Azure Mobile Services on Windows Phone

This article explains how to get started with Azure Mobile Services on Windows Phone and provides tips to speed up your development.

30 Jun 2013

Note: There are a lot of good tutorials about using Azure, but unlike this article they are primary focused on Windows 8.

Introduction

Microsoft Windows Azure is a cloud computing platform and infrastructure for building, deploying and managing applications and services through a global network of Microsoft-managed data-centres. It provides both platform as a service (PaaS) and infrastructure as a service (IaaS) services, and supports many different programming languages, tools and frameworks (including both Microsoft-specific and third-party software and systems).

Mobile Services are a component of Azure. The services make it easier to:

- Store data to the cloud. The mobile services are a very easy way to store structured data in the cloud. The SQL database allows you to add a classification to add to a game, manage a list of friends and a lot more. You can connect to an existing database or create a new one.
- Include custom business logic with server-side script. Simple server-side scripts allow you to do things like send push notifications, SMS and email messages (Using SendGrig module). You can also configure scripts to run according to a fixed schedule that allows you to periodically delete old data or duplicated in tables, edit and resize images provided by the user, and query for data aggregated by an external Web service.
- Manage users. Incorporate user authentication from Facebook, Twitter, Google or Microsoft accounts.
- Push Notifications. Mobile services have an integrated push system and server-side scripting, providing an easy way to send push notifications to the application without having to write, test, or manage back-end infrastructure code.

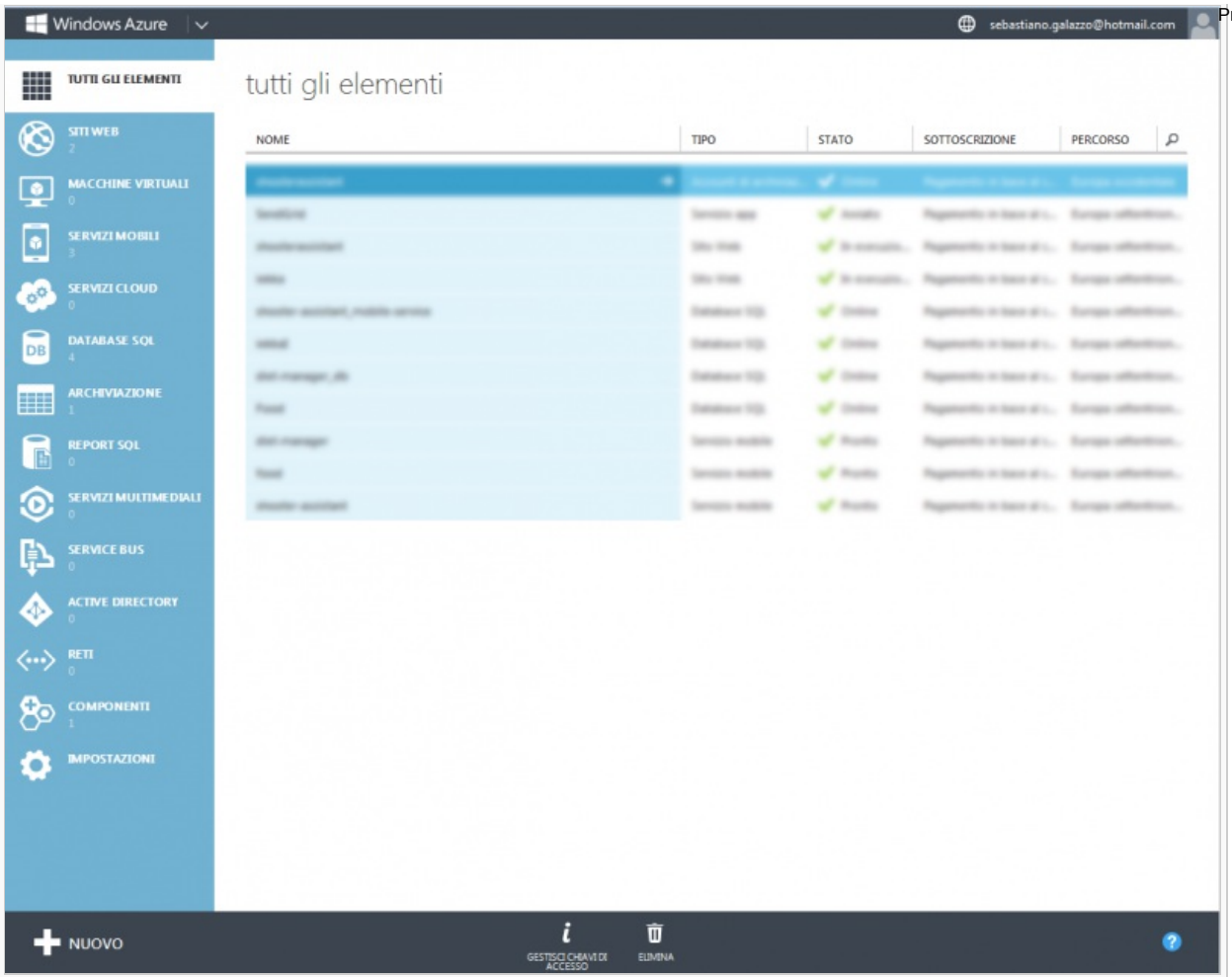
Mobile Services fully support the development for the Windows Store, Windows Phone, iOS and Android and REST API. They are **free** for the first ten services with **1GB** of storage available. More than enough to build complex mobile application.

This article is focused on data storage and management. We will create both a new mobile service and a simple "To do list" app that stores app data in the new mobile service.

Get Started

Assuming you already have a Microsoft account, login to Windows Azure Management Portal. The first screen displayed is the Management Panel:

(C) Copyright Nokia 2014. All rights reserved.



Windows Azure Management Portal

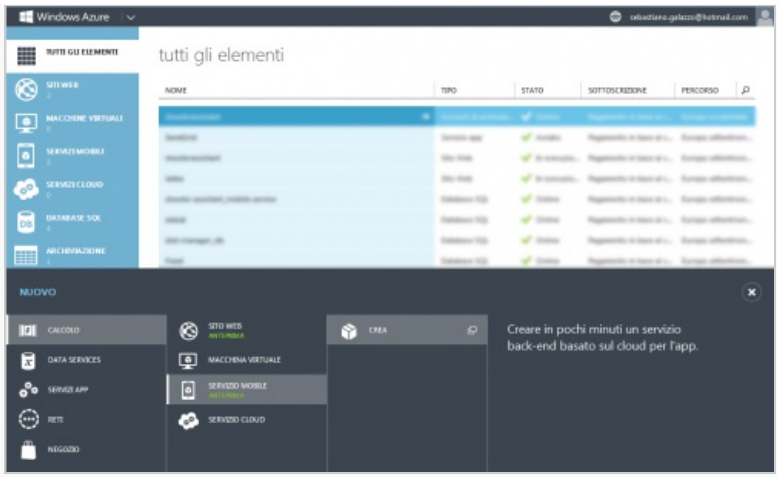
Create a new Mobile Service

At the bottom of the navigation pane, click **+NEW**.

Create new mobile service button

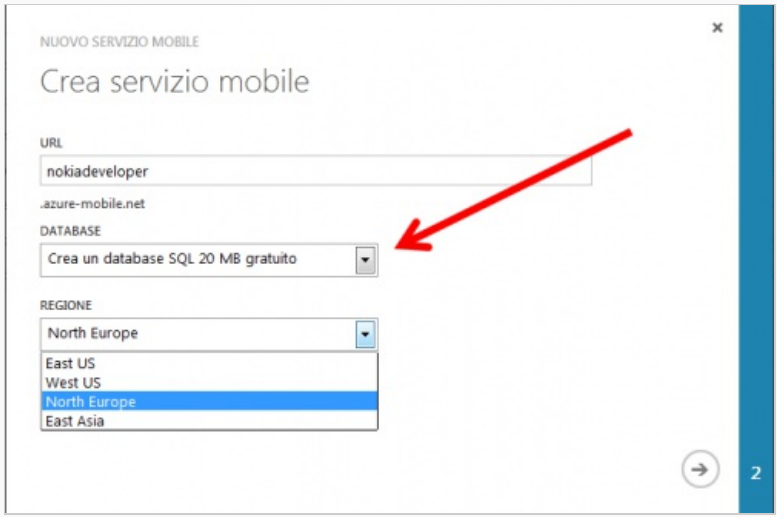
Expand **Compute** and **Mobile Service**, then click **Create**.

(C) Copyright Nokia 2014. All rights reserved.



Expand Compute and Mobile Service

In the **Create a mobile service** page, type a subdomain name for the new mobile service in the URL text box and wait for name verification. Once name verification completes, click the right arrow button to go to the next page.

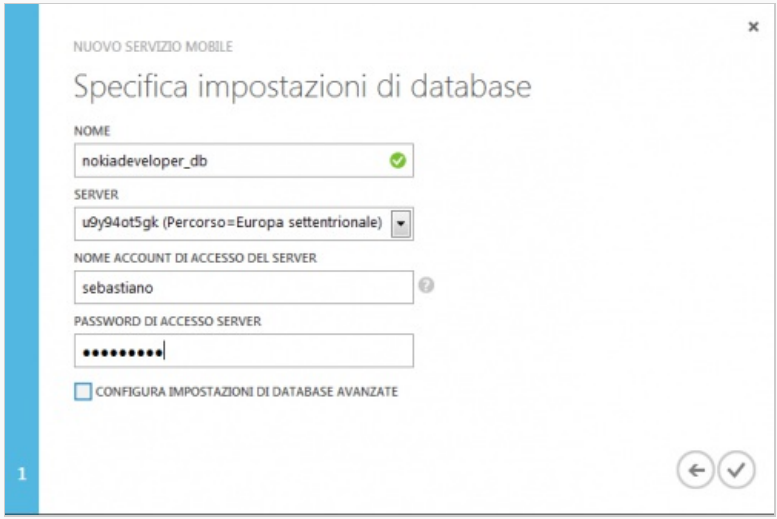


Create a mobile service page

When you create a new Mobile Service, you create a new SQL Database instance and server. You can reuse this new database and administer it as you would any other SQL Database instance. If you already have a database in the same region as the new mobile service, you can instead choose **Use existing Database** and then select that database. The use of a database in a different region is not recommended because of additional bandwidth costs and higher latencies.

To help us during development stages, Azure provide a **20MB free** Database which we will use in this tutorial.

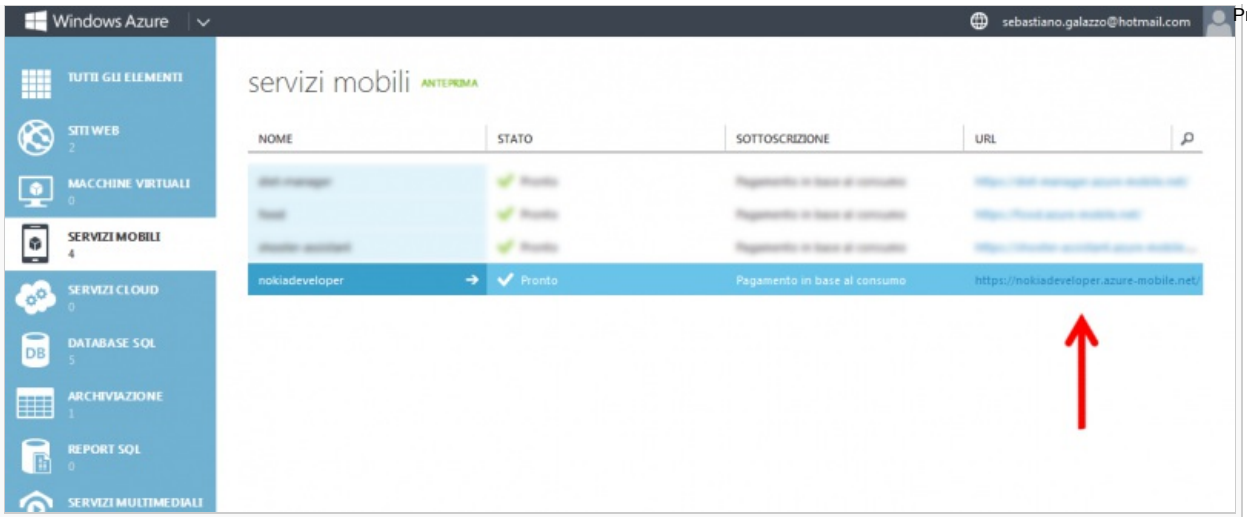
In the database settings page we set all Db parameters:



Database settings page

The Mobile Service is now created and and it appears in your list

(C) Copyright Nokia 2014. All rights reserved.



Mobile services list

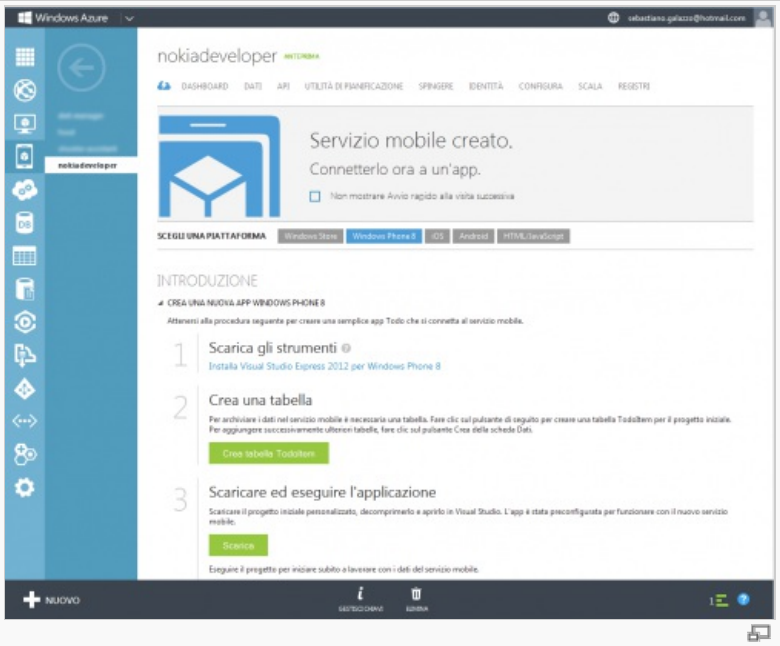
Note: A public URL was created https://nokiadeveloper.azure-mobile.net/ to be used with REST API

Creating the Windows Phone application

Once you have created your mobile service, you can follow an easy quickstart in the Management Portal to either create a new app or modify an existing app to connect to your mobile service. In this section you will create a new Windows Phone 8 app that is connected to your mobile service.

In the Management Portal, click **Mobile Services**, and then click the mobile service that you just created.

In the quickstart tab, click **Windows Phone 8** under *Choose platform* and expand **Create a new Windows Phone 8 app**.

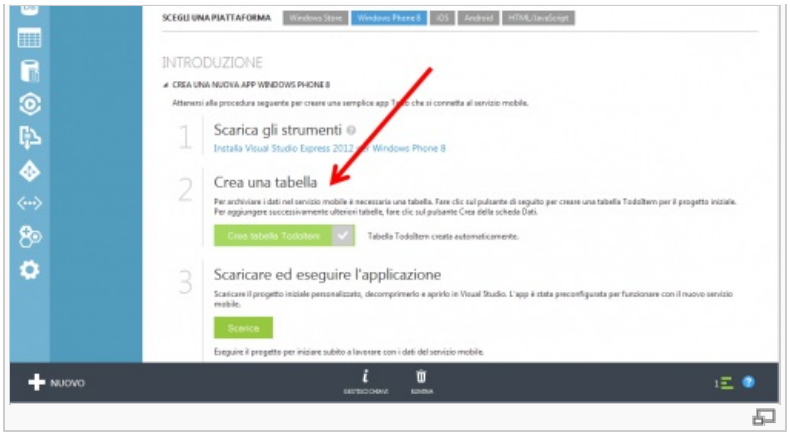


Tip: I highly recommend you install Microsoft SQL Server Management Studio Express (SSMSE) because it is fundamental to perform some operation like importing data (i.e. from an Excel file) and an easier way to execute management queries.

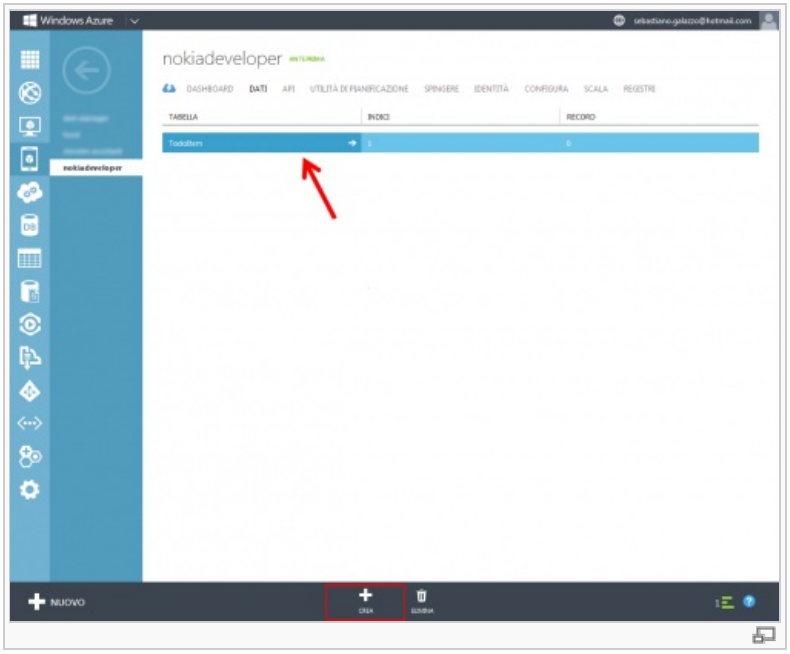
Creating a new Table

For our tutorial we create a sample Table by clicking **Create TodoItem table**

(C) Copyright Nokia 2014. All rights reserved.

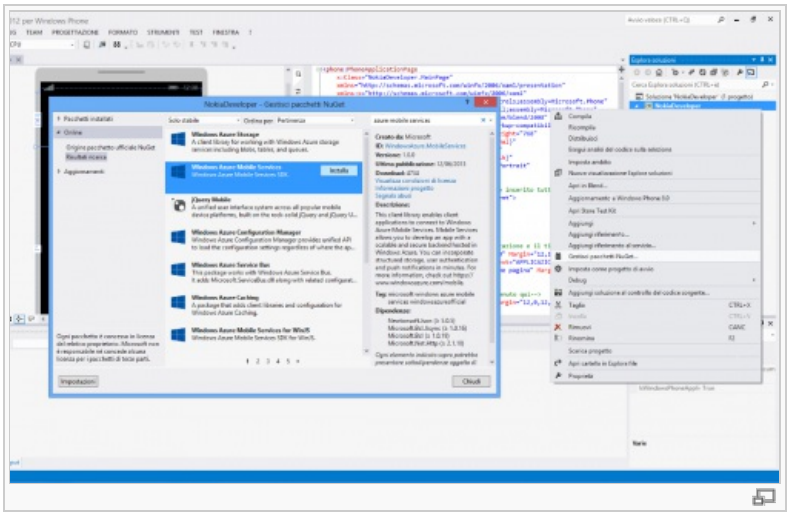


Accessing the DATA tab we can manage, create or delete our tables



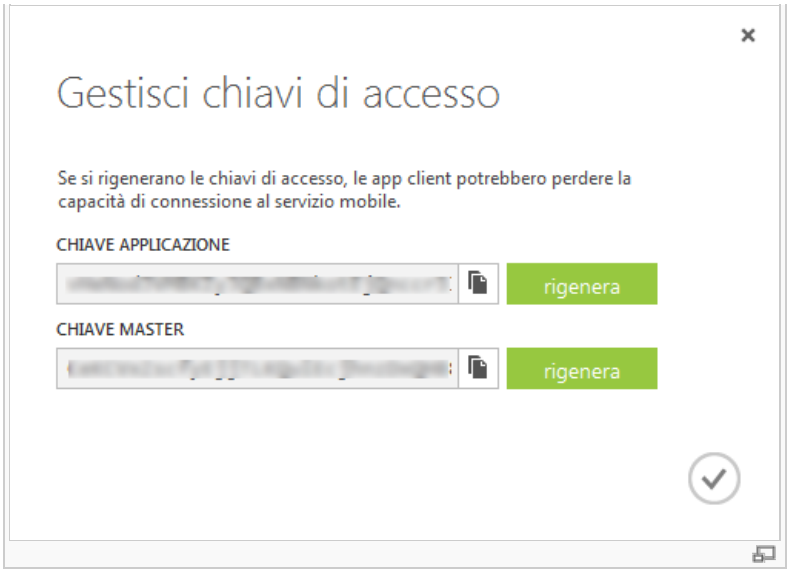
Open Visual Studio and create a new project (in this case named 'NokiaDeveloper'). Right click on the project name in the upper right corner, select **Manage NuGet packages**, search for **Azure Mobile Services** and install the package.

All dependencies will be installed.



In order to access to Mobile Services from our application we need **Access Keys**. In the Dashboard tab click the **Manage Keys** button you can find in the bottom of the page.

(C) Copyright Nokia 2014. All rights reserved.



Start coding

Open **App.xaml.cs** and add the namespace **Microsoft.WindowsAzure.MobileServices** and the following code:

```
...
using System.Windows.Shapes;
using Microsoft.Phone.Controls;
using Microsoft.Phone.Shell;

using Microsoft.WindowsAzure.MobileServices;

namespace NokiaDeveloper
{
    public partial class App : Application
    {
        /// <summary>
        /// Offre facile accesso al frame radice dell'applicazione Windows Phone.
        /// </summary>
        /// <returns>Nome radice dell'applicazione Windows Phone.</returns>
        public PhoneApplicationFrame RootFrame { get; private set; }

        public static MobileServiceClient MobileService = new
MobileServiceClient("https://nokiadeveloper.azure-mobile.net/", "YOUR KEY HERE");

        /// <summary>
        /// Costruttore dell'oggetto Application.
        /// </summary>
        public App()
        {
            // Global handler for uncaught exceptions.
            UnhandledException += Application_UnhandledException;
...
```

In the **MainPage.xaml.cs** we can perform all our operations.

First of all we need to create a new class to represent our tables into Mobile Service. Having created the sample table (in the service) the class will be the following:

```
namespace NokiaDeveloper
{
    public class Item {
```


(C) Copyright Nokia 2014. All rights reserved.

```
        public int Id { get; set; }
        public string Text { get; set; }
    }

    public partial class MainPage : PhoneApplicationPage
    {
        // Costruttore
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

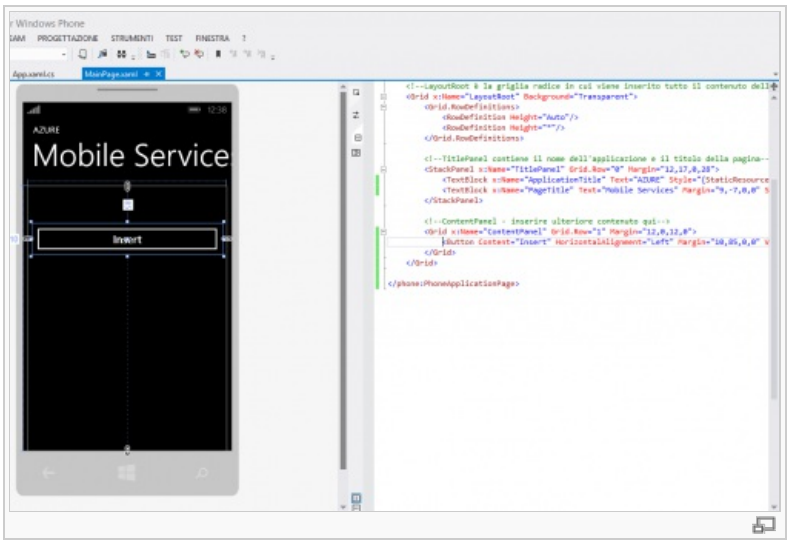
It's very important that all classes you create have the field `Id`.

When creating a new table there are no columns and the *Dynamic Schema* option is enabled. When dynamic schema is enabled on your mobile service, *new columns are created automatically* when JSON objects are sent to the mobile service by an insert or update operation.

This means that performing an insert operation creates any new fields specified if they do not already exist!

Insert a new item

Create a new `Button` on your application (called **Insert** in this case). We're using the button to click to trigger the insertion of our new item.



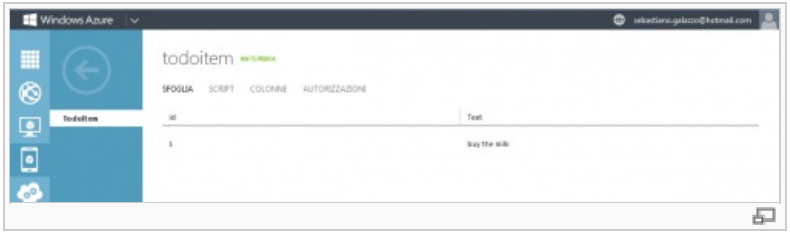
Manage the `Click` event with the following code:

```
using Microsoft.WindowsAzure.MobileServices;

namespace NokiaDeveloper
{
    public class TodoItem {
        public int Id { get; set; }
        public string Text { get; set; }

        public TodoItem()
        {

        }
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    public partial class MainPage : PhoneApplicationPage
    {
        TodoItem item = new TodoItem();

        // Costruttore
        public MainPage()
        {
            InitializeComponent();
        }

        private async void Insert(TodoItem item)
        {
            try
            {
                await
App.MobileService.GetTable<TodoItem>().InsertAsync(item).ContinueWith(t =>
                {
                    if (t.IsFaulted)
                    {
                        Dispatcher.BeginInvoke(() => MessageBox.Show("\nInsert
failed!\n") );
                    }
                    else
                    {
                        Dispatcher.BeginInvoke(() => MessageBox.Show("\nInserted a new
item with id:"+item.Id+"\n") );
                    }
                });
            }
            catch (System.Net.WebException e)
            {
                Dispatcher.BeginInvoke(() => MessageBox.Show("\nConnection failed!\n")
);
            }
        }

        private void InsertButton_Click(object sender, RoutedEventArgs e)
        {
            item.Text = "buy the milk";
            Insert(item);
        }

    }

}
```

As result you will find your record into the Mobile Service



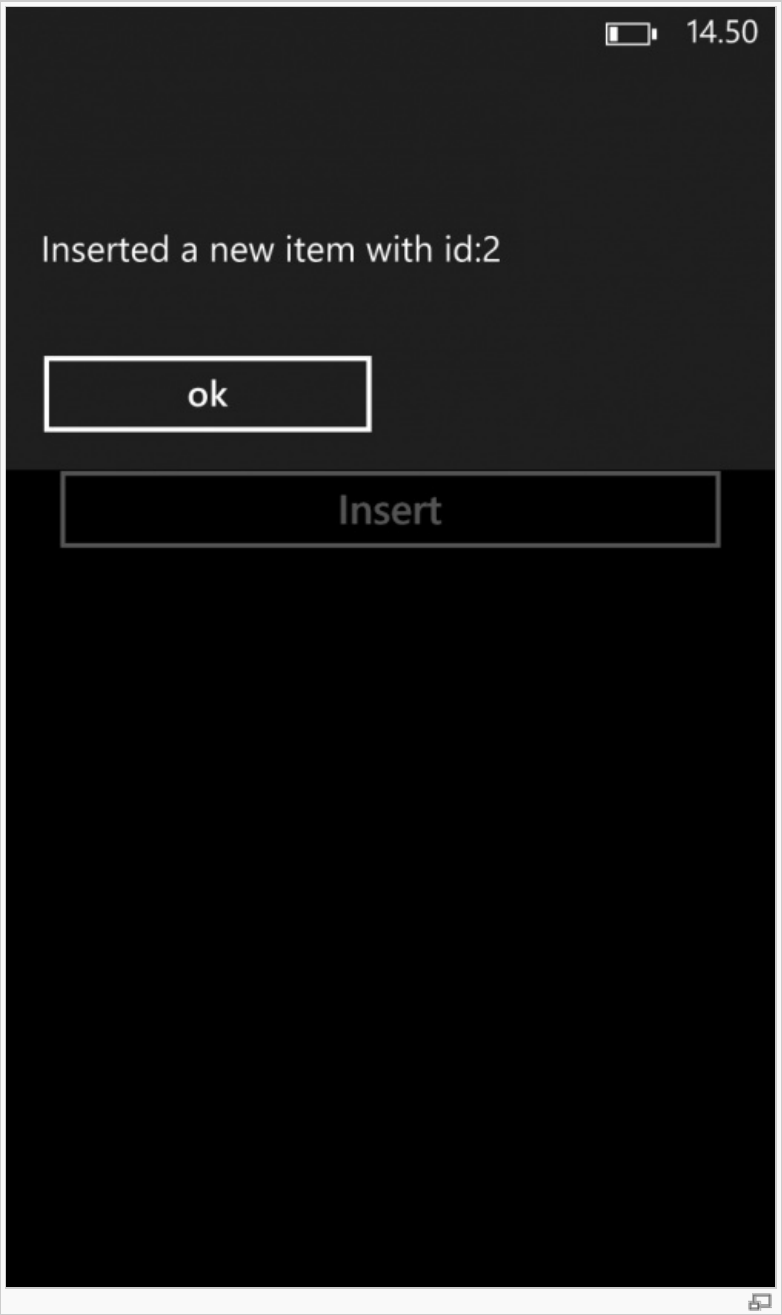
Note that all columns where added dynamically. You can do a test by adding a new field into `TodoItem` class and repeating the insert operation. A new column will be added and previous record will be filled with default value.

(C) Copyright Nokia 2014. All rights reserved.

You can always delete a new field or set a new index from the *Management Panel*.

Warning: It is very important when performing an insert operation you don't set the Id field as it has `auto_increment` property.

If the operation was successful the object you inserted will be returned with the `Id` field filled. This is the last inserted Id and is useful for further operations.



Update an item

We can perform the update operation using the following code:

```
private async void Update(TodoItem item)
        {
            try
            {
                await
App.MobileService.GetTable<TodoItem>().UpdateAsync(item).ContinueWith(t =>
                {
                    if (t.IsFaulted)
                    {
                        Dispatcher.BeginInvoke(() => MessageBox.Show("\nUpdate
failed!\n"));
```


(C) Copyright Nokia 2014. All rights reserved.

```
                    }
                    else
                    {
                        Dispatcher.BeginInvoke(() => MessageBox.Show("\nUpdated item
with id:" + item.Id + "\n"));
                    }
                });
            }
            catch (System.Net.WebException e)
            {
                Dispatcher.BeginInvoke(() => MessageBox.Show("\nConnection failed!\n"));
            }
        }

        private void UpdateButton_Click(object sender, RoutedEventArgs e)
        {
            item.Id = 3;
            item.Text = "buy lemons";
            Update(item);
        }
```

Selecting an item

We can perform the select operation using the following code:

```
private async void Select(TodoItem item)
        {
            try
            {
                await App.MobileService.GetTable<TodoItem>().Where(citem => citem.Id ==
item.Id).ToListAsync().ContinueWith(t =>
                {
                        if (t.IsFaulted)
                        {
                            System.Diagnostics.Debug.WriteLine("Faulted");
                            Dispatcher.BeginInvoke(() => MessageBox.Show("Select
failed!\n"));
                        }
                        else
                        {
                            System.Collections.Generic.List<TodoItem> items = t.Result;
                            int count = items.Count;
                            if (count > 0)
                            {
                                foreach(var record in items ) {
                                    Dispatcher.BeginInvoke(() =>
MessageBox.Show(record.Text));
                                }
                            }
                            else
                            {
                                Dispatcher.BeginInvoke(() => MessageBox.Show("No record
found!\n"));
                            }
                        }
                });
```


(C) Copyright Nokia 2014. All rights reserved.

```
            }
            catch (System.Net.WebException e)
            {
                Dispatcher.BeginInvoke(() => MessageBox.Show("\nConnection failed!\n")
);
            }
        }

        private void SelectButton_Click(object sender, RoutedEventArgs e)
        {
            item.Id = 3;
            Select(item);
        }
```

Deleting an Item

We can perform the delete operation using the following code:

```
private async void Delete(TodoItem item)
        {
            try
            {
                await
App.MobileService.GetTable<TodoItem>().DeleteAsync(item).ContinueWith(t =>
                {
                    if (t.IsFaulted)
                    {
                        Dispatcher.BeginInvoke(() => MessageBox.Show("\nDelete
failed!\n"));
                    }
                    else
                    {
                        Dispatcher.BeginInvoke(() => MessageBox.Show("\nItem deleted"));
                    }
                });
            }
            catch (System.Net.WebException e)
            {
                Dispatcher.BeginInvoke(() => MessageBox.Show("\nConnection failed!\n"));
            }
        }

        private void DeleteButton_Click(object sender, RoutedEventArgs e)
        {
            item.Id = 2;
            Delete(item);
        }
```

Reading Data

```
private async void Read()
        {
            try
            {
                await App.MobileService.GetTable<TodoItem>().ReadAsync().ContinueWith(t
```


(C) Copyright Nokia 2014. All rights reserved.

```
=>
                {
                    if (t.IsFaulted)
                    {
                        System.Diagnostics.Debug.WriteLine("Faulted");
                        Dispatcher.BeginInvoke(() => MessageBox.Show("Read failed!\n"));
                    }
                    else
                    {
                        System.Collections.Generic.IEnumerable<TodoItem> items =
t.Result;
                        foreach (var record in items)
                        {
                            Dispatcher.BeginInvoke(() => MessageBox.Show(record.Text));
                        }
                    }
                });
            }
            catch (System.Net.WebException e)
            {
                Dispatcher.BeginInvoke(() => MessageBox.Show("\nConnection failed!\n"));
            }
        }

        private void ReadButton_Click(object sender, RoutedEventArgs e)
        {
            Read();
        }
```

Filtering Data

```
private async void Take(int count)
        {
            try
            {
                await
App.MobileService.GetTable<TodoItem>().Take(count).ToListAsync().ContinueWith(t =>
                {
                    if (t.IsFaulted)
                    {
                        System.Diagnostics.Debug.WriteLine("Faulted");
                        Dispatcher.BeginInvoke(() => MessageBox.Show("Take failed!\n"));
                    }
                    else
                    {
                        System.Collections.Generic.IEnumerable<TodoItem> items =
t.Result;
                        foreach (var record in items)
                        {
                            Dispatcher.BeginInvoke(() => MessageBox.Show(record.Text));
                        }
                    }
                });
            }
            catch (System.Net.WebException e)
```


(C) Copyright Nokia 2014. All rights reserved.

```
            {
                Dispatcher.BeginInvoke(() => MessageBox.Show("\nConnection failed!\n"));
            }
        }

        private void TakeButton_Click(object sender, RoutedEventArgs e)
        {
            Take(2);
        }
```

Pagination - Skipping items when reading

Is it possible to combine the methods in order to perform more complex operations like to skip some elements and take a limited number of record starting from that point.

This method help us to perform paging operation when you work with a large amount of data.

```
private async void SkipTake(int skip, int take)
        {
            try
            {
                await
App.MobileService.GetTable<TodoItem>().Skip(skip).Take(take).ToListAsync().ContinueWith(t
=>
                {
                    if (t.IsFaulted)
                    {
                        System.Diagnostics.Debug.WriteLine("Faulted");
                        Dispatcher.BeginInvoke(() => MessageBox.Show("Skip - Take
failed!\n"));
                    }
                    else
                    {
                        System.Collections.Generic.IEnumerable<TodoItem> items =
t.Result;
                        foreach (var record in items)
                        {
                            Dispatcher.BeginInvoke(() => MessageBox.Show(record.Text));
                        }
                    }
                });
            }
            catch (System.Net.WebException e)
            {
                Dispatcher.BeginInvoke(() => MessageBox.Show("\nConnection failed!\n"));
            }
        }

        private void SkipTakeButton_Click(object sender, RoutedEventArgs e)
        {
            SkipTake(10, 20);
        }
```

Summary

Mobile Services are a powerful tool that can push our application to the next level. They provide not only a quick and professional

(C) Copyright Nokia 2014. All rights reserved.

service to manage data, but we can manage also:

- Push Notification
- Manage login using Microsoft, Facebook, Google account
- Create a custom API. Using the **Node.js** engine we can create our own web server, responding at the public URL created (https://nokiadeveloper.azure-mobile.net/) and interacting with HTTP protocol

The Azure team is actively adding new features and lowering (already low) costs. Many big companies have adopted Azure and my experience has been very positive.

(C) Copyright Nokia 2014. All rights reserved.