**Creating a Cloud Native App using Azure Mobile Services**

Step 1: Create an Azure Mobile Service

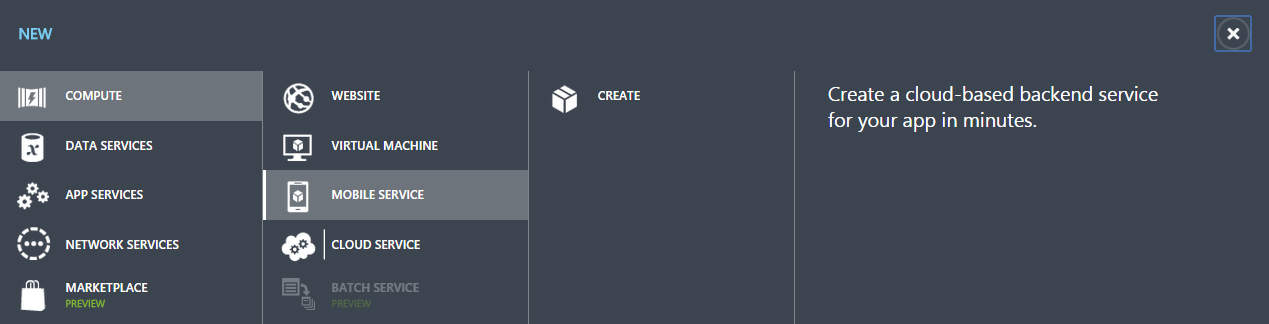
Step 2: Integrate your native app with the created mobile services: Start using cloud storage for your app in Azure Mobile Service

Step 3: Add notification to your app

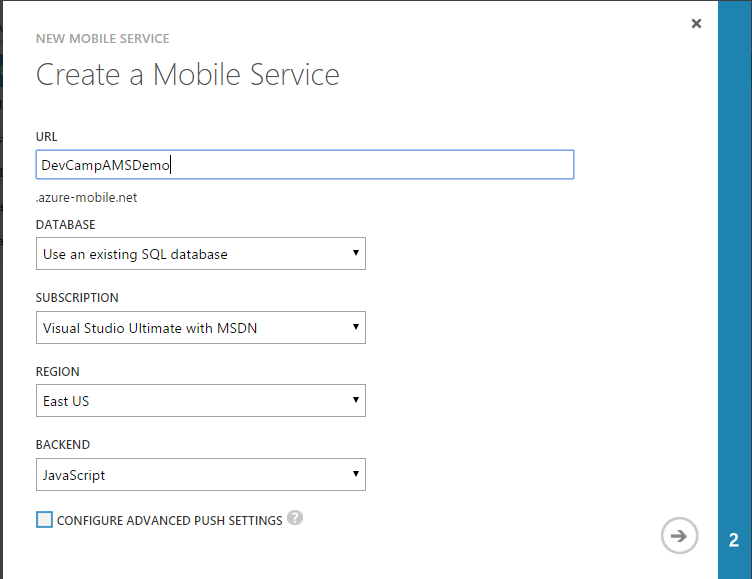
Step 4: Add authentication to your app

Step 1: Create an Azure Mobile Service

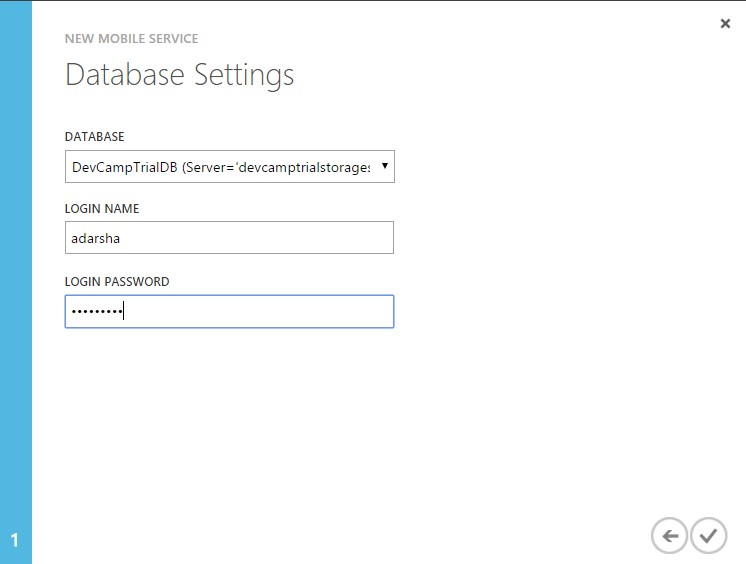
1. Login to manage.windowsazure.com Click New-> Compute ->Mobile Service -> Create



1. Enter details as required:

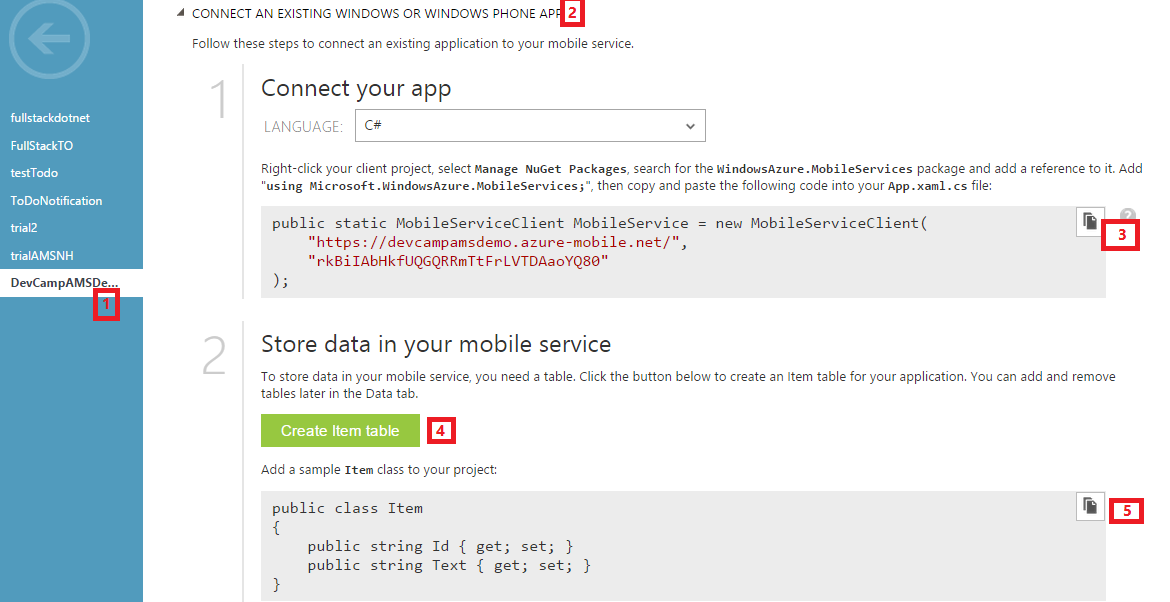


1. Enter the details of your database:



Step 2: Integrate your app with the created mobile services: Start with using cloud storage for your app in Azure Mobile Service

1. Download the sample Todo Windows Universal App which uses local storage from [here](https://code.msdn.microsoft.com/Get-Started-with-Data-in-0e863e57).
2. Open the project solution in Visual Studio. Build and run the app and you will notice that the ToDo list uses local memory storage. Now we will modify this to use the Mobile Services created in Step 1.
3. Go to the Azure portal and the newly created mobile service. Click on Connect to existing app and select the connection string that we will need to connect our app to the Azure backend.



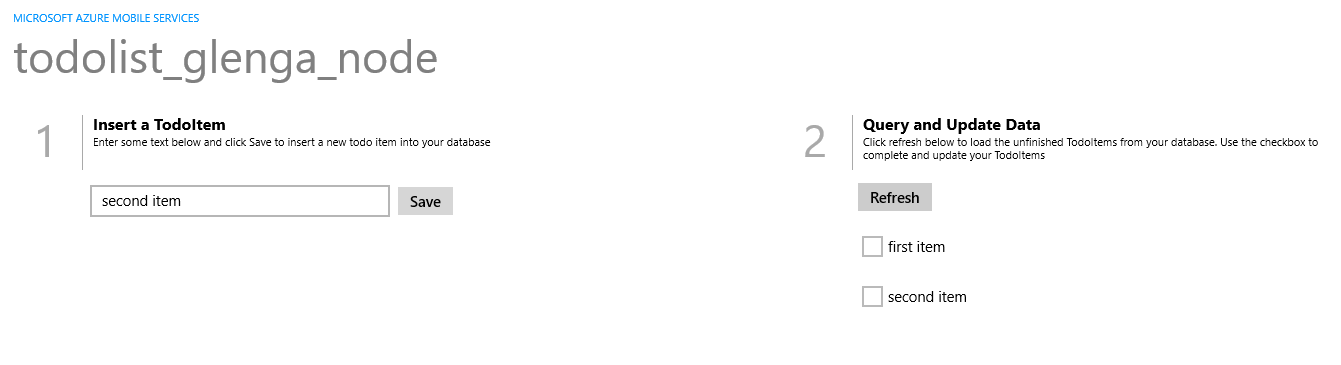
1. In Visual Studio once you have opened the solution, do the following:
   1. Right click on Solution -> Nuget Package Manager Manager -> Install Windows Azure Mobile Service Nuget package
   2. Under Shared folder -> App.xaml -> App.xaml.cs: Uncomment the “using Microsoft.WindowAzure.MobileServices”
   3. From the previous step, add the copied App Connection String to the ‘sealed partial class App’. This ensures that the solution can ‘talk’ to the cloud.
   4. Double click on MainPage.cs:
      1. Add “using Microsoft.WindowsAzure.MobileServices;”
      2. Follow the instructions that are marked as ‘TODO’ in comments. You will be essentially using MobileServiceCollection objects (which will directly interact with the cloud) instead of using ObservableCollection objects( which was using local storage).
      3. You will be converting the regular methods to ‘async’ methods which is required for asynchronous functions when using the cloud
      4. Replace this:

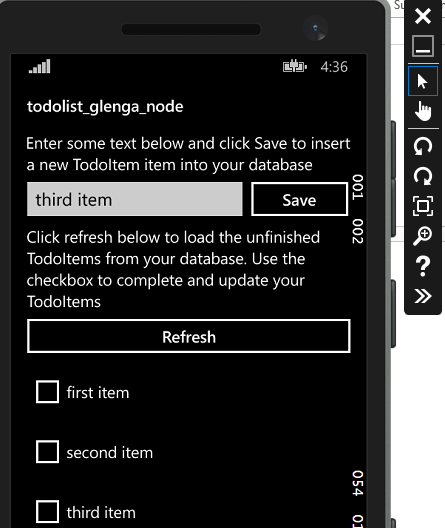
private IMobileServiceTable<TodoItem> todoTable = MobileService.GetTable<TodoItem>();

with:

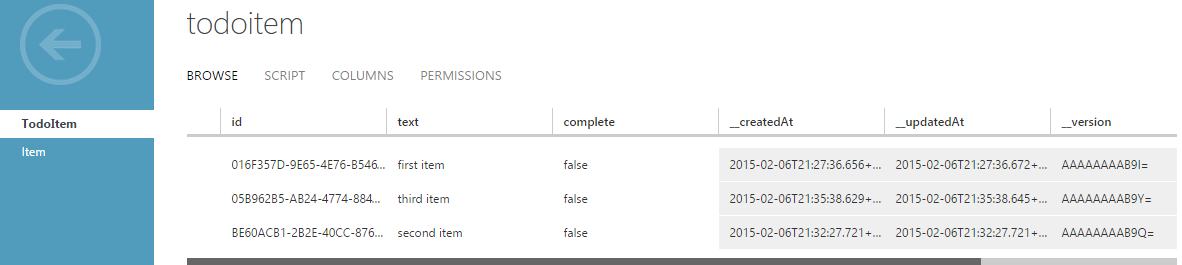
private IMobileServiceTable<TodoItem> todoTable = App.MobileService.GetTable<TodoItem>();

* 1. Go to DataModel -> TodoItem.cs and uncomment the JSON properties
  2. Build and run the solution. Add a few items to your ToDo list, If you have the Store App as the Start Project, you will see this:



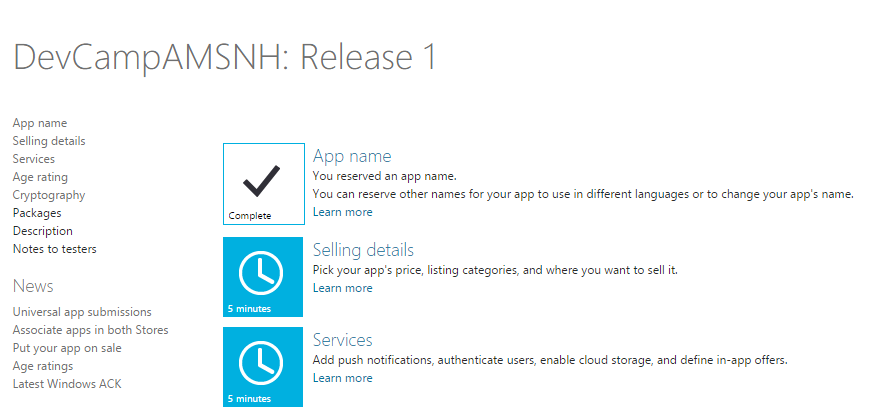
* 1. If you have the phone as your starter project, you will see this:

Note: you will see the same data across the form factors as both are accessing the same storage. If we login to our Azure portal and have a look at the data in our mobile services, we will notice this data:

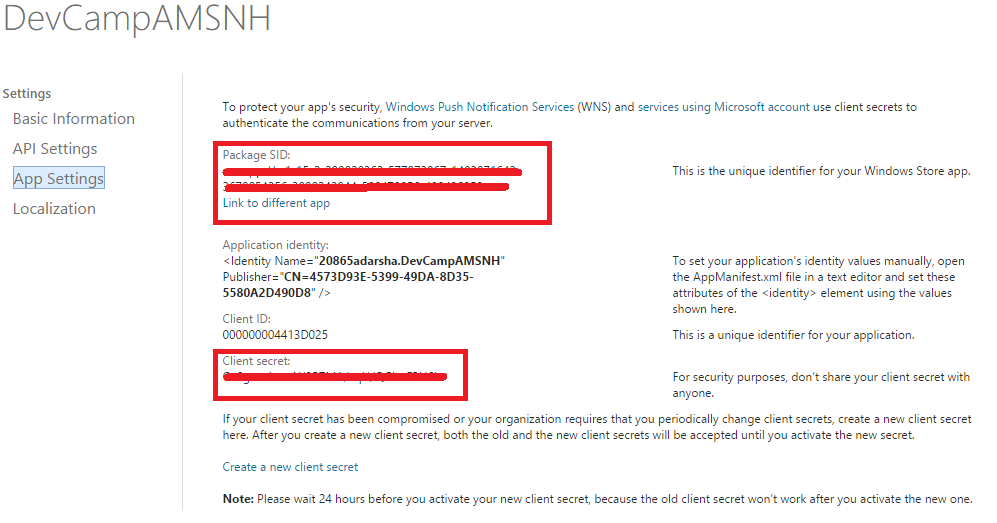


Step 3: Adding Notification to your app:

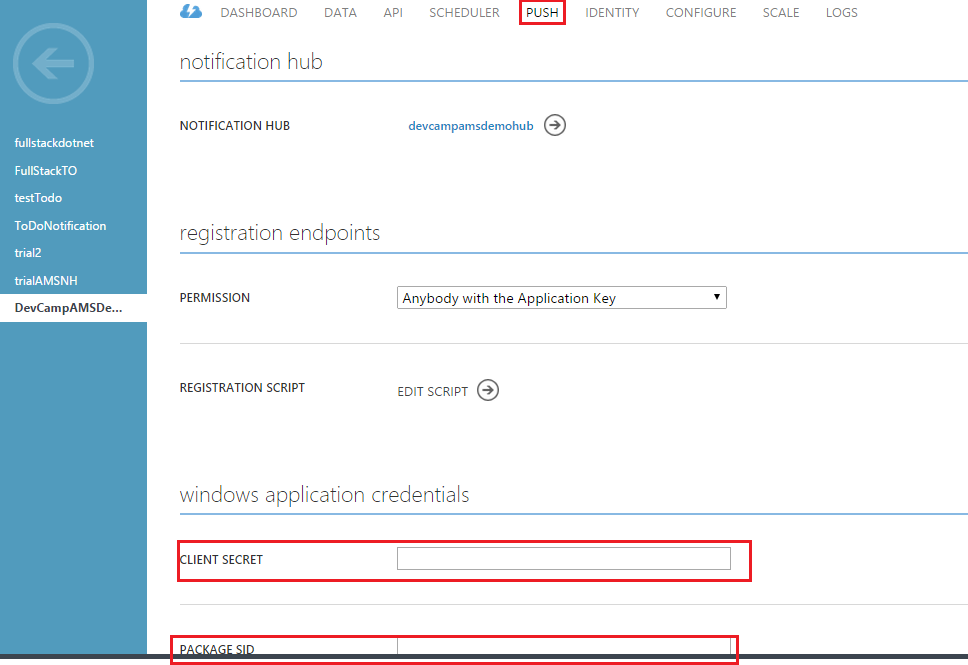
1. Register your app with [Windows Dec Center](https://appdev.microsoft.com/StorePortals/en-US/Developer/Catalog/ReleaseAnchor?wa=wsignin1.0). Click on App Name -> enter App name and click on ‘reserve your app name’ -> Save



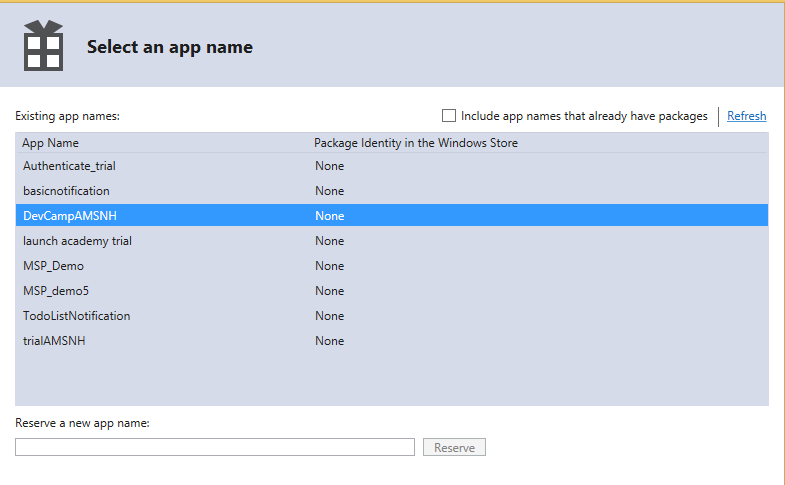
1. Click on Services-> Live Services Site and make a note of the Package Id and client Secret:



1. Go to the Azure Management Portal -> your created mobile service -> push -> Windows application credentials -> enter the above values -> Save



1. Back in the project solution of Visual Studio, right click on Project and click on Store -> Associate App with Store. Enter details and associate your app with the app store.



1. Update the App to register for notification.
   1. In App.xamls.cs add the following using statements:

using Windows.Networking.PushNotifications;

using Windows.UI.Popups;

1. Add the following code to the app class to retrieve the Channel URI for the app from WNS

private async void InitNotificationsAsync()

{

var channel = await PushNotificationChannelManager.CreatePushNotificationChannelForApplicationAsync();

var hub = new NotificationHub("<hub name>", "<connection string with listen access>");

var result = await hub.RegisterNativeAsync(channel.Uri);

// Displays the registration ID so you know it was successful

if (result.RegistrationId != null)

{

var dialog = new MessageDialog("Registration successful: " + result.RegistrationId);

dialog.Commands.Add(new UICommand("OK"));

await dialog.ShowAsync();

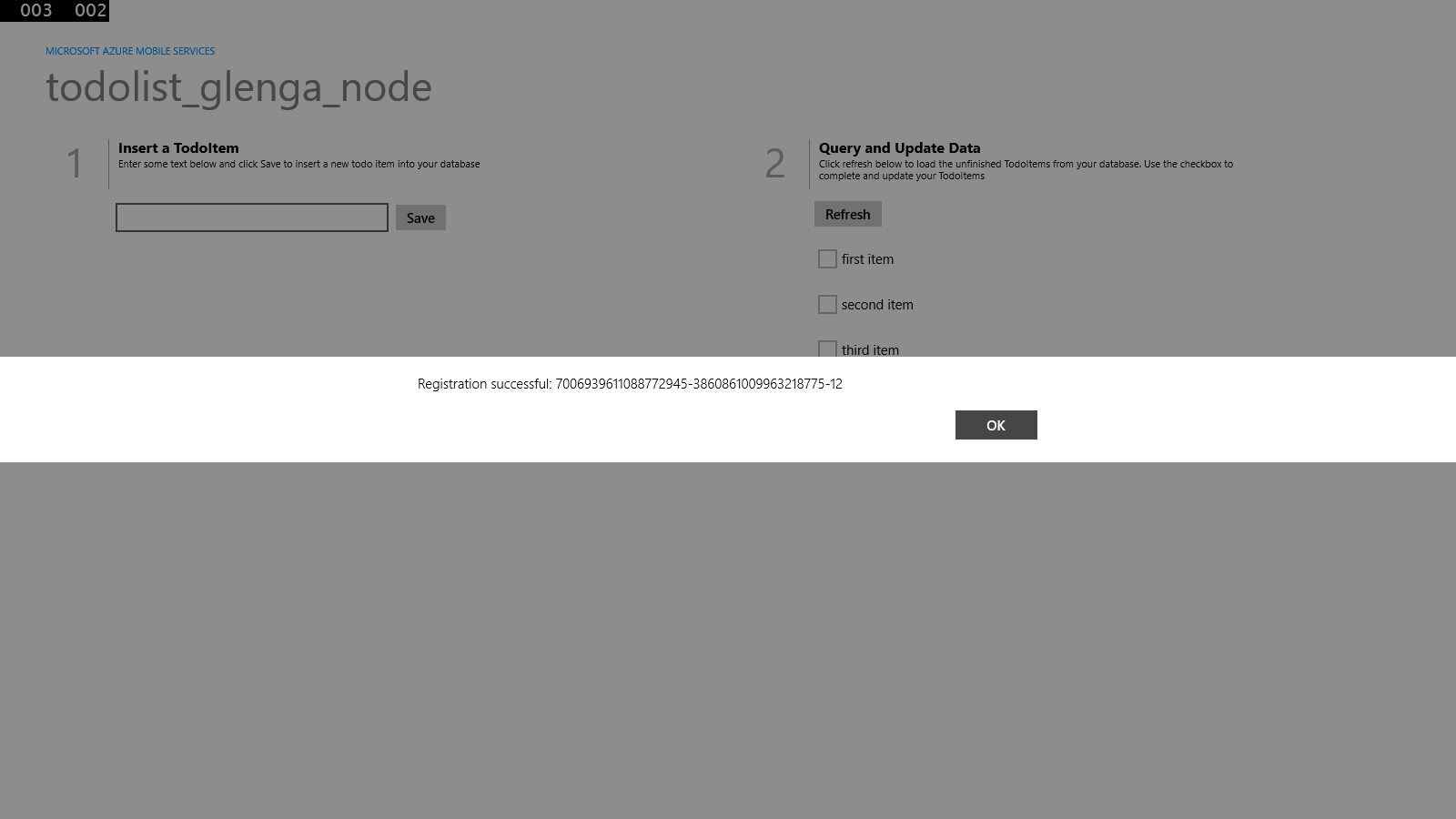
}

}

1. Add the following in OnLaunched() which initiates a registration with WNS every time the app is launched:

InitNotificationsAsync();

1. Now if we run the program, we will get a pop up which shows the Registration ID, showing that the app was registered successfully.



1. Go to App manifest of each project and set toast notification to true.
2. Now that the app has registered successfully, let us send a notification from the backend server as soon as there is an item added to the list. Go to the portal -> Azure Mobile Service -> <select your mobile service> -> Data -> <Select the table> -> Script -> Operation: select ‘Insert’. Replace the existing script with the following:

function insert(item, user, request) {

// Define a payload for the Windows Store toast notification.

var payload = '<?xml version="1.0" encoding="utf-8"?><toast><visual>' +

'<binding template="ToastText01"> <text id="1">' +

item.text + '</text></binding></visual></toast>';

request.execute({

success: function() {

// If the insert succeeds, send a notification.

push.wns.send(null, payload, 'wns/toast', {

success: function(pushResponse) {

console.log("Sent push:", pushResponse);

request.respond();

},

error: function (pushResponse) {

console.log("Error Sending push:", pushResponse);

request.respond(500, { error: pushResponse });

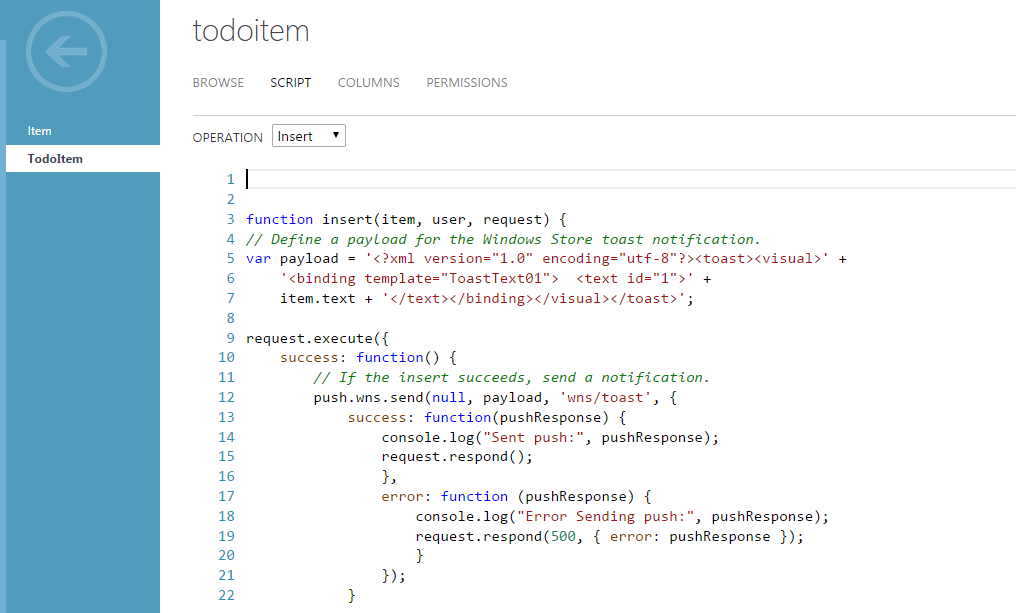
}

});

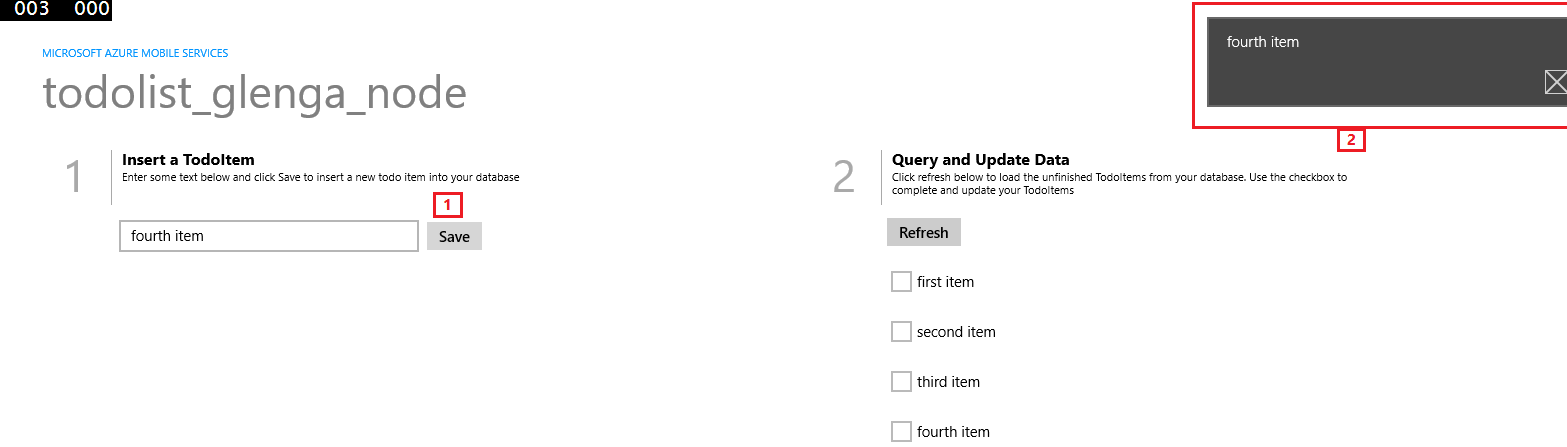
}

});

}



1. Now if you insert an item, you will be able to see a toast notification.

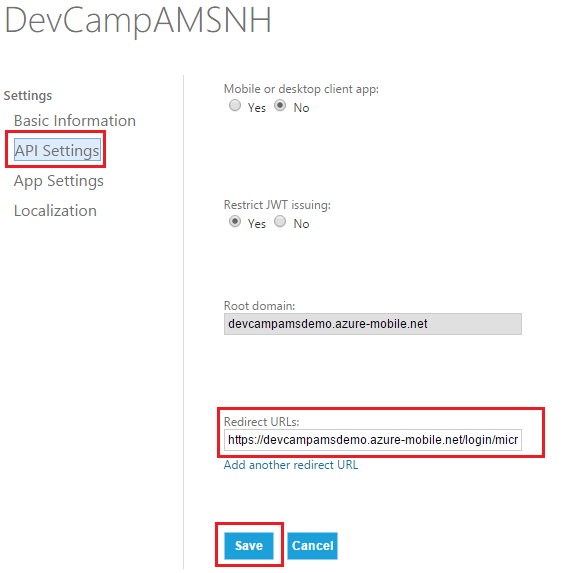


Step 4: Add authentication to your app

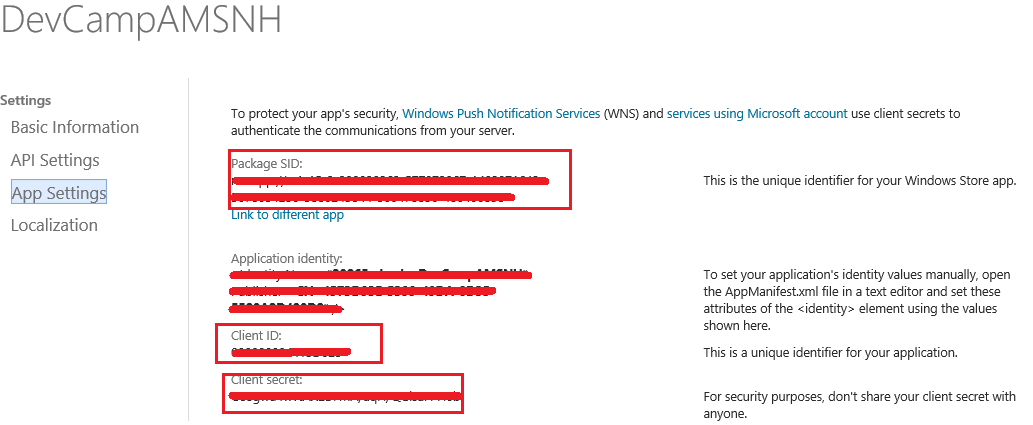
(We are going to authenticate our app using a Microsoft Live Id, however you have the option to do it with Twitter, Facebook, Google, Active Directory)

1. Let us go back to [Windows Dec Center](https://appdev.microsoft.com/StorePortals/en-US/Developer/Catalog/ReleaseAnchor?wa=wsignin1.0). As shown above, navigate to Dashboard -> <choose your app name> -> Services -> Live Services Site -> API Settings. Fill in the Redirect Details as:

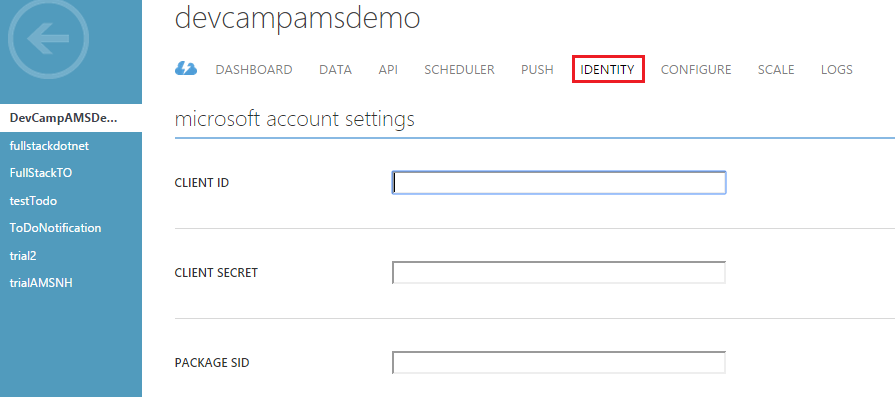
https://<AzureMobileServiceName>.azure-mobile.net/login/microsoftaccount



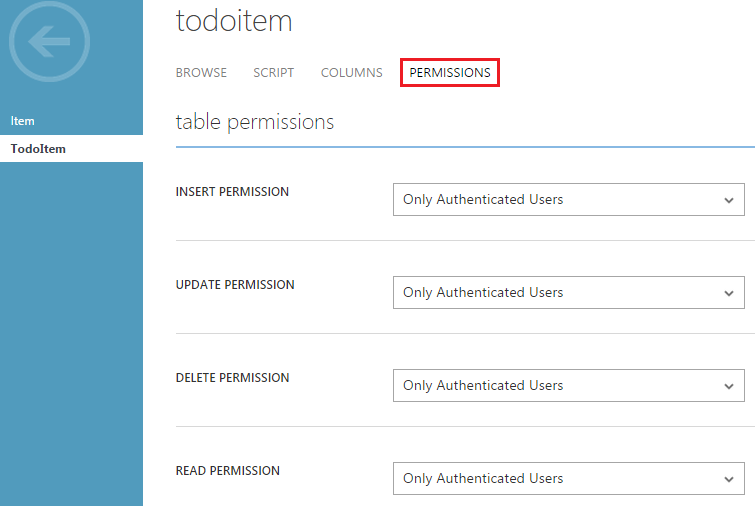
1. Let us go back to [Windows Dec Center](https://appdev.microsoft.com/StorePortals/en-US/Developer/Catalog/ReleaseAnchor?wa=wsignin1.0). As shown above, navigate to Dashboard -> <choose your app name> -> Services -> Live Services Site -> App Settings. Make a note of Package Id, Client Id and Client Secret.



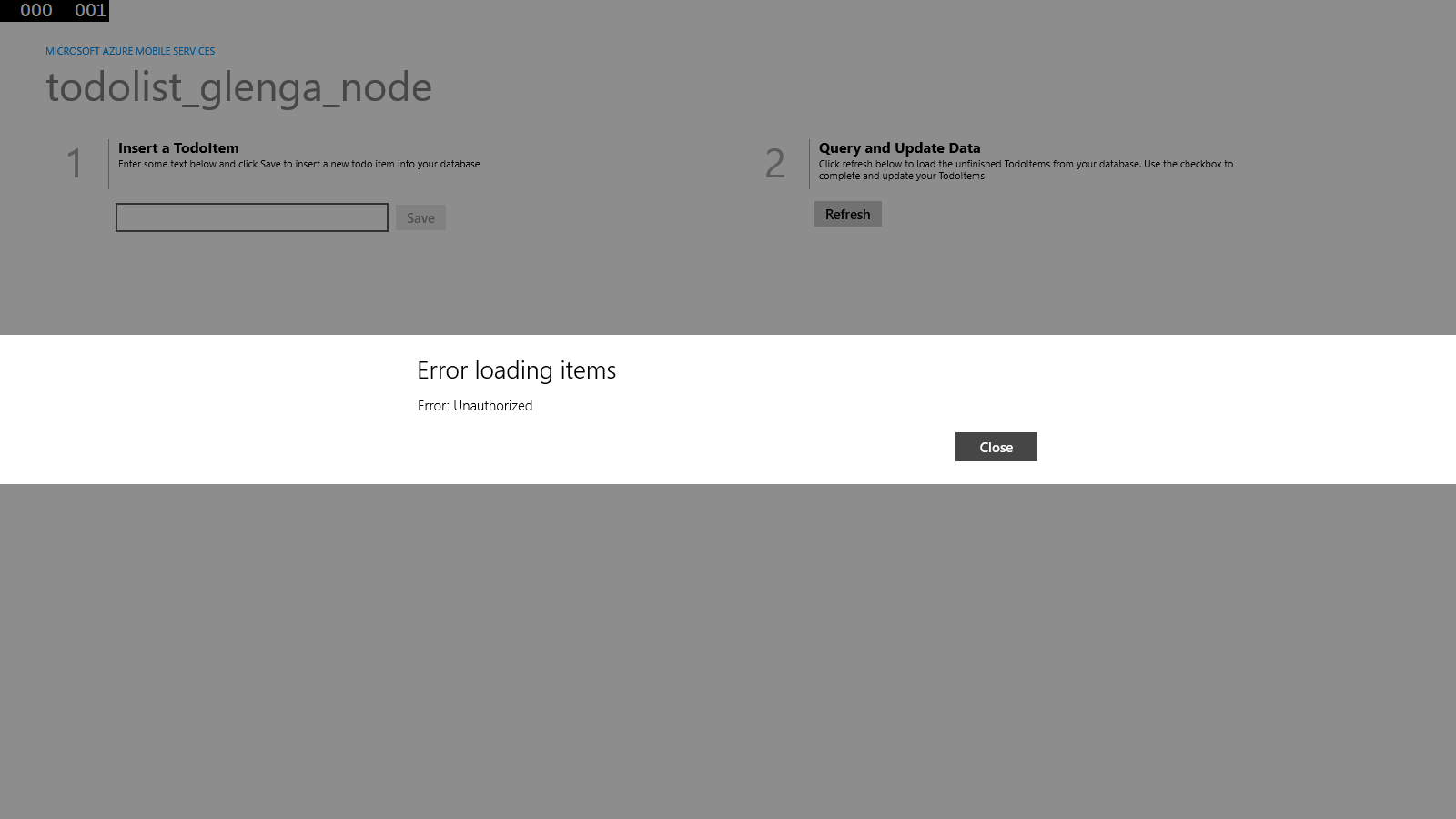
1. Go to the portal: manage.windowsazure.com -> Mobile Services-> Created Mobile Service used by App -> Identity. Enter the values from above.



1. Go to Data Tab -> todoitem table -> Permissions. Change the permissions of all the operations to “Only Authenticated Users” -> Save



1. In Visual Studio, now if we execute the solution, we will get an authentication error.



1. Now, lets the authentication piece to ensure authenticated access to data:
   1. In shared folder, add the following in MainPage.cs

Using Windows.UI.Popups;

* 1. In the MainPage.cs, in the MainPage Class, add the following:

// Define a member variable for storing the signed-in user.

private MobileServiceUser user;

// Define a method that performs the authentication process

// using a Facebook 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 Facebook authentication.

user = await App.MobileService

.LoginAsync(MobileServiceAuthenticationProvider.Facebook);

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();

}

}

1. In OnNavigatedTo, comment out RefreshTodoItems
2. Add the following to MainPage Class:

private async void ButtonLogin\_Click(object sender, RoutedEventArgs e)

{

// Login the user and then load data from the mobile service.

await AuthenticateAsync();

// Hide the login button and load items from the mobile service.

this.ButtonLogin.Visibility = Windows.UI.Xaml.Visibility.Collapsed;

RefreshTodoItems();

}

1. In the Windows Store App, open MainPage.xaml and add the following Button before the Save button:

<Button Name="ButtonLogin" Click="ButtonLogin\_Click" Visibility="Visible">Sign in</Button>

1. Open the shared App.xaml.cs, add the following code:

protected override void OnActivated(IActivatedEventArgs args)

{

// Windows Phone 8.1 requires you to handle the respose from the WebAuthenticationBroker.

#if WINDOWS\_PHONE\_APP

if (args.Kind == ActivationKind.WebAuthenticationBrokerContinuation)

{

// Completes the sign-in process started by LoginAsync.

// Change 'MobileService' to the name of your MobileServiceClient instance.

App.MobileService.LoginComplete(args as WebAuthenticationBrokerContinuationEventArgs);

}

#endif

base.OnActivated(args);

}

1. Run the project, click on the Sign-In button. Now you will be redirected to enter your credentials, once you get authenticated, you will be able to access all the data.

