Send us feedbacks..." Foreground="White" TextWrapping="Wrap"/>  <Button Grid.Row="4" Grid.ColumnSpan="2" Name="RefreshButton" Click="RefreshButton\_Click" HorizontalAlignment="Stretch" Foreground="White" VerticalAlignment="Top" Margin="0,10" >Refresh</Button>  <ListView Name="ltvGuestbook" Grid.Row="5" Grid.ColumnSpan="2">  <ListView.ItemTemplate>  <DataTemplate>  <StackPanel Orientation="Horizontal" Margin="2">  <TextBlock Name="textComment" Text="{Binding Description}" MinWidth="200" Margin="5"/>  <TextBlock Margin="5">Posted by :</TextBlock>  <TextBlock Name="textName" Text="{Binding Name}" MinWidth="100" Margin="5" />  </StackPanel>  </DataTemplate>  </ListView.ItemTemplate>  </ListView>  </Grid>  <StackPanel x:Name="FooterPanel" Grid.Row="2" Margin="0,18,0,18" >  <TextBlock Grid.ColumnSpan="2" VerticalAlignment="Bottom" Text="Fill the service guest book" Foreground="White" Margin="110,20,0,0" TextWrapping="Wrap"/>  </StackPanel>  </Grid> |

1. Add a new class **CommentItem**.



1. Replace the whole code behind using the snippet **Azure Guestbook Identity Client**.

To insert the snippet, press Ctrl+K, then X, and then navigate to the **Azure Guestbook Identity client** snippet shown as follows:



The code should be as follows:

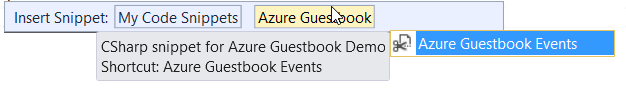
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Text;  using Newtonsoft.Json;  namespace AzureGuestbook  {  public class CommentItem  {  public string Id { get; set; }  [JsonProperty(PropertyName = "Description")]  public string Description { get; set; }  [JsonProperty(PropertyName = "Name")]  public string Name { get; set; }  }  } |

1. Open the **MainPage.xaml.cs** code and add the namespaces.

|  |
| --- |
| using Windows.System;  using Windows.UI.Popups;  using System.Threading.Tasks;  using Microsoft.WindowsAzure.MobileServices; |

1. Insert the code events handler after the **MainPage** constructor using the snippet **Azure Guestbook Events**.

To insert the snippet, press Ctrl+K, then X, and then navigate to the **Azure Guestbook Events** snippet shown as follows:

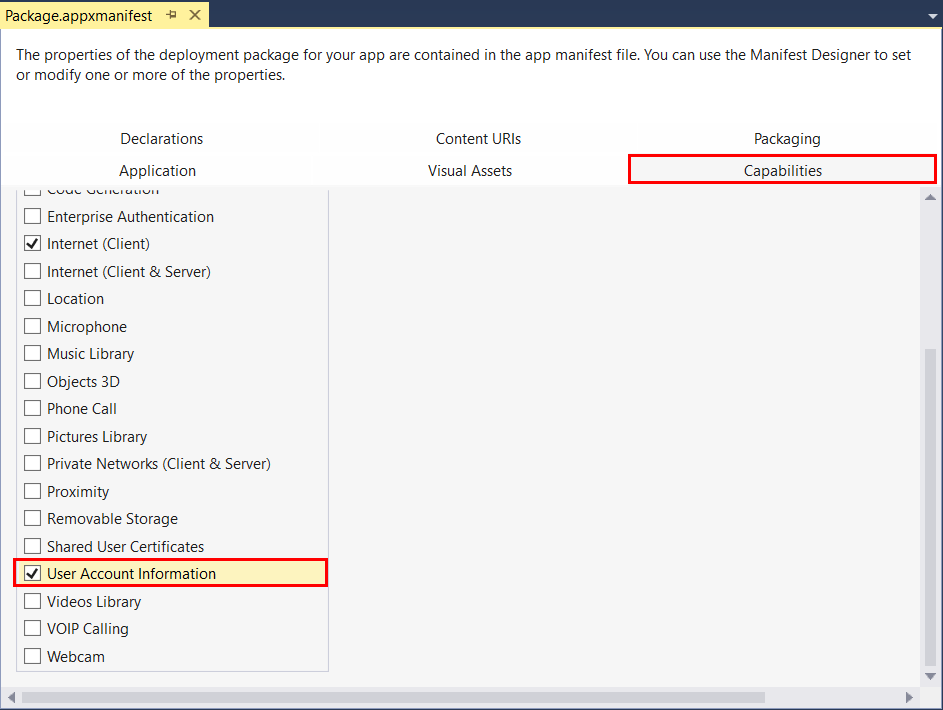


The code should be as follows:

|  |
| --- |
| private MobileServiceCollection<CommentItem, CommentItem> items;  private IMobileServiceTable<CommentItem> commentTable = App.MobileService.GetTable<CommentItem>();  //private IMobileServiceSyncTable<CommentItem> commentTable = App.MobileService.GetSyncTable<CommentItem>(); // offline sync  private async Task InsertCommentItem(CommentItem commentItem)  {  // This code inserts a new CommentItem into the database. When the operation completes  // and Mobile Services has assigned an Id, the item is added to the CollectionView  await commentTable.InsertAsync(commentItem);  items.Add(commentItem);  await RefreshCommentItems();  //await SyncAsync(); // offline sync  }  private async Task RefreshCommentItems()  {  MobileServiceInvalidOperationException exception = null;  try  {  IReadOnlyList<User> users = await User.FindAllAsync();  string displayName = "";  foreach (User u in users)  {  var pFirstName = await u.GetPropertyAsync(KnownUserProperties.FirstName);  var pLastName = await u.GetPropertyAsync(KnownUserProperties.LastName);  displayName = string.Format("{0} {1}", pFirstName, pLastName);  }  // This code refreshes the entries in the list view by querying the CommentItems table.  items = await commentTable  //.Where(commentItem => commentItem.Name == displayName)  .OrderByDescending(commentItem => commentItem.Id)  .ToCollectionAsync();  }  catch (MobileServiceInvalidOperationException e)  {  exception = e;  }  if (exception != null)  {  await new MessageDialog(exception.Message, "Error loading items").ShowAsync();  }  else  {  ltvGuestbook.ItemsSource = items;  this.SubmitButton.IsEnabled = true;  }  }  private async void RefreshButton\_Click(object sender, RoutedEventArgs e)  {  RefreshButton.IsEnabled = false;  //await SyncAsync(); // offline sync  await RefreshCommentItems();  RefreshButton.IsEnabled = true;  }  private async void SubmitButton\_Click(object sender, RoutedEventArgs e)  {  IReadOnlyList<User> users = await User.FindAllAsync();  string displayName = "";  foreach (User u in users)  {  var pFirstName = await u.GetPropertyAsync(KnownUserProperties.FirstName);  var pLastName = await u.GetPropertyAsync(KnownUserProperties.LastName);  displayName = string.Format("{0} {1}", pFirstName, pLastName);  }  var num = 0;  if (items.Count > 0)  {  var lastComment = items.OrderByDescending(item => item.Id).First();  num = Convert.ToInt32(lastComment.Id) + 1;  }  var commentItem = new CommentItem { Id = num.ToString("D3"), Description = CommentTextBox.Text, Name = displayName };  await InsertCommentItem(commentItem);  }  protected override async void OnNavigatedTo(NavigationEventArgs e)  {  //await InitLocalStoreAsync(); // offline sync  await RefreshCommentItems();  }  #region Offline sync  //private async Task InitLocalStoreAsync()  //{  // if (!App.MobileService.SyncContext.IsInitialized)  // {  // var store = new MobileServiceSQLiteStore("localstore.db");  // store.DefineTable<CommentItem>();  // await App.MobileService.SyncContext.InitializeAsync(store);  // }  //  // await SyncAsync();  //}  //private async Task SyncAsync()  //{  // await App.MobileService.SyncContext.PushAsync();  // await commentTable.PullAsync("commentItems", commentTable.CreateQuery());  //}  #endregion |

**Note** Notice that the methods are async.

1. Open the **Package.appxmanifest** file for the Windows App project and in the **Capabilities** tab, enable the **User Account Information** capabilities.



1. Build and run.

Exercise 2: Send push notification

#### Objectives

In this exercise, you will learn:

* How to send push notifications.

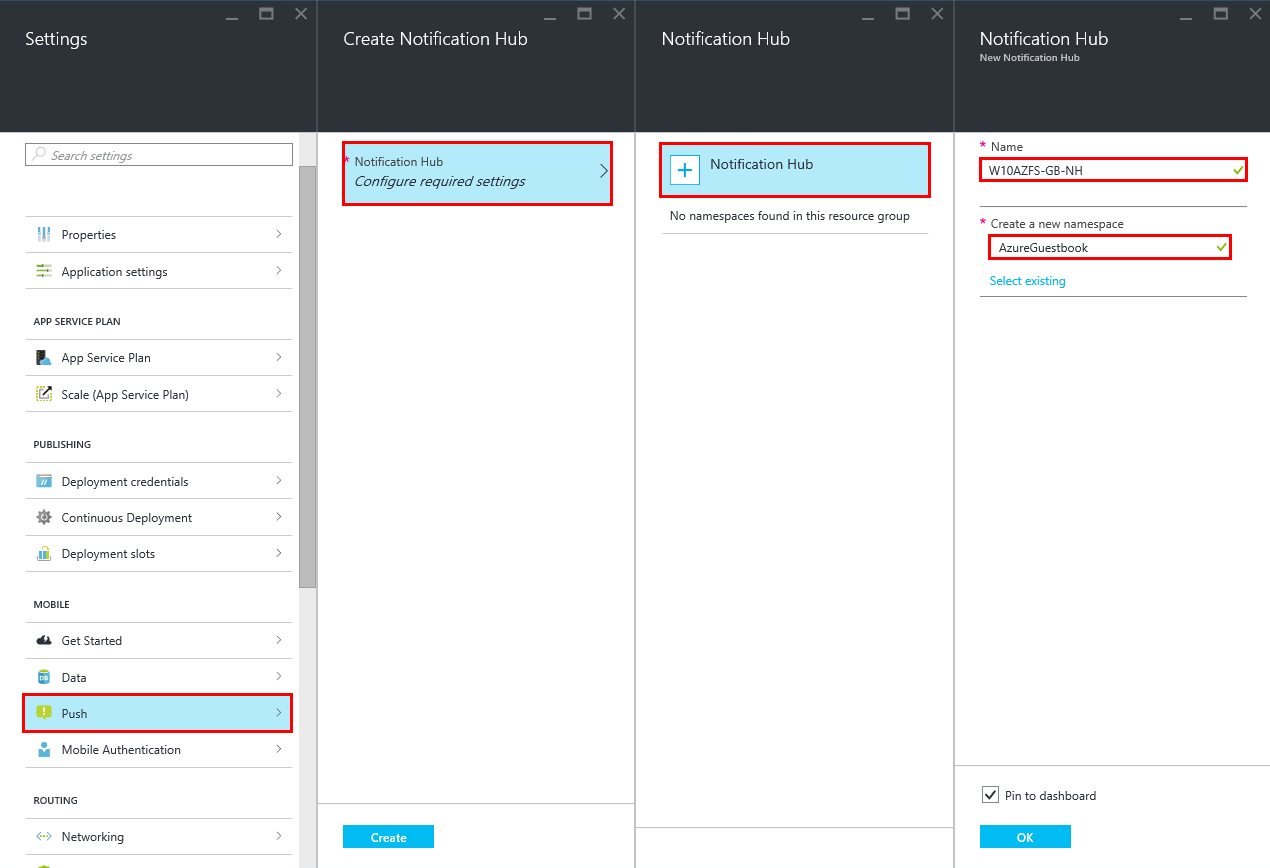
#### Scenario

In this scenario, you will learn how to send push notifications to a Windows Universal App by using Azure App Service Mobile Apps and Azure Notification Hubs. In this scenario, when a new item is added, your Mobile App backend sends a push notification to all Windows apps registered with the Windows Notification Service (WNS).

Task 1: Create a Notification Hub

1. Log onto the [Azure Portal](https://portal.azure.com/).
2. In the top-left side of the window, click **Browse**, then navigate to **Mobile Apps** > **W10AZFS-GB-MA**.
3. From the Mobile App blade, click **Settings** > **MOBILE** > **Push**.

Follow the workflow to create a notification hub. You will need to create a new namespace if there is none in your current resource group. Click **Create** after you have configured all the settings.

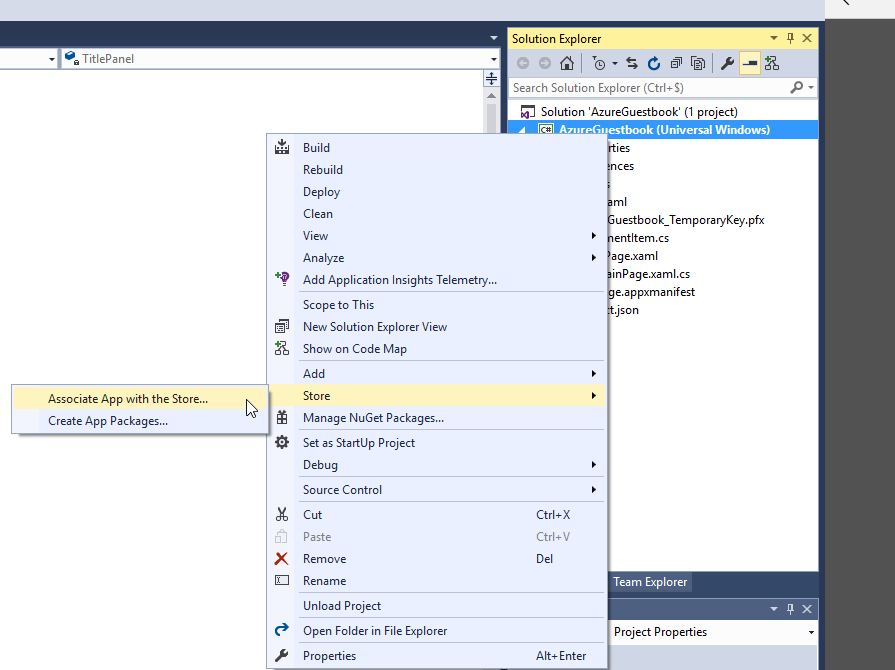


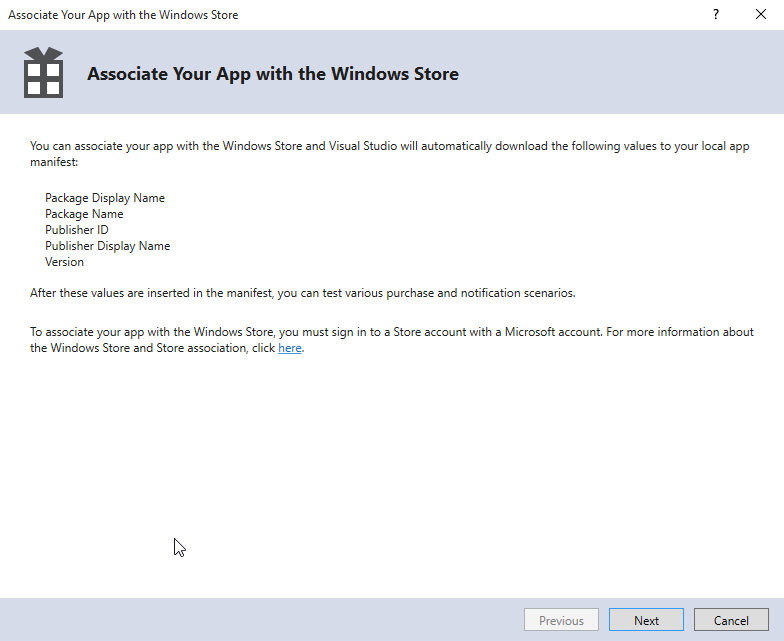
1. Click **OK** and then click **Create**.

Next, you will use this notification hub to enable push for your app.

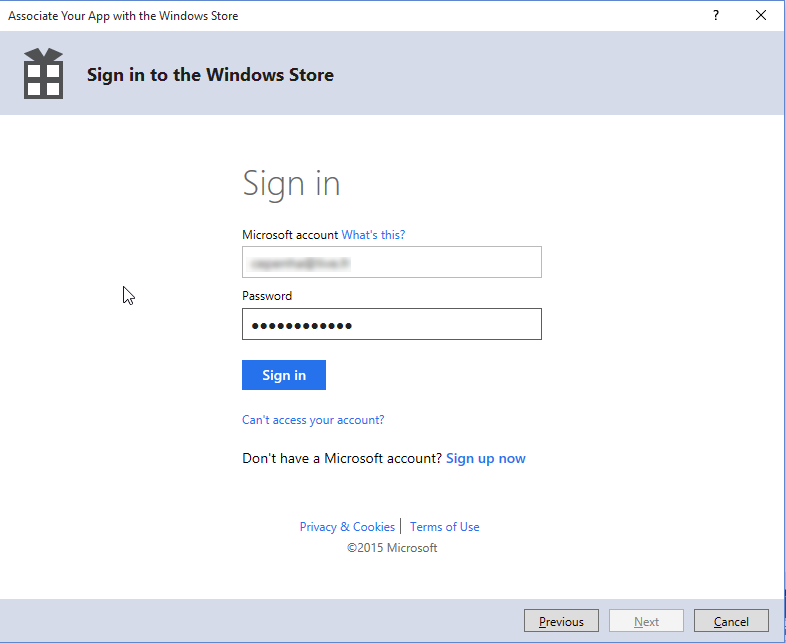
Task 2: Register Your App for Push Notifications

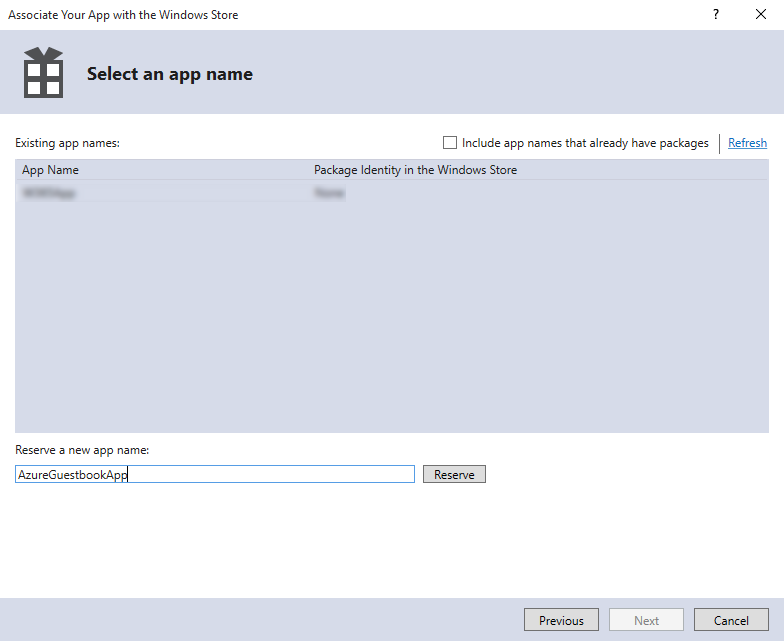
1. Open the **AzureGuestbook** (Windows Universal App) solution you created before.
2. On Visual Studio Solution Explorer, right-click the Windows Store app project, click **Store** > **Associate App with the Store**.



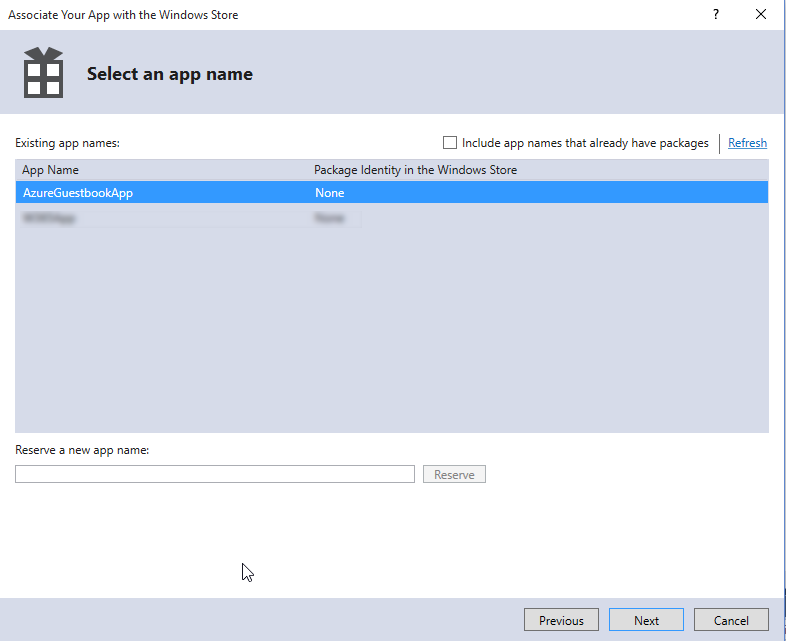


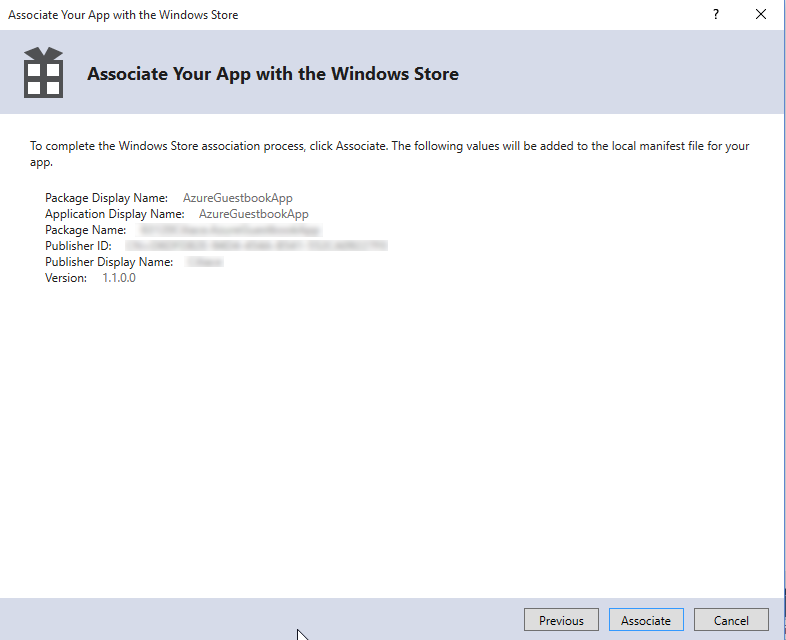
1. In the wizard, click **Next**. Sign in with your Microsoft account, type a name for your app in **Reserve a new app name**, and then click **Reserve**.



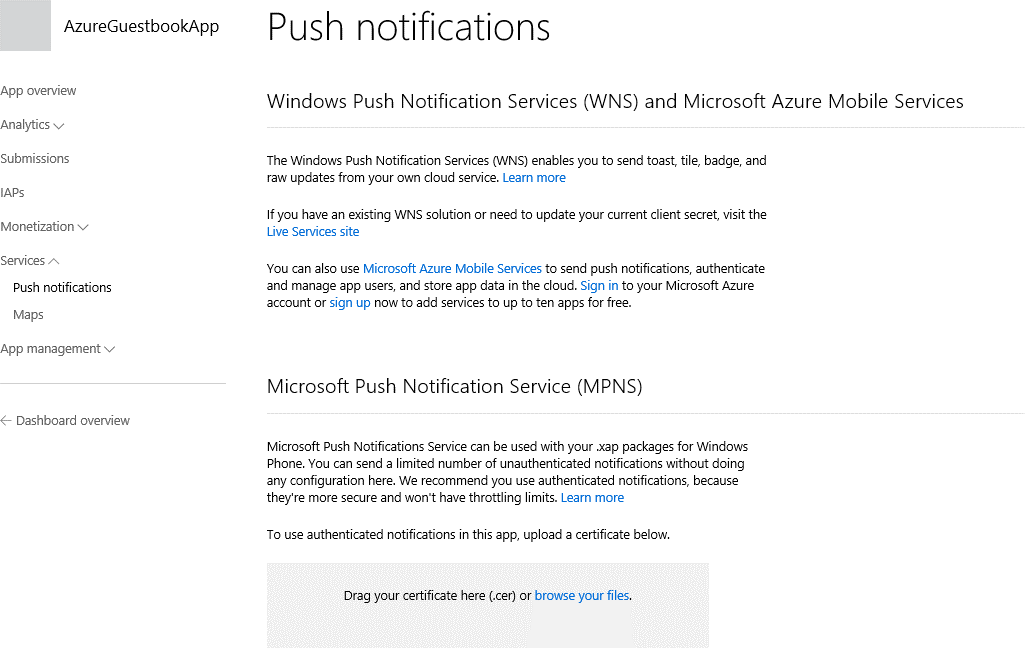


1. After the app registration is successfully created, select the new **App name**, click **Next**, and then click **Associate**. This adds the required Windows Store registration information to the application manifest.

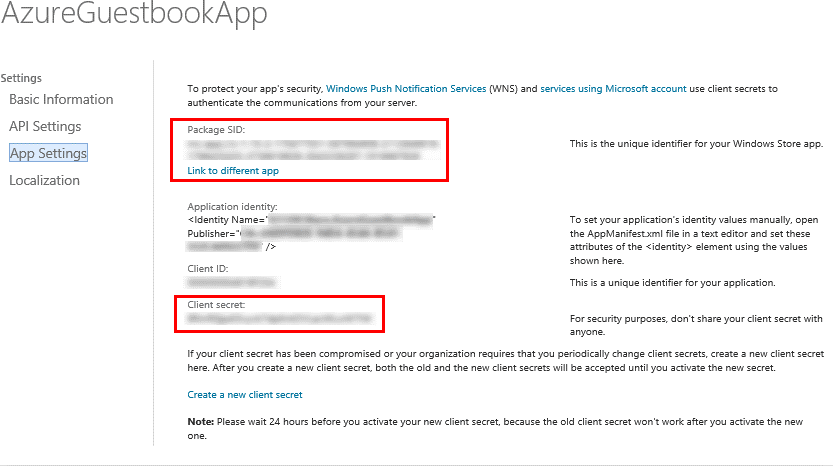




1. Navigate to the [Windows Dev Center](https://dev.windows.com/en-us/overview), sign in with your Microsoft account, click the new app registration in My apps, and then expand **Services** > **Push notifications**.
2. In the **Push notifications** page, click **Live Services** **site** under Microsoft Azure Mobile Services.



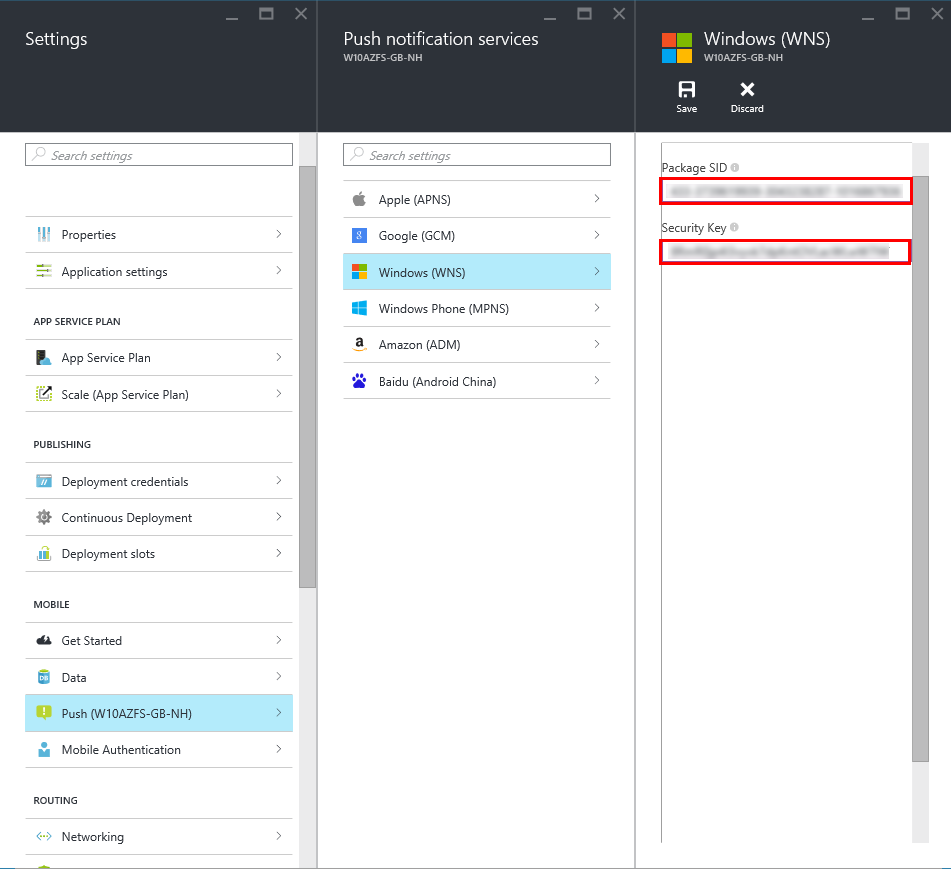
1. In the **App Settings** tab, note the values of **Client secret** and **Package SID**.



**Important** The client secret and package SID are important security credentials. Do not share these values with anyone or distribute them with your app.

Task 3: Configure Mobile App to Send Push Request

1. Log on to the Azure Portal, select **Browse** > **Mobile App** > **W10AZFS-GB-MA** > **Push notification** services.
2. In Windows Notification Service, enter the **Security Key** (client secret) and **Package SID** that you obtained from the Live Services site, and then click **Save**.



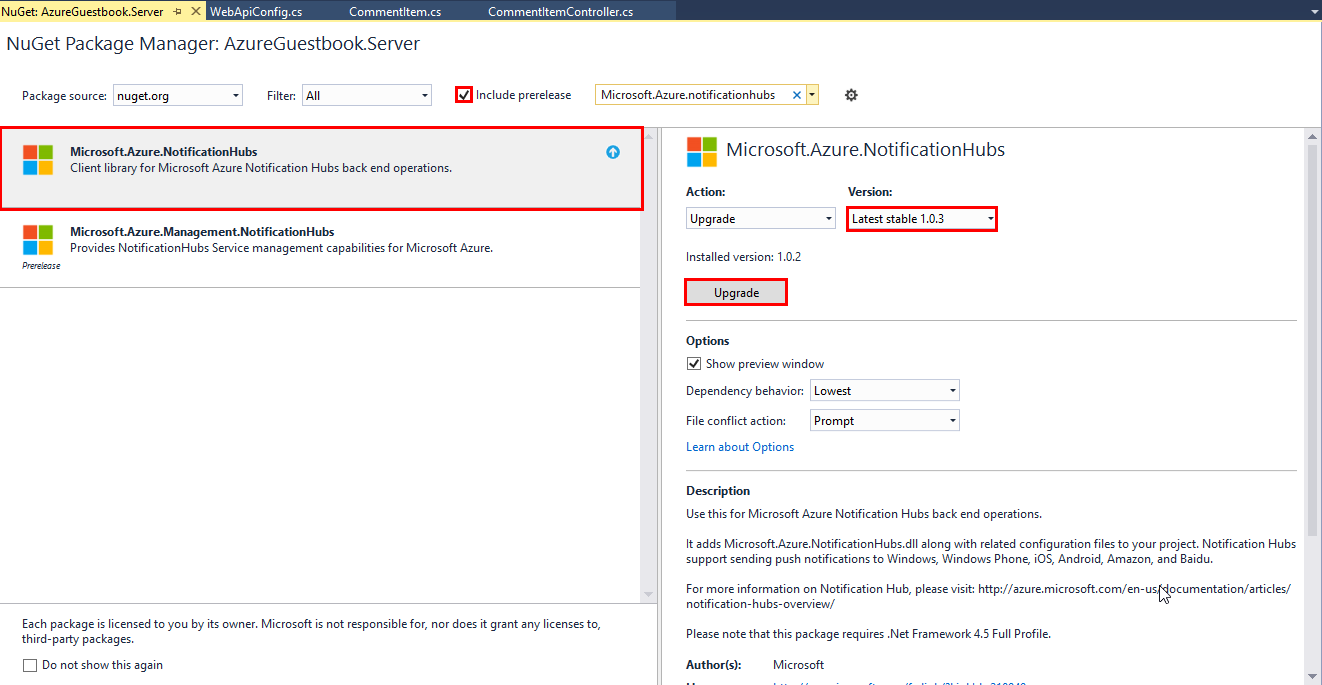
1. Click **Save**.

**Note** Your Mobile App backend is now configured to work with WNS.

Now that push notifications are enabled in the app, you must update your app back end to send push notifications.

Task 4: Update the Server to Send Push Notifications

1. Open the Mobile App Backend solution in Visual Studio.
2. Right-click the server project and click **Manage NuGet Packages**, search for **Microsoft.Azure.NotificationHubs**, and then click **Install** (if not installed). This installs the Notification Hubs client library.



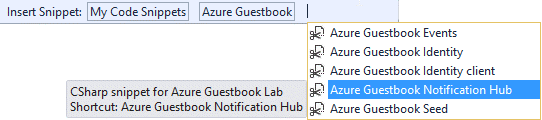
1. In the server project, open **Controllers** > **CommentItemController.cs**, and add the following using statements:

|  |
| --- |
| using System.Collections.Generic;  using Microsoft.Azure.NotificationHubs;  using Microsoft.Azure.Mobile.Server.Config; |

1. In the **PostCommentItem** method, add the following code after the call to **InsertAsync**:

Insert code after the call to **InsertAsync** of the **PostCommentItem** method using the snippet **Azure Guestbook Notification Hub**.

To insert the snippet, press Ctrl+K, then X, and then navigate to the **Azure Guestbook Notification Hub** snippet shown as follows:



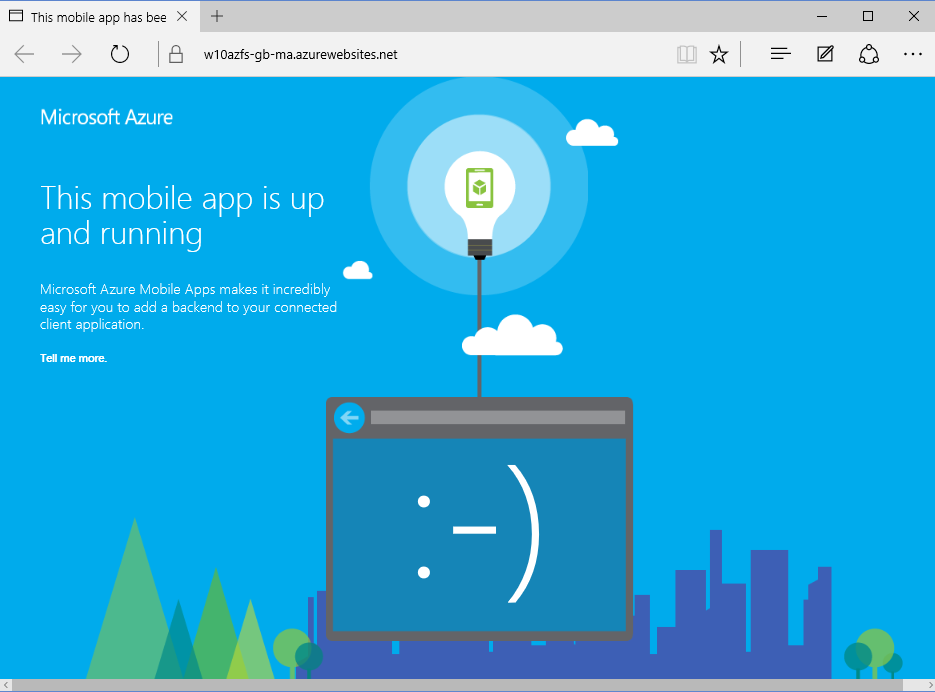
The code should be as follows:

|  |
| --- |
| // Get the settings for the server project.  HttpConfiguration config = this.Configuration;  MobileAppSettingsDictionary settings =  this.Configuration.GetMobileAppSettingsProvider().GetMobileAppSettings();  // Get the Notification Hubs credentials for the Mobile App.  string notificationHubName = settings.NotificationHubName;  string notificationHubConnection = settings  .Connections[MobileAppSettingsKeys.NotificationHubConnectionString].ConnectionString;  // Create a new Notification Hub client.  NotificationHubClient hub = NotificationHubClient  .CreateClientFromConnectionString(notificationHubConnection, notificationHubName);  // Define a WNS payload  var windowsToastPayload = @"<toast><visual><binding template=""ToastText01""><text id=""1"">" + item.Name + @" posted a new comment : """  + item.Description + @"""</text></binding></visual></toast>";  try  {  // Send the push notification and log the results.  var result = await hub.SendWindowsNativeNotificationAsync(windowsToastPayload);  // Write the success result to the logs.  config.Services.GetTraceWriter().Info(result.State.ToString());  }  catch (System.Exception ex)  {  // Write the failure result to the logs.  config.Services.GetTraceWriter()  .Error(ex.Message, null, "Push.SendAsync Error");  } |

This code tells the notification hub to send a push notification after a new item is inserted.

Task 5: Publish the Mobile Backend to Azure

1. In Visual Studio, right-click the project, click **Publish** > **Publish Web**, and then click **Publish**.
2. When your Mobile App backend has published successfully, you will see a landing page indicating success.



Task 6: Add Push Notifications to Your App

1. In Visual Studio, open the AzureGuestbook (Windows Universal App) solution.
2. Open the **App.xaml.cs** project file and add the following using statements:

|  |
| --- |
| using System.Threading.Tasks;  using Windows.Networking.PushNotifications; |

1. In the same file, add the following **InitNotificationsAsync** method definition and **PushNotification** class after the **App** class:

|  |
| --- |
| public class PushNotfication  {  public static async Task InitNotificationsAsync()  {  try  {  var channel = await PushNotificationChannelManager.CreatePushNotificationChannelForApplicationAsync();  await App.MobileService.GetPush().RegisterAsync(channel.Uri);  }  catch (Exception exception)  {  const int ERROR\_WINHTTP\_INVALID\_SERVER\_RESPONSE = unchecked((int)0x80072EE7);    if (exception.HResult == ERROR\_WINHTTP\_INVALID\_SERVER\_RESPONSE) {  // The mobile service client is not responding. You may be offine.  }  }  }  } |

**Note** This code retrieves the ChannelURI for the app from WNS, and then registers that ChannelURI with your App Service Mobile App.

Notice the exception handler to ignore exception on an offline scenario.

1. At the top of the **OnLaunched** event handler in **App.xaml.cs**, add the **async** modifier to the method definition and add the following call to the new **InitNotificationsAsync** method, shown as follows:

|  |
| --- |
| protected async override void OnLaunched(LaunchActivatedEventArgs e)  {  await PushNotfication.InitNotificationsAsync();  … |

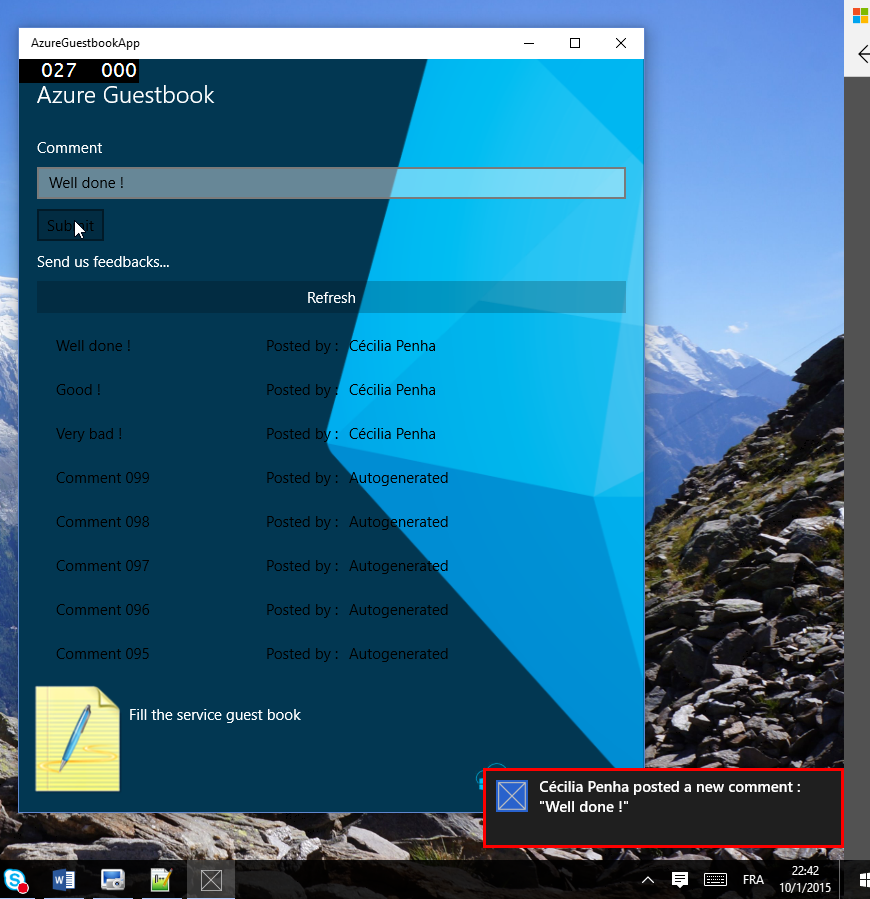
**Note** This ensures that the short-lived ChannelURI is registered each time the application is launched.

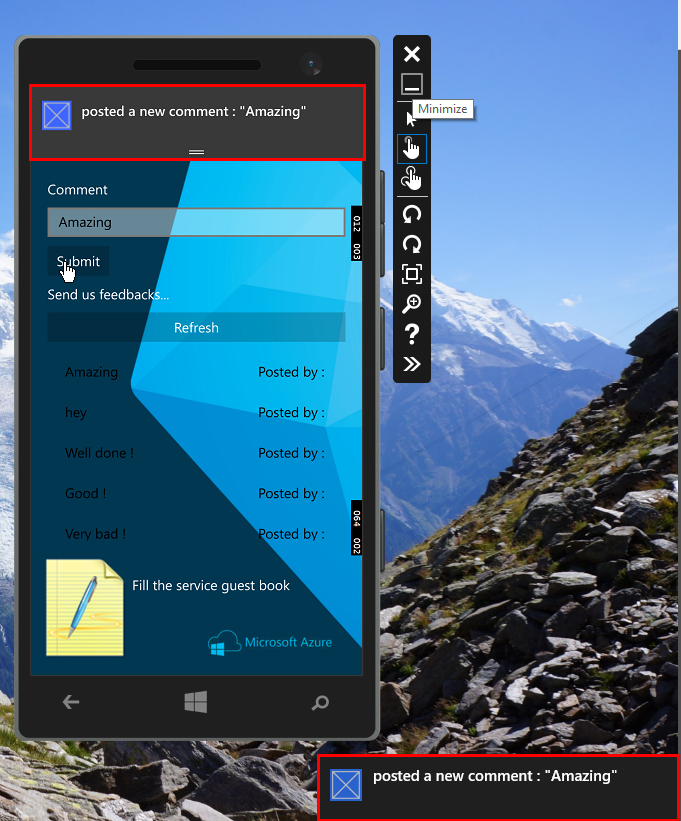
1. Build.

Task 7: Test Push Notifications in Your App

1. Press the F5 key to run the AzureGuestbook app.
2. After the app starts, the device is registered for push notifications.
3. Enter a comment and then click **Submit**.

Note that after the insert completes, the Windows Universal App receive a push notification from WNS. The notification is displayed on Windows even when the app is not running.





# Lab 2: Authentication

#### Introduction

In this lab, you will learn how to add authentication to your AzureGuestbook App.

#### Objectives

After completing this lab, you will be able to:

* Configure Authentication for your Mobile App.
* Add Authentication.

#### Estimated Time to Complete This Lab

30 minutes

#### Prerequisites

All resources allocate in Azure will follow these Naming Rules:

* Prefix: **W10AZFS-GB**: Windows 10 Azure Fast Start – Guestbook.
* You can use your own or ignore it as long as you stay consistent.

Before starting:

* Ensure you have an active Azure account. If you do not have an account, you can sign up for an Azure trial and get up to ten free mobile apps that you can keep using even after your trial ends. For details, see Azure Free Trial.
* Copy the AzureGuestbook. snippets to the Visual Studio code snippets folder: Documents\Visual Studio\Code Snippets
* Ensure you have a Visual Studio 2015 with the latest Azure SDK installed.

Exercise 1: Configure authentication for your Mobile App

#### Objectives

In this exercise, you will learn:

* How to authenticate users of an App Service Mobile App from your client application.

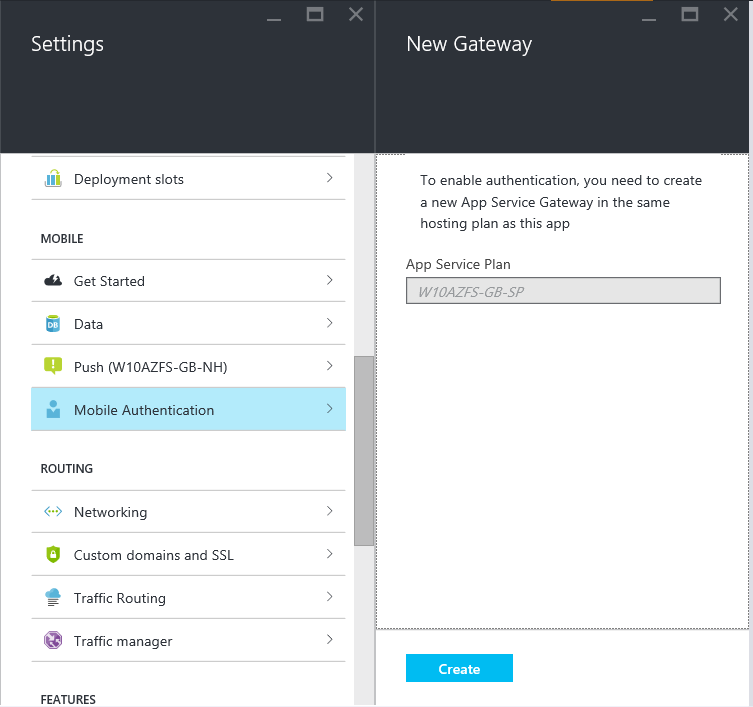
#### Scenario

In this scenario, we will learn how to authenticate users of an App Service Mobile App from your client application.

Task 1: Create an App Service Gateway

1. Log onto the [Azure Portal](https://portal.azure.com/).
2. Click **Browse** > **Mobile Apps** > **W10AZFS-GB-MA**, and then click **Settings** > **Mobile** > **Mobile authentication**.

You will be presented with a blade for creating your gateway. You do not need to enter any information, and the name of this gateway is automatically generated for you. Click **Create**.

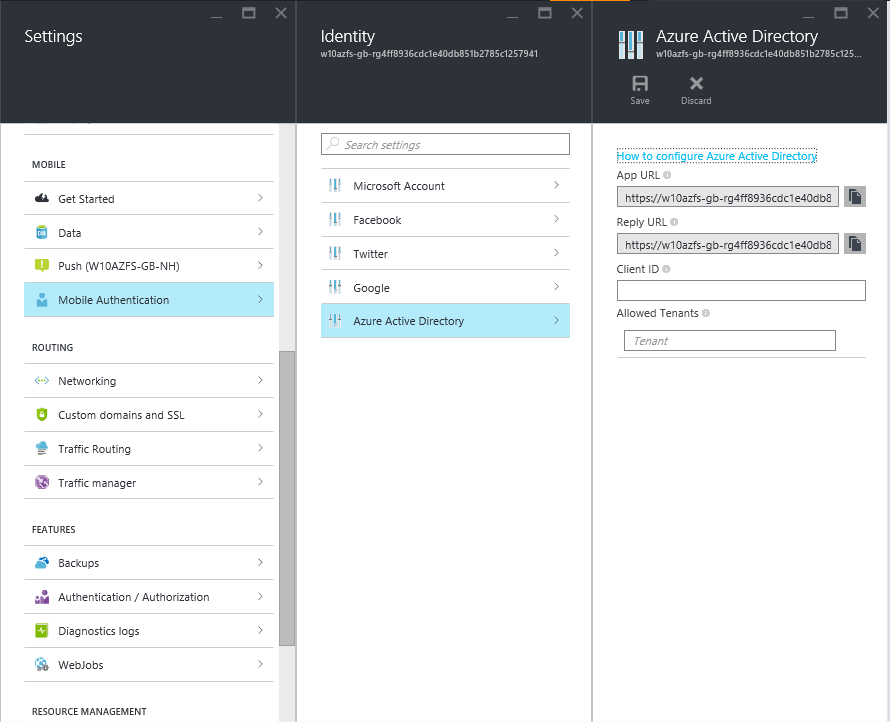


Next, you will use this gateway with an authentication provider.

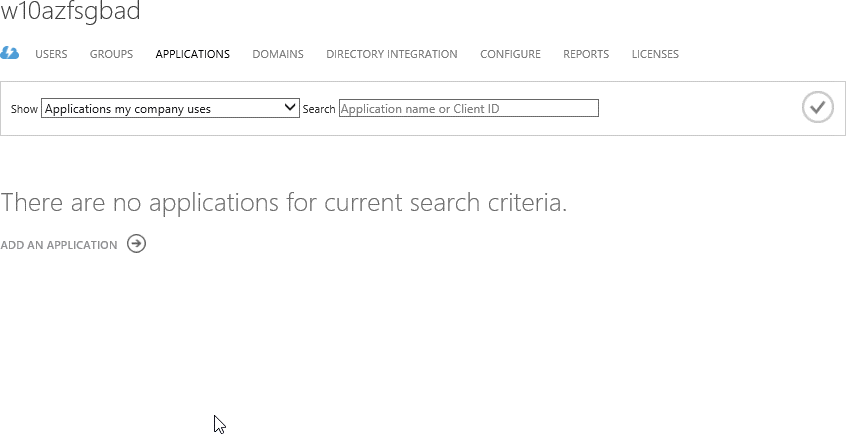
Task 2: Register Your App for Authentication to Azure AD Provider

You need to register your app at an identity provider's site, and then you will set the provider-generated credentials in the Mobile App backend.

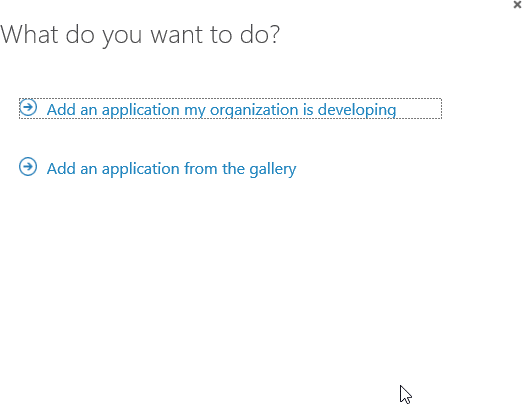
1. Register your app to **Azure AD** provider.
2. Click **Browse** > **Mobile Apps** > **W10AZFS-GB-MA**, and then click **Settings** > **MOBILE** > **Mobile authentication**.
3. Then, click **Azure Active Directory**. Copy the **App URL** and the **Reply URL** listed there. You will use these later. Ensure that the **App URL** and **Reply URL** are using the HTTPS scheme.



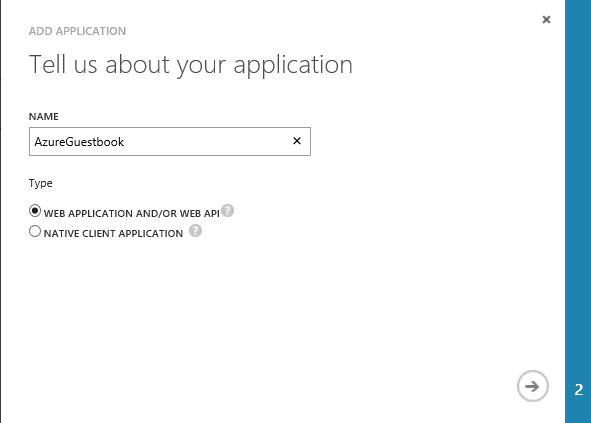
1. Navigate to your Azure AD, and then select the **Applications** tab at the top. Click **ADD** at the bottom to create a new app registration.



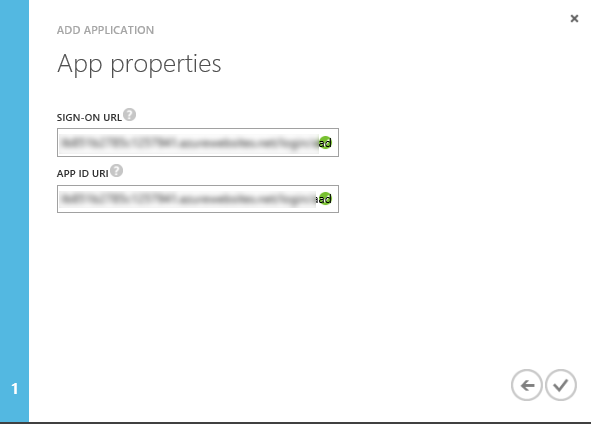
1. Click **Add an application my organization is developing**.



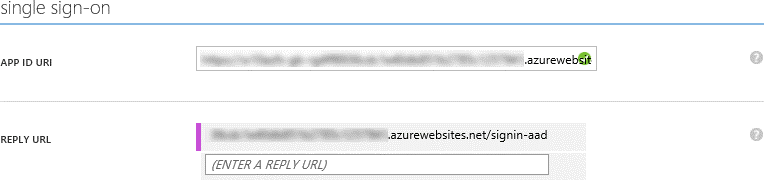
1. In the **ADD APPLICATION** wizard, enter a **Name** for your application and click the **WEB APPLICATION AND/OR WEB API** type. Then, click to continue.



1. In the **SIGN-ON URL** box, paste **the App ID** you copied from the Active Directory identity provider settings of your Mobile App. Enter that same resource identifier in the **APP ID URI** box. Then, click to continue.



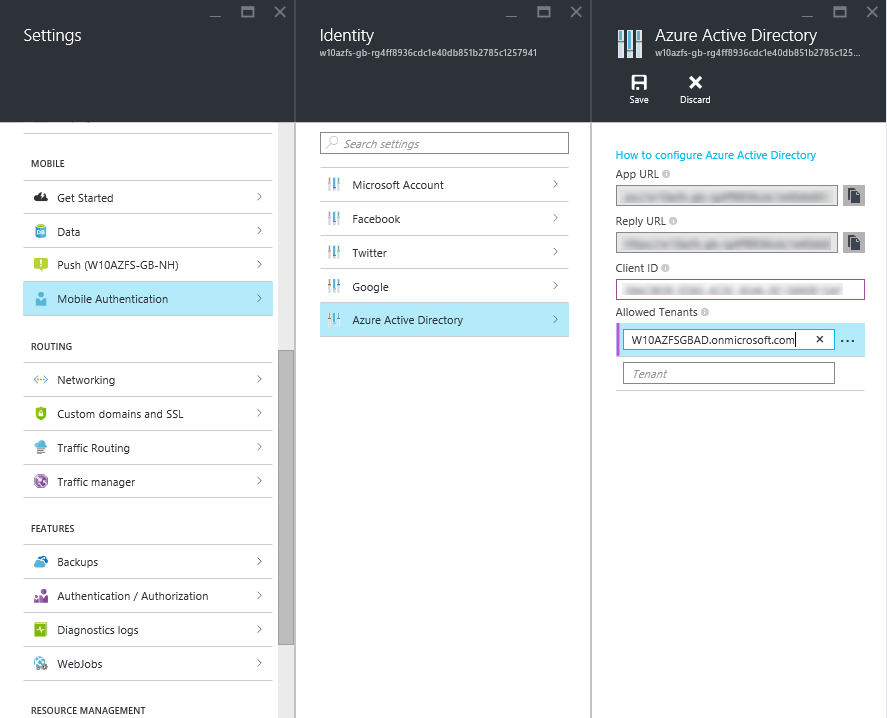
1. After the application has been added, click the **Configure** tab. Edit the **REPLY URL** under **single sign-on** to be **the Mobile App Reply URL** you copied earlier. It should be the Mobile App gateway appended with **/signin-aad**. Ensure that you are using the HTTPS scheme.



1. Click **Save**. Then, copy the **Client ID** for the App.

Task 3: Add Azure AD Information to Your Mobile App

1. Return to the Azure portal and the **Azure Active Directory** settings blade for your **Mobile App** **W10AZFS-GB-MA**, then click **Settings** > **MOBILE** > **Mobile authentication**. Paste the **Client ID** setting for the Azure Active Directory identity provider.
2. In the **Allowed Tenants** list, you need to add the domain of the directory in which you registered the application **W10AZFSGBAD.onmicrosoft.com**. You can find your default domain name by clicking the **Domains** tab on your **Azure Active Directory** tenant. Add your domain name to the **Allowed Tenants** list, then click **Save**.



You are now ready to use Azure Active Directory for authentication in your app. Next, you will set the provider-generated credentials in the Mobile App backend.

Exercise 2: Add authentication in your Mobile App

#### Objectives

In this exercise, you will learn:

* How to authenticate users of an App Service Mobile App from your client application.

#### Scenario

In this scenario, we will learn how to authenticate users of an App Service Mobile App from your client application.

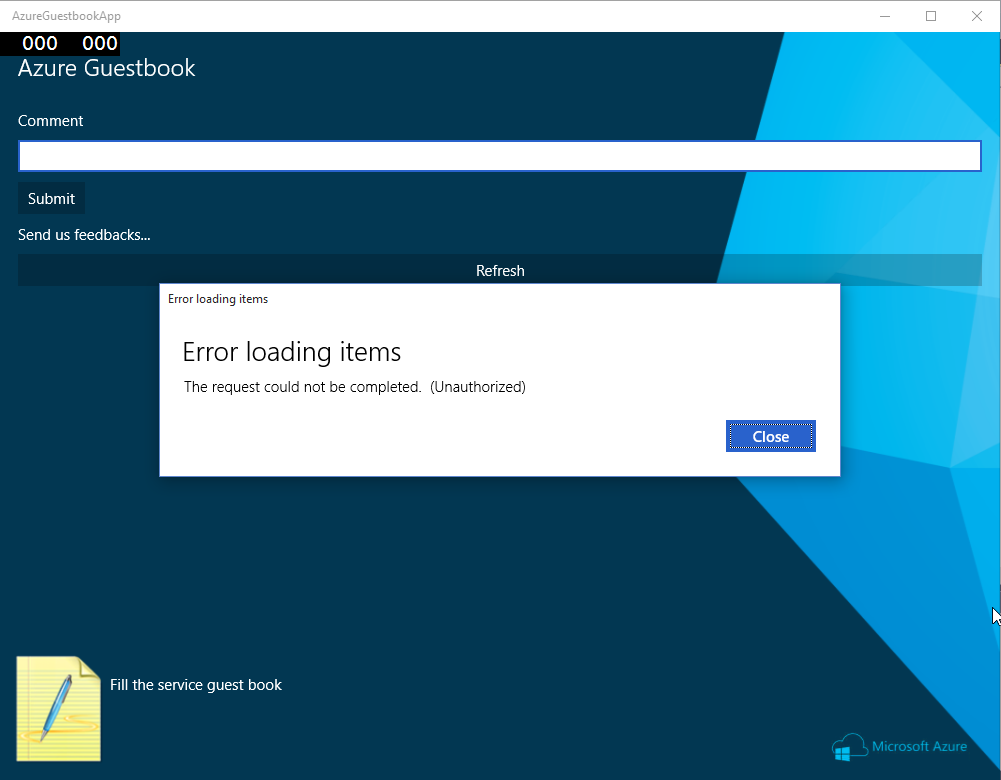
Task 1: Restrict Permissions to Authenticated Users

1. Open the Mobile App backend project in Visual Studio and navigate to **Controllers** > **CommentItemController.cs**.
2. Add the **[Authorize]** attribute to the **CommentItemController** class, as follows. This requires that all operations against the **CommentItem** table be made by an authenticated user.

|  |
| --- |
| [Authorize]  public class CommentItemController : TableController<CommentItem>  {  … |

To restrict access only to specific methods, you can also apply this attribute just to those methods instead of the class.

1. Republish your Mobile App project.
2. Open the **AzureGuestbook** app project in Visual Studio and open the **App.xaml.cs** project file. Ensure that the **MobileServiceClient** instance is configured to use both the URL of the Mobile App backend and the Gateway.
3. Press F5 to run the app; verify that an unhandled exception with a status code of 401 (Unauthorized) is raised after the app starts.



Next, you will add authentication to the app.

Task 2: Add Authentication to the App

1. Open the **AzureGuestbook** app project in Visual Studio. Open the **MainPage.cs** file project and add the following code snippet to the **MainPage** class:

|  |
| --- |
| // Define a member variable for storing the signed-in user.  private MobileServiceUser user;  // Define a method that performs the authentication process  // using a Azure AD sign-in.  private async System.Threading.Tasks.Task AuthenticateAsync()  {  while (user == null)  {  string message;  try  {  // Change 'MobileService' to the name of your MobileServiceClient instance.  // Sign-in using Azure AD authentication.  user = await App.MobileService  .LoginAsync(MobileServiceAuthenticationProvider.WindowsAzureActiveDirectory);  message =  string.Format("You are now signed in - {0}", user.UserId);  }  catch (InvalidOperationException)  {  message = "You must log in. Login Required";  }  var dialog = new MessageDialog(message);  dialog.Commands.Add(new UICommand("OK"));  await dialog.ShowAsync();  }  } |

This code authenticates the user with an Azure AD login. If you are using an identity provider other than Azure AD, change the value of **MobileServiceAuthenticationProvider** above to the value for your provider.

1. Comment out or delete the call to the **RefreshCommentItems** method in the existing **OnNavigatedTo** method override.

This prevents the data from being loaded before the user is authenticated. Next, you will add a **Sign in** button to the app that triggers authentication.

1. Add the following design code to the **MainPage.xaml** class:

|  |
| --- |
| <StackPanel x:Name="SignInPanel" Grid.Row="1" Margin="18,0,18,0" Visibility="Visible">  <Button Grid.Row="0" Grid.ColumnSpan="2" Name="LoginButton" Click="LoginButton\_Click" HorizontalAlignment="Stretch" Foreground="White" VerticalAlignment="Top" Margin="0,10" >Sign In</Button>  </StackPanel> |

1. Add the property **Visibility** to the Grid content panel.

|  |
| --- |
| <!--ContentPanel - place additional content here-->  <Grid x:Name="ContentPanel" Grid.Row="1" Margin="18,0,18,0" Visibility="Collapsed">  … |

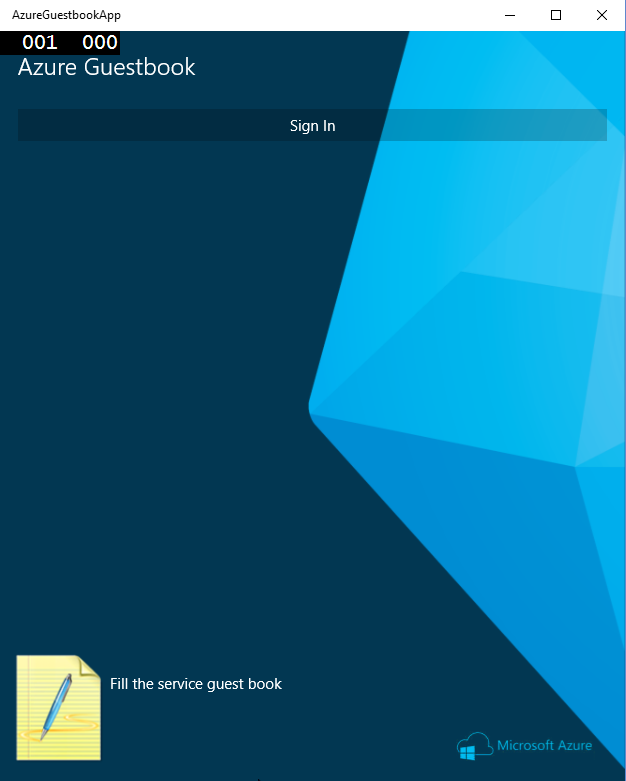
1. Add the following code snippet to the **MainPage** class:

|  |
| --- |
| private async void LoginButton\_Click(object sender, RoutedEventArgs e)  {  // Login the user and then load data from the mobile app.  await AuthenticateAsync();  // Hide the login button and load items from the mobile app.  this.SignInPanel.Visibility = Windows.UI.Xaml.Visibility.Collapsed;  this.ContentPanel.Visibility = Windows.UI.Xaml.Visibility.Visible;  await RefreshCommentItems();  } |

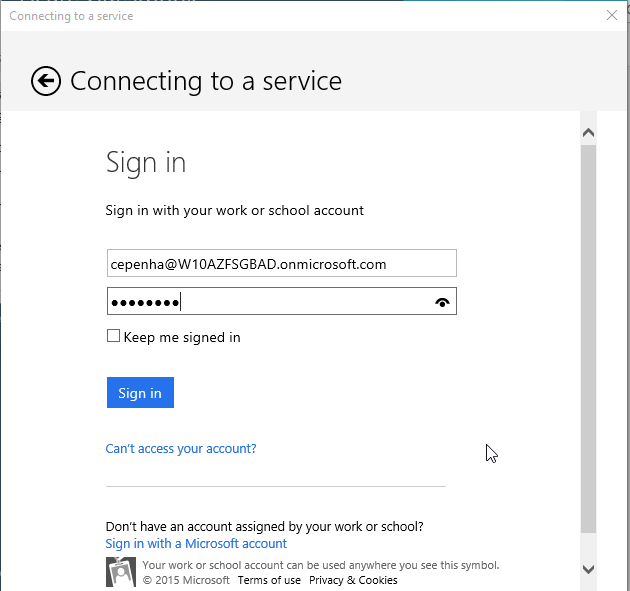
1. Add the following Gateway URI to the MobileService Call in the App class shown as follows:

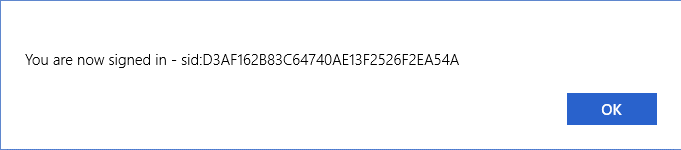
|  |
| --- |
| // This MobileServiceClient has been configured to communicate with the Azure Mobile Service and  // Azure Gateway using the application key. You're all set to start working with your Mobile Service!  public static MobileServiceClient MobileService = new MobileServiceClient(  "https://w10azfs-gb-ma.azurewebsites.net",  "https://w10azfs-gb-rg4ff8936cdc1e40db851b2785c1257941.azurewebsites.net",  ""  ); |

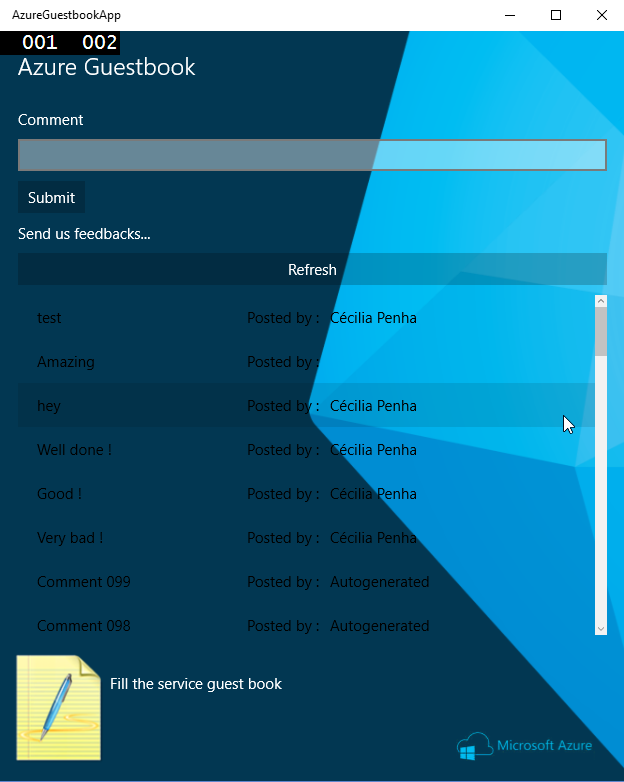
1. Press F5 to run the app. Click **Sign In**.



1. Enter the user credentials allowed to authenticate to the Mobile App (for example, Tenant W10AZFSGBAD.onmicrosoft.com).







Task 3: Store the Authentication Token on the Client

The previous example showed a standard sign-in that requires the client to contact both the identity provider and the App Service every time that the app starts. Not only is this method inefficient, you can face usage-relates issues if many customers try to start your app at the same time. A better approach is to cache the authorization token returned by your App Service and try to use this first before using a provider-based sign-in.

1. In the **MainPage.xaml.cs** project file, add the following using statements:

|  |
| --- |
| using System.Linq;  using Windows.Security.Credentials; |

1. Replace the **AuthenticateAsync** method with the following code:

|  |
| --- |
| private async System.Threading.Tasks.Task AuthenticateAsync()  {  string message;  // This sample uses the Azure Active Directory provider.  var provider = "WindowsAzureActiveDirectory";  // Use the PasswordVault to securely store and access credentials.  PasswordVault vault = new PasswordVault();  PasswordCredential credential = null;  while (credential == null)  {  try  {  // Try to get an existing credential from the vault.  credential = vault.FindAllByResource(provider).FirstOrDefault();  }  catch (Exception)  {  // When there is no matching resource an error occurs, which we ignore.  }  if (credential != null)  {  // Create a user from the stored credentials.  user = new MobileServiceUser(credential.UserName);  credential.RetrievePassword();  user.MobileServiceAuthenticationToken = credential.Password;  // Set the user from the stored credentials.  App.MobileService.CurrentUser = user;  try  {  // Try to return an item now to determine if the cached credential has expired.  await App.MobileService.GetTable<CommentItem>().Take(1).ToListAsync();  }  catch (MobileServiceInvalidOperationException ex)  {  if (ex.Response.StatusCode == System.Net.HttpStatusCode.Unauthorized)  {  // Remove the credential with the expired token.  vault.Remove(credential);  credential = null;  continue;  }  }  }  else  {  try  {  // Login with the identity provider.  user = await App.MobileService  .LoginAsync(provider);  // Create and store the user credentials.  credential = new PasswordCredential(provider,  user.UserId, user.MobileServiceAuthenticationToken);  vault.Add(credential);  }  catch (MobileServiceInvalidOperationException ex)  {  message = "You must log in. Login Required";  }  }  message = string.Format("You are now logged in - {0}", user.UserId);  var dialog = new MessageDialog(message);  dialog.Commands.Add(new UICommand("OK"));  await dialog.ShowAsync();  }  } |

1. Restart the app twice.

Notice that on the first start, sign-in with the provider is again required. However, on the second restart, the cached credentials are used and sign-in is bypassed.

**Note** This app tests for expired tokens during logon but token expiration can occur after authentication when the app is in use. For a solution to handling authorization errors related to expiring tokens, see the post: [Caching and handling expired tokens in Azure Mobile Services managed SDK](http://blogs.msdn.com/b/carlosfigueira/archive/2014/03/13/caching-and-handling-expired-tokens-in-azure-mobile-services-managed-sdk.aspx)

# Lab 3: Offline Data Sync in Mobile App

#### Introduction

In this lab, you will learn how to add offline support to a Windows Universal app using an Azure Mobile App backend. Offline sync allows user to interact with a mobile app—viewing, adding, or modifying data—even when there is no network connection. Changes are stored in a local database; once the device is back online, these changes are synced with the remote backend.

#### Objectives

After completing this lab, you will be able to:

* Update the client app to support offline features.
* Add Authentication.

#### Estimated Time to Complete This Lab

30 minutes

#### Prerequisites

All resources allocate in Azure will follow these Naming Rules:

* Prefix: **W10AZFS-GB**: Windows 10 Azure Fast Start – Guestbook.
* You can use your own or ignore it as long as you stay consistent.

Before starting:

* Ensure you have an active Azure account. If you do not have an account, you can sign up for an Azure trial and get up to ten free mobile apps that you can keep using even after your trial ends. For details, see Azure Free Trial.
* Copy the AzureGuestbook snippets to the Visual Studio code snippets folder: Documents\Visual Studio\Code Snippets
* Ensure you have a Visual Studio 2015 with the latest Azure SDK installed.

Exercise 1: Add offline sync data

#### Objectives

In this exercise, you will learn:

* How to authenticate users of an App Service Mobile App from your client application.
* How to add offline support to a Windows Universal App using an Azure Mobile App backend.

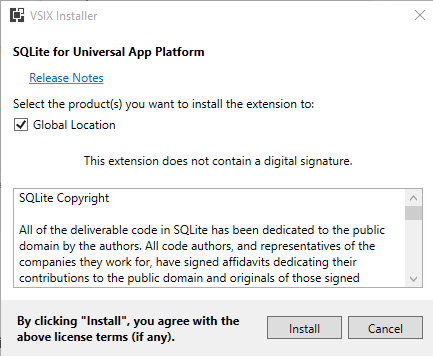
#### Scenario

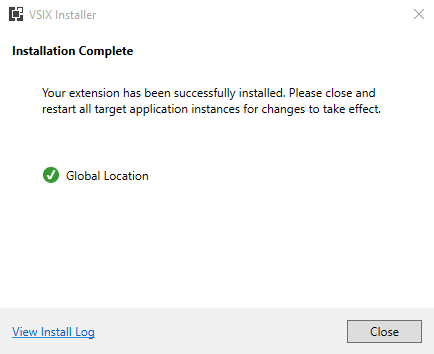
In this scenario, we will learn how to add offline support to a Windows Universal App using an Azure Mobile App backend.

Task 1: Update the Client App to Support Offline Features

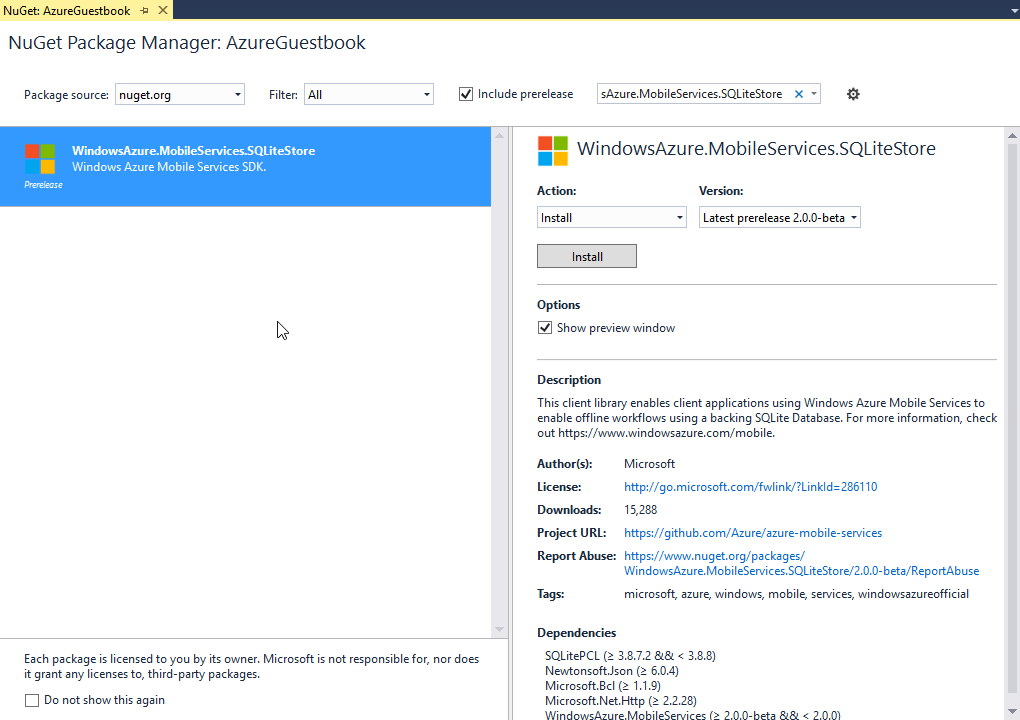
Azure Mobile App offline features allow you to interact with a local database when you are in an offline scenario. To use these features in your app, you initialize a **MobileServiceClient.SyncContext** to a local store. Then, reference your table through the **IMobileServiceSyncTable** interface. In this tutorial, we use SQLite for the local store.

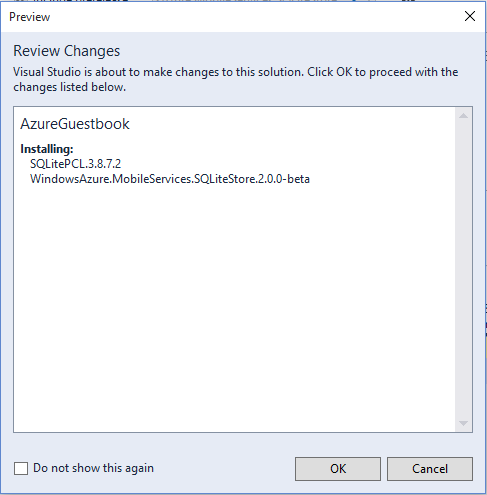
1. Install the package extension **SQLite for Universal App Platform** (sqlite-uap-3081101.vsix) available in the **Asset** folder of the lab.

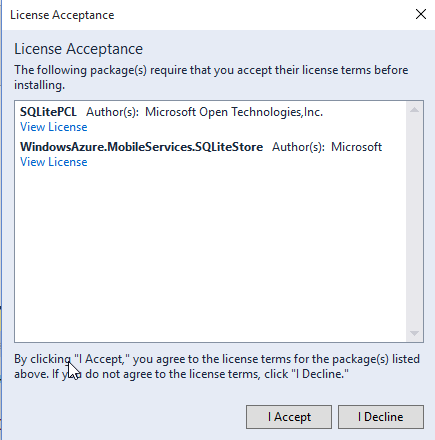




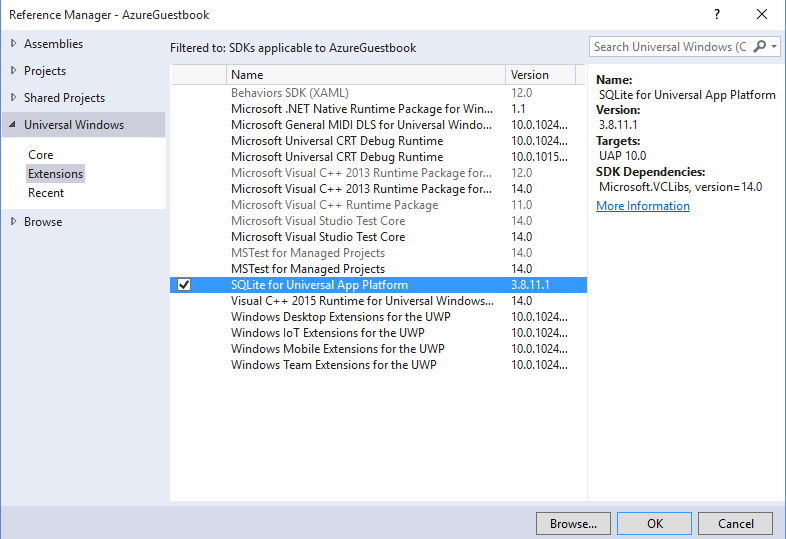
1. Restart Visual Studio.
2. Open the project **W10AZFS\_GB\_MAService** project from the end of Lab 1. Republish the Mobile App Backend.
3. Open the project **AzureGuestbook** App from the end of Lab 1. Install the **WindowsAzure.MobileServices.SQLiteStore** NuGet package.

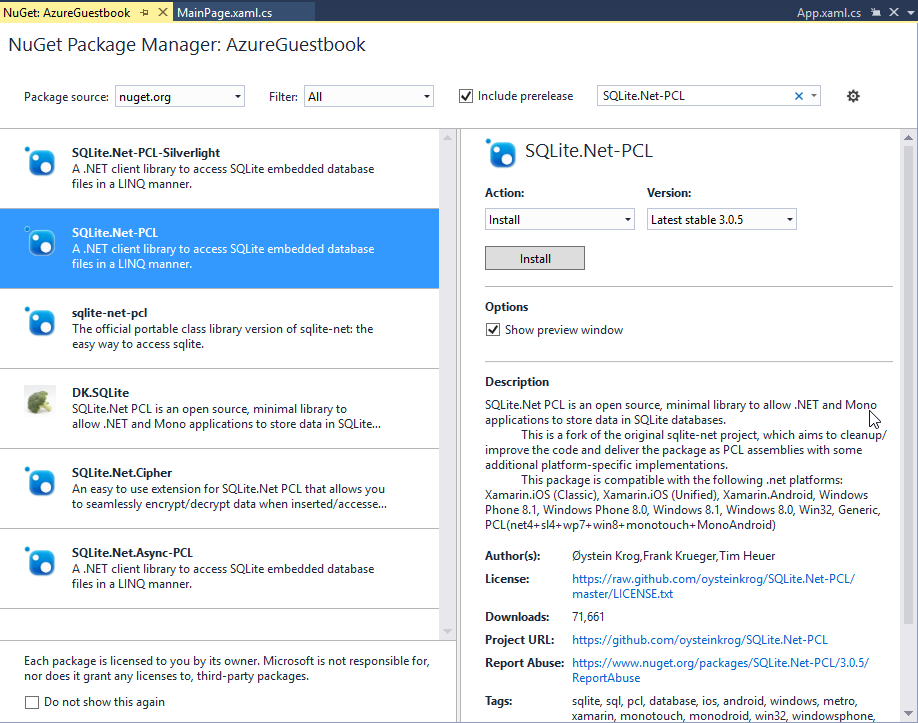


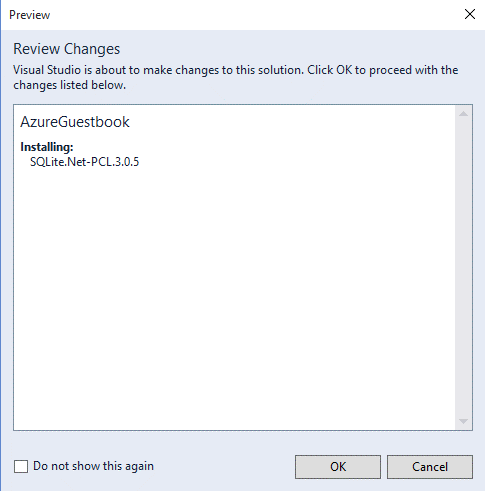




1. Add the reference **SQLite for Universal App Platform** to the project under **Universal Windows** > **Extensions**.







1. In **Solution Explorer**, open the **MainPage.xaml.cs** file. Add the following using statements at the top of the file:

|  |
| --- |
| using Microsoft.WindowsAzure.MobileServices.SQLiteStore; // offline sync  using Microsoft.WindowsAzure.MobileServices.Sync; // offline sync |

1. In **MainPage.xaml.cs**, comment the line of code that initializes **commentTable** as an **IMobileServiceTable**. **Uncomment** the line of code that initializes **commentTable** as an **IMobileServiceSyncTable**.

|  |
| --- |
| //private IMobileServiceTable<CommentItem> commentTable = App.MobileService.GetTable<CommentItem>();  private IMobileServiceSyncTable<CommentItem> commentTable = App.MobileService.GetSyncTable<CommentItem>(); // offline sync |

1. In **MainPage.cs**, in the region marked **Offline** **sync**, uncomment the methods **InitLocalStoreAsync** and **SyncAsync**. The method **InitLocalStoreAsync** initializes the client sync context with a SQLite store. In Visual Studio, you can select all commented lines and use the Ctrl+K+U keyboard shortcut to uncomment.

|  |
| --- |
| #region Offline sync  private async Task InitLocalStoreAsync()  {  if (!App.MobileService.SyncContext.IsInitialized)  {  var store = new MobileServiceSQLiteStore("localstore.db");  store.DefineTable<CommentItem>();  await App.MobileService.SyncContext.InitializeAsync(store);  }  await SyncAsync();  }  private async Task SyncAsync()  {  await App.MobileService.SyncContext.PushAsync();  await commentTable.PullAsync("commentItems", commentTable.CreateQuery());  }  #endregion |

1. In the **OnNavigatedTo** event handler, uncomment the call to **InitLocalStoreAsync**.

|  |
| --- |
| protected override async void OnNavigatedTo(NavigationEventArgs e)  {  await InitLocalStoreAsync(); // offline sync  await RefreshCommentItems();  } |

1. In the **InsertCommentItem** and **RefreshButton\_Click** event handlers, uncomment the line.

|  |
| --- |
| await SyncAsync(); // offline sync |

1. Add exception handlers in the **SyncAsync** method. In an offline situation, a MobileServicePushFailedException will be thrown with PushResult.Status == CancelledByNetworkError

|  |
| --- |
| private async Task SyncAsync()  {  String errorString = null;  try  {  await App.MobileService.SyncContext.PushAsync();  await commentTable.PullAsync("commentItems", commentTable.CreateQuery()); // first param is query ID, used for incremental sync  }  catch (MobileServicePushFailedException ex)  {  errorString = "Push failed because of sync errors. You may be offine.\nMessage: " +  ex.Message + "\nPushResult.Status: " + ex.PushResult.Status.ToString();  }  catch (Exception ex)  {  errorString = "Pull failed: " + ex.Message +  "\n\nIf you are still in an offline scenario, " +  "you can try your Pull again when connected with your Mobile Service.";  }  if (errorString != null)  {  MessageDialog d = new MessageDialog(errorString);  await d.ShowAsync();  }  } |

In this PullAsync example, we retrieve all records in the remote commentTable but it is also possible to filter records by passing a query. The first parameter to PullAsync is a query ID that is used for incremental sync, which uses the **Updated At** timestamp to get only records modified since the last sync. The query ID should be a descriptive string that is unique for each logical query in your client application. To opt-out of incremental sync, pass null as the query ID. This will retrieve all records on each pull operation, which is potentially inefficient.

1. Press F5 to rebuild and run the Client App.

The app will perform the same as it did before the offline sync changes, because it does a sync operation on the insert, and refresh operations. However, it will populate a local database which can be used in an offline scenario. We will cause and offline scenario in the next section now that the local database is populated.

Task 2: Update the Sync Behavior

In the steps, you will modify the app so that it does not sync on the insert operation, but only when the **Refresh** button is pressed. Then, you will break the app connection with the mobile service to simulate an offline scenario. When you add data items, they will be held in the local store, but not synced to the mobile service.

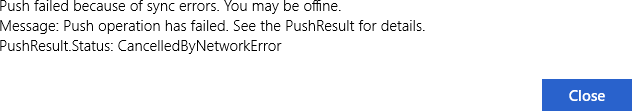
1. Open **MainPage.cs** in the AzureGuestbook project. Edit the method **InsertCommentItem** to comment out the calls to **SyncAsync** given as follows:

|  |
| --- |
| //await SyncAsync(); // offline sync |

1. Edit **App.xaml.cs** in the AzureGuestbook project. Comment out the initialization of the MobileServiceClient and add the following lines that use an invalid mobile service URL.

|  |
| --- |
| public static MobileServiceClient MobileService = new MobileServiceClient(  "https://your-mobile-service.azure-mobile.xxx/",  "AppKey"  ); |

1. In **InitLocalStoreAsync**(), comment out the call to **SyncAsync**(), so that the app does not perform a sync on launch.
2. Press F5 to build and run the app. Enter some new comment items and click **Submit** for each one. The new comment items exist only in the local store until they can be pushed to the mobile service. The client app performs as if its connected to the mobile service supporting all create, read, update, and delete (CRUD) operations.
3. Close the app and restart it to verify that the new items you created are persisted to the local store.
4. Click **Refresh**. The refresh will sync and handled the exception as follows:



1. Edit **App.xaml.cs** in the AzureGuestbook project. Change back the mobile service URL as follows:

|  |
| --- |
| public static MobileServiceClient MobileService = new MobileServiceClient(  "https://w10azfs-gb-ma.azurewebsites.net",  "",  ""  ); |

1. Press F5 and click **Refresh**. The refresh will sync successfully and you will be notified for each new comment synchronized.