# Corso di Sistemi Distribuiti e Cloud Computing

Corso di Laurea Magistrale in Ingegneria Informatica A.A. 2019/2020
DIMES - Università degli Studi della Calabria

UNIVERSITÀ DELLA CALABRIA
Dipartimento di INGEGNERIA INFORMATICA, MODELLISTICA, ELETTRONICA E SISTEMISTICA

## DEVELOPMENT SERVICES ON WINDOWS AZURE

## STORE DATA IN TABLES

## STORE DATA IN BLOBS

## REST WEB SERVICES API

Ing. Loris Belcastro

## Summary

- Store data in tables
- Store data in blobs

## Summary

- Store data in tables

- Store data in blobs

## The Table Service

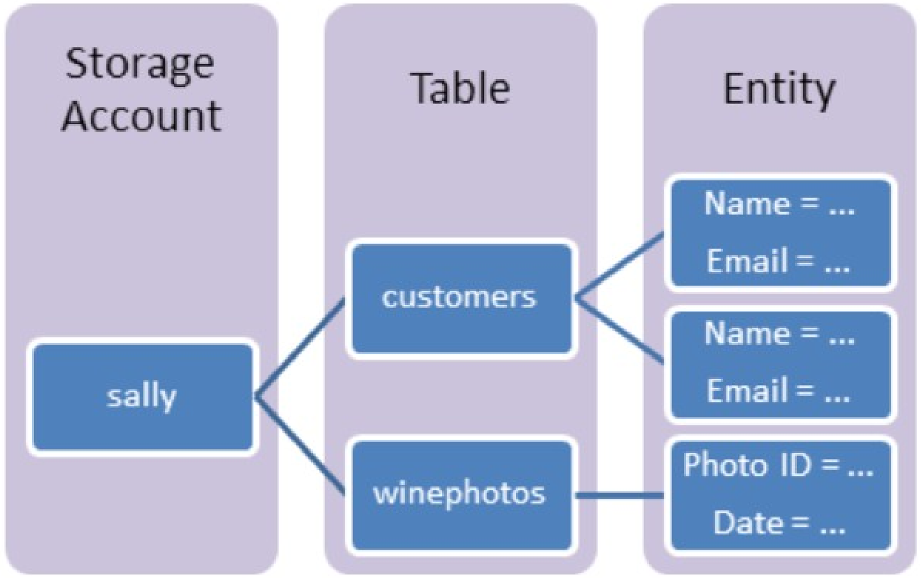
- The Azure Table storage service stores large amounts of structured data.
  - tables are ideal for storing structured, non-relational data
  - the service is a NoSQL datastore
  - accepts authenticated calls from inside and outside the Azure cloud

- Table service can be exploited to:
  - store TBs of structured data capable of serving web scale applications
  - store datasets that don't require complex joins, foreign keys, or stored procedures and can be denormalized for fast access
  - quickly query data using a clustered index
  - access data using the OData protocol and LINQ queries with WCF Data Service .NET Libraries

- Following the Cloud philosophy, tables will scale as demand increases

## Concepts

- **Storage Account:** it needs to access to any Azure Storage service.
- **Table**: a collection of entities, no table schema enforced (i.e., a single table can contain entities that have different sets of properties).
- **Entity:** a set of properties, like a database row (size up to 1MB).
- **Property:** a name-value pair. Each entity can include up to 252 properties to store data. Each entity also has 3 system properties that specify a partition key, a row key, and a timestamp. Entities with the same partition key can be queried more quickly, and inserted/updated in atomic operations. An entity's row key is its unique identifier within a partition



## Preliminary steps

- Obtain the assembly (by NuGet)
- Create and configure a ConnectionString (i.e, ‘StorageConnectionString’)

# Entity class

- Entities are custom classes derived from TableEntity
  - The entity class defines the properties of the entity
  - Each entity class (or type) must expose a parameter-less constructor.
  - About properties:
    - Row key
    - Partition key
      - Entities with the same partition key can be queried faster
      - Nevertheless, using diverse partition keys allows for greater parallel operation scalability.
    - Other properties…, but each one must be public and of a supported type
    - Each property must expose both get and set

## Entity class – Customer & Employee

```
//class CustomerEntity
    public class CustomerEntity : TableEntity {
       public CustomerEntity(string lastName, string firstName) {
            this.PartitionKey = lastName;
            this.RowKey = firstName;
        }
        public CustomerEntity() { }
        public string Email { get; set; }
        public string PhoneNumber { get; set; }
    }//class CustomerEntity

    //class EmployeeEntity
    public class EmployeeEntity : TableEntity {
        public EmployeeEntity() { }
        public EmployeeEntity(int id, string name, double sal) {
            Id = id;
            Name = name;
            Salary = sal;
            PartitionKey = id.ToString();
            RowKey = name;
        }
        public int Id { get; set; }
        public string Name { get; set; }
        public double Salary { get; set; }
    } //class EmployeeEntity
```

Auto-implemented property

# Code snippets – insert an entity

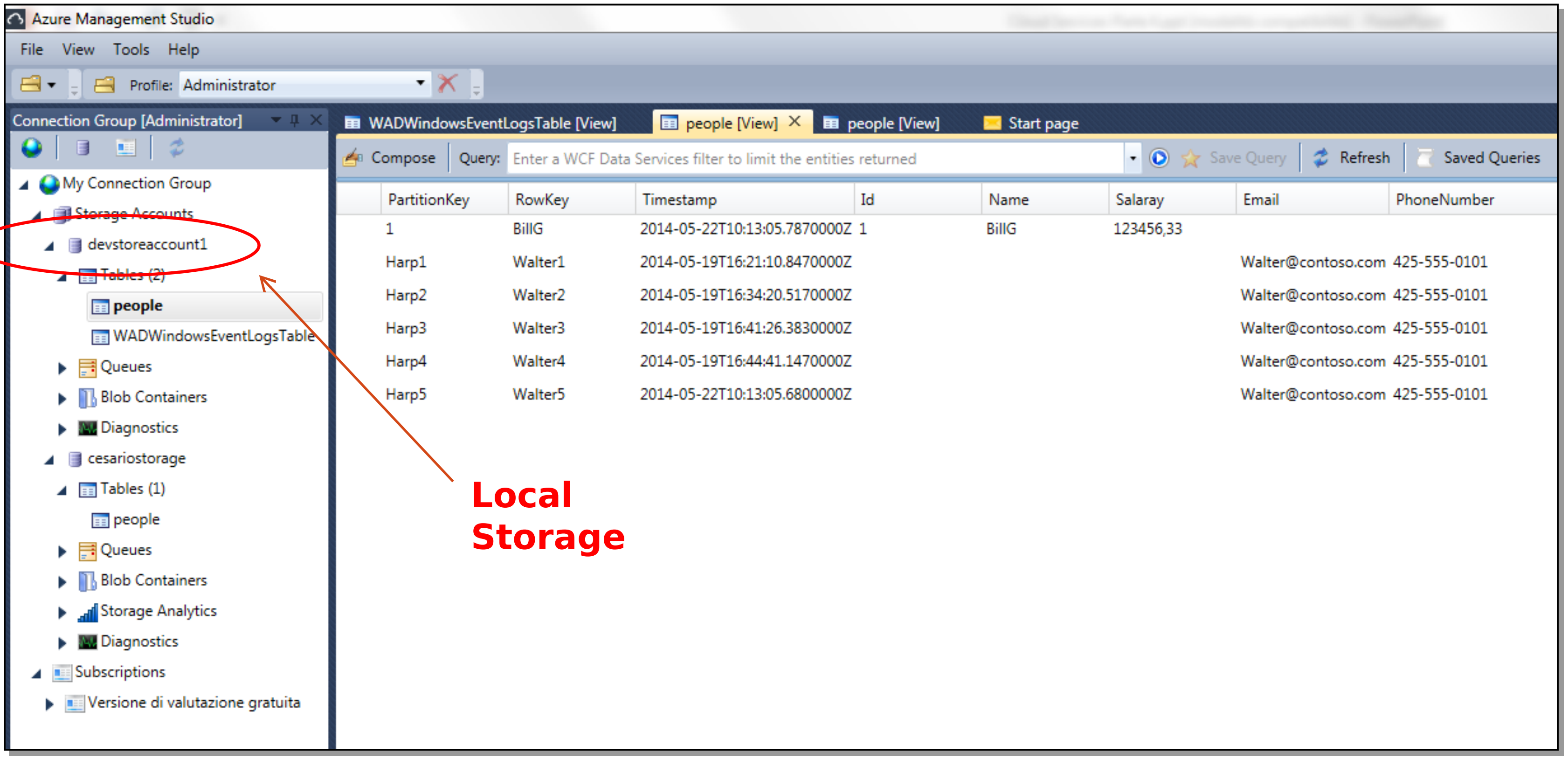
```
protected void btn_Click(object sender, EventArgs e){

        // Retrieve the storage account from the connection string.
        CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
           CloudConfigurationManager.GetSetting("StorageConnectionString"));

        // Create the table client.
        CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

        // Create the CloudTable object that represents the "people" table.
        CloudTable table = tableClient.GetTableReference("people");

        // Create the table if it doesn't exist.
        table.CreateIfNotExists();

        // Create a new customer entity.
        CustomerEntity customer = new CustomerEntity("Harp", "Walter");
        customer.Email = "Walter@contoso.com";
        customer.PhoneNumber = "425-555-0101";

        ...
```

# Code snippets – insert an entity

```
 protected void btn_Click(object sender, EventArgs e){

...

           // Create the TableOperation that inserts the customer entity.
           TableOperation insertCustomerOperation = TableOperation.Insert(customer);

           // Execute insert operations.
           table.Execute(insertCustomerOperation);

           EmployeeEntity employee = new EmployeeEntity(1, "BillG", 123456.33);
           TableOperation insertEmployeeOperation = TableOperation.Insert(employee);


           // Execute insert operations.
      table.Execute(insertEmployeeOperation);

 } //btn_Click
```

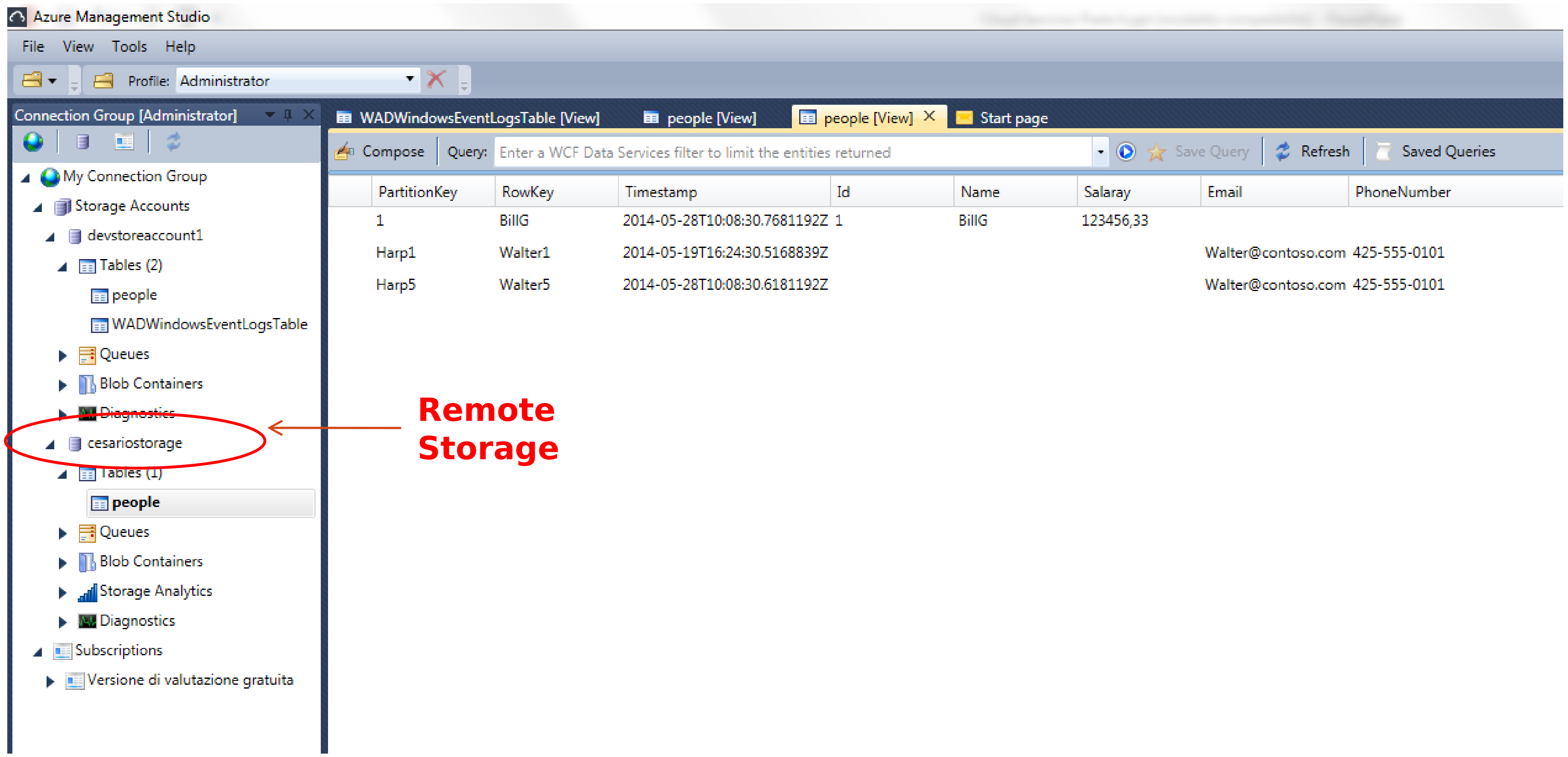
# View Content

- How to view table entries?
  - Azure Management Studio
  - Azure Storage Explorer



| PartitionKey | RowKey | Timestamp | Id | Name | Salaray | Email | PhoneNumber |
|---|---|---|---|---|---|---|---|
| 1 | BillG | 2014-05-22T10:13:05.7870000Z | 1 | BillG | 123456,33 | | |
| Harp1 | Walter1 | 2014-05-19T16:21:10.8470000Z | | | | Walter@contoso.com | 425-555-0101 |
| Harp2 | Walter2 | 2014-05-19T16:34:20.5170000Z | | | | Walter@contoso.com | 425-555-0101 |
| Harp3 | Walter3 | 2014-05-19T16:41:26.3830000Z | | | | Walter@contoso.com | 425-555-0101 |
| Harp4 | Walter4 | 2014-05-19T16:44:41.1470000Z | | | | Walter@contoso.com | 425-555-0101 |
| Harp5 | Walter5 | 2014-05-22T10:13:05.6800000Z | | | | Walter@contoso.com | 425-555-0101 |

# View Content



| PartitionKey | RowKey | Timestamp | Id | Name | Salaray | Email | PhoneNumber |
|---|---|---|---|---|---|---|---|
| 1 | BillG | 2014-05-28T10:08:30.7681192Z | 1 | BillG | 123456,33 | | |
| Harp1 | Walter1 | 2014-05-19T16:24:30.5168839Z | | | | Walter@contoso.com | 425-555-0101 |
| Harp5 | Walter5 | 2014-05-28T10:08:30.6181192Z | | | | Walter@contoso.com | 425-555-0101 |

## Demo

- Demo in classroom

- Local Storage
  - Fix connection string
  - Insert tuples
  - Check contents

- Remote Storage
  - Login the Azure Management Portal
  - Fix connection string
  - Insert tuples
  - Check contents

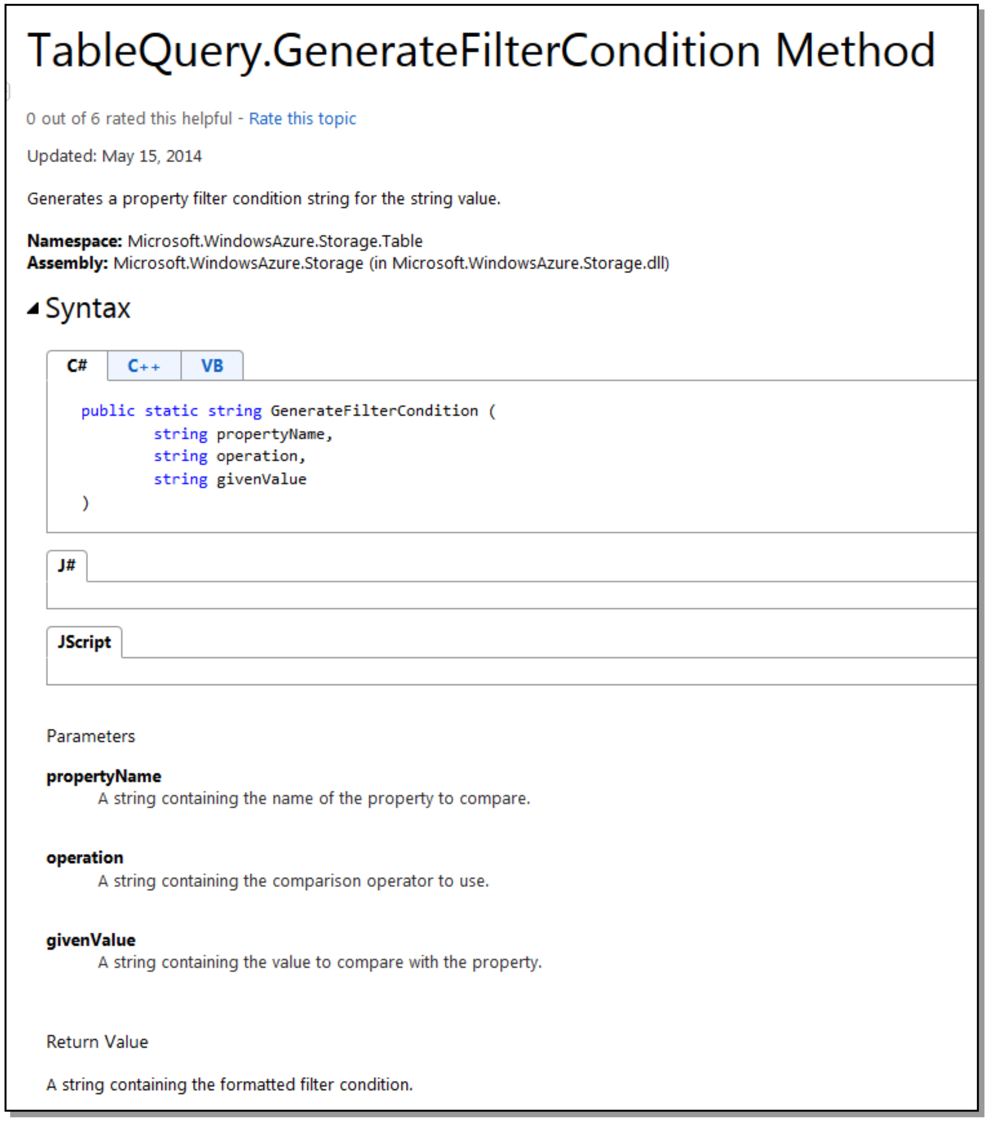
## Code snippets - Insert a batch of entities

```
// Retrieve the storage account from the connection string.
// Create the table client.
...
// Create the CloudTable object that represents the "people" table.
CloudTable table = tableClient.GetTableReference("people");

// Create the batch operation.
TableBatchOperation batchOperation = new TableBatchOperation();

// Create a customer entity and add it to the table.
CustomerEntity customer1 = new CustomerEntity("Smith", "Jeff");
customer1.Email = "Jeff@contoso.com";
customer1.PhoneNumber = "425-555-0104";

// Create another customer entity and add it to the table.
CustomerEntity customer2 = new CustomerEntity("Smith", "Ben");
customer2.Email = "Ben@contoso.com";
customer2.PhoneNumber = "425-555-0102";

// Add both customer entities to the batch insert operation.
batchOperation.Insert(customer1);
batchOperation.Insert(customer2);

// Execute the batch operation.
table.ExecuteBatch(batchOperation);
```

## Code snippets - Retrieve all entities in a partition

```
// Retrieve the storage account from the connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the table client.
CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

// Create the CloudTable object that represents the "people" table.
CloudTable table = tableClient.GetTableReference("people");

// Construct the query operation for all customer entities where PartitionKey="Smith".
TableQuery<CustomerEntity> query =
   new TableQuery<CustomerEntity>().Where(TableQuery.GenerateFilterCondition(
                    "PartitionKey", QueryComparisons.Equal, "Smith"));

// Print the fields for each customer.
foreach (CustomerEntity entity in table.ExecuteQuery(query))
{
    Console.WriteLine("{0}, {1}\t{2}\t{3}", entity.PartitionKey, entity.RowKey,
        entity.Email, entity.PhoneNumber);
}
```

# TableQuery.GenerateFilterCondition



## TableQuery.GenerateFilterCondition Method

0 out of 6 rated this helpful - Rate this topic

Updated: May 15, 2014

Generates a property filter condition string for the string value.

**Namespace:** Microsoft.WindowsAzure.Storage.Table
**Assembly:** Microsoft.WindowsAzure.Storage (in Microsoft.WindowsAzure.Storage.dll)

### Syntax

C# C++ VB

```
public static string GenerateFilterCondition (
        string propertyName,
        string operation,
        string givenValue
)
```

J#

JScript

Parameters

**propertyName**
A string containing the name of the property to compare.

**operation**
A string containing the comparison operator to use.

**givenValue**
A string containing the value to compare with the property.

Return Value

A string containing the formatted filter condition.

## Code snippets - Retrieve a range of entities in a partition

```
// Retrieve the storage account from the connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the table client.
CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

//Create the CloudTable object that represents the "people" table.
CloudTable table = tableClient.GetTableReference("people");

// Create the table query.
TableQuery<CustomerEntity> rangeQuery = new TableQuery<CustomerEntity>().Where(
    TableQuery.CombineFilters(
        TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal, "Smith"),
        TableOperators.And,
        TableQuery.GenerateFilterCondition("RowKey", QueryComparisons.LessThan, "E")));

// Loop through the results, displaying information about the entity.
foreach (CustomerEntity entity in table.ExecuteQuery(rangeQuery))
{
    Console.WriteLine("{0}, {1}\t{2}\t{3}", entity.PartitionKey, entity.RowKey,
        entity.Email, entity.PhoneNumber);
}
```

# Code snippets - Retrieve a single entity

```
// Retrieve the storage account from the connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the table client.
CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

// Create the CloudTable object that represents the "people" table.
CloudTable table = tableClient.GetTableReference("people");

// Create a retrieve operation that takes a customer entity.
TableOperation retrieveOperation = TableOperation.Retrieve<CustomerEntity>("Smith", "Ben");

// Execute the retrieve operation.
TableResult retrievedResult = table.Execute(retrieveOperation);

// Print the phone number of the result.
if (retrievedResult.Result != null)
   Console.WriteLine(((CustomerEntity)retrievedResult.Result).PhoneNumber);
else
   Console.WriteLine("The phone number could not be retrieved.");
```

# Code snippets - Replace an entity

```
// Retrieve the storage account from the connection string.
// Create the table client
// Create the CloudTable object that represents the "people" table.
...
// Create a retrieve operation that takes a customer entity.
TableOperation retrieveOperation = TableOperation.Retrieve<CustomerEntity>("Smith", "Ben");

// Execute the operation.
TableResult retrievedResult = table.Execute(retrieveOperation);

// Assign the result to a CustomerEntity object.
CustomerEntity updateEntity = (CustomerEntity)retrievedResult.Result;

if (updateEntity != null) {
   // Change the phone number.
   updateEntity.PhoneNumber = "425-555-0105";
   // Create the InsertOrReplace TableOperation
   TableOperation updateOperation = TableOperation.Replace(updateEntity);
   // Execute the operation.
   table.Execute(updateOperation);
   Console.WriteLine("Entity updated.");
}
else
   Console.WriteLine("Entity could not be retrieved.");
```

## Code snippets - Insert-or-replace an entity

```
// Retrieve the storage account from the connection string.
// Create the table client.
// Create the CloudTable object that represents the "people" table.
…

// Create a retrieve operation that takes a customer entity.
TableOperation retrieveOperation = TableOperation.Retrieve<CustomerEntity>("Smith", "Ben");

// Execute the operation.
TableResult retrievedResult = table.Execute(retrieveOperation);

// Assign the result to a CustomerEntity object.
CustomerEntity updateEntity = (CustomerEntity)retrievedResult.Result;

if (updateEntity != null) {
   // Change the phone number.
   updateEntity.PhoneNumber = "425-555-1234";
   // Create the InsertOrReplace TableOperation
   TableOperation insertOrReplaceOperation = TableOperation.InsertOrReplace(updateEntity);
   // Execute the operation.
   table.Execute(insertOrReplaceOperation);
   Console.WriteLine("Entity was updated.");
}
else
   Console.WriteLine("Entity could not be retrieved.");
```

# Code snippets - Delete an entity

```
// Retrieve storage account from connection string
// Create the table client
//Create the CloudTable that represents the "people" table.
...
// Create a retrieve operation that expects a customer entity.
TableOperation retrieveOperation = TableOperation.Retrieve<CustomerEntity>("Smith", "Ben");

// Execute the operation.
TableResult retrievedResult = table.Execute(retrieveOperation);

// Assign the result to a CustomerEntity.
CustomerEntity deleteEntity = (CustomerEntity)retrievedResult.Result;

// Create the Delete TableOperation.
if (deleteEntity != null) {
   TableOperation deleteOperation = TableOperation.Delete(deleteEntity);

   // Execute the operation.
   table.Execute(deleteOperation);

   Console.WriteLine("Entity deleted.");
}
else
   Console.WriteLine("Could not retrieve the entity.");
```

# Code snippets - Delete a table

```
// Retrieve the storage account from the connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the table client.
CloudTableClient tableClient = storageAccount.CreateCloudTableClient();

//Create the CloudTable that represents the "people" table.
CloudTable table = tableClient.GetTableReference("people");

// Delete the table it if exists.
table.DeleteIfExists();
```

## Summary

- Store data in tables
- Store data in blobs

Development Services on Windows Azure

## The Blob Service

- The Azure Blob storage service stores large amounts of unstructured data that can be accessed via HTTP or HTTPS.

- A single blob can be hundreds of gigabytes in size.

- Blob storage can be used for:
  - Storing files for distributed access
  - Streaming video and audio
  - Performing secure backup and disaster recovery
  - Serving images or documents directly to a browser

## Concepts

The Blob service contains the following components:

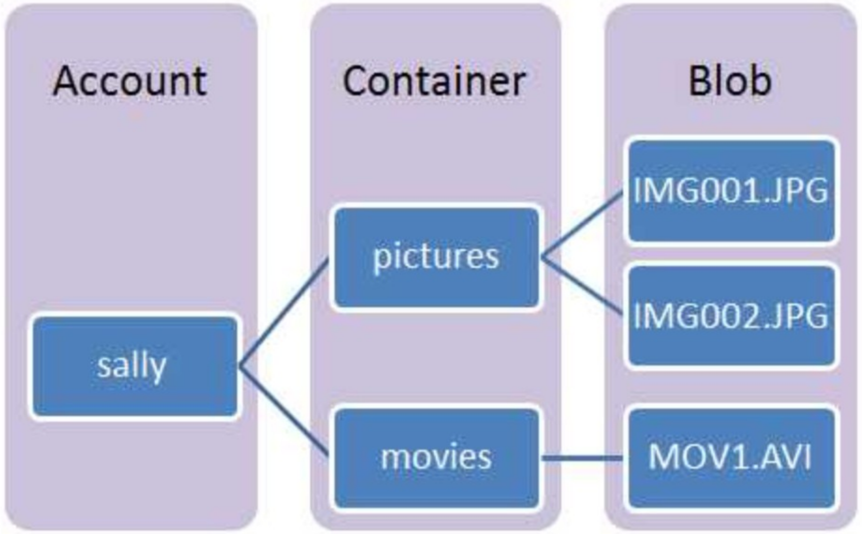
**Storage Account:** it is needed to access to any Azure Storage service

**Container:** it provides a grouping of a set of blobs

- All blobs must be in a container
- An account can contain an unlimited number of containers
- A container can store an unlimited number of blobs.

**Blob:** it is a file of any type and size

- two types of blobs: block and page blobs.
- A Block blob can be up to 200 GB in size.
- A Page blob, can be up to 1 TB in size
  - (more efficient when ranges of bytes in a file are modified frequently)
- A Directory blob is a directory



## Preliminary steps

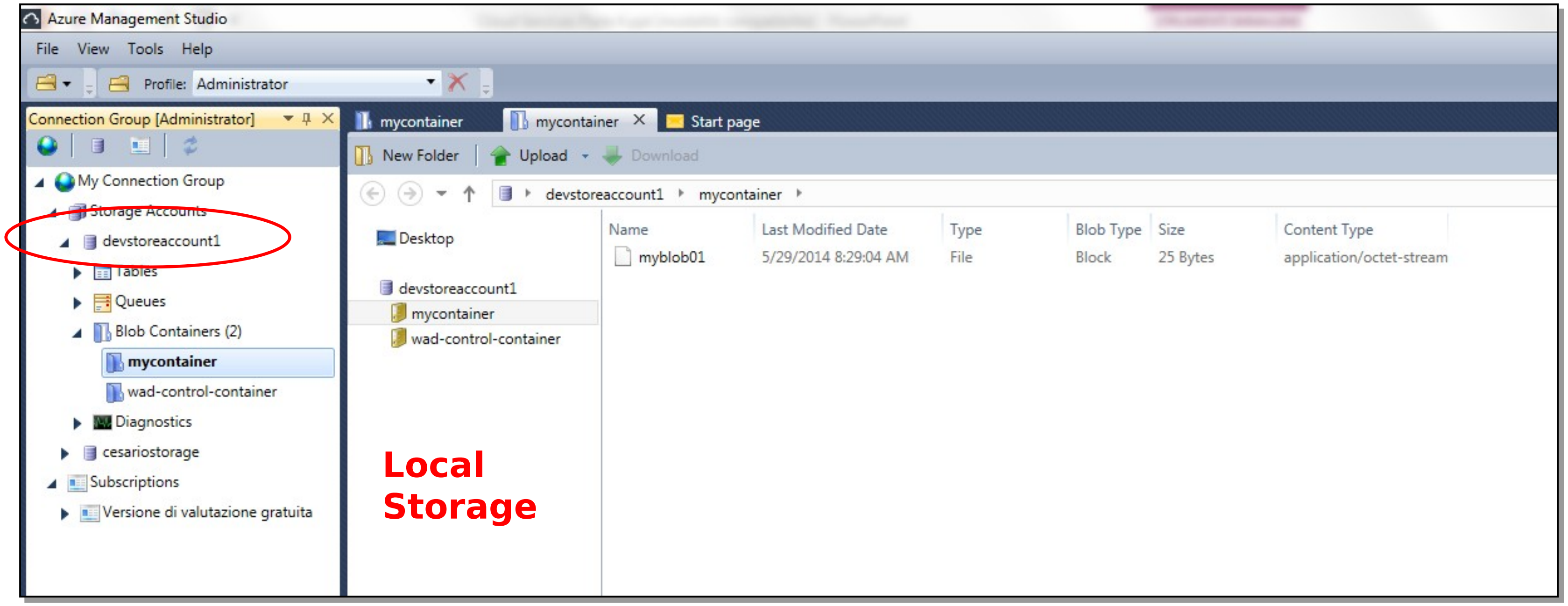
- Obtain the assembly (by NuGet)
- Create and configure a ConnectionString (i.e, ‘StorageConnectionString’)

# Code snippets - Upload a blob into a container

```
protected void btn_Click(object sender, EventArgs e) {

           // Retrieve storage account from connection string.
           CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
               CloudConfigurationManager.GetSetting("StorageConnectionString"));

           // Create the blob client.
           CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

           // Retrieve a reference to a container.
           CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

           // Create the container if it doesn't already exist.
           container.CreateIfNotExists();

           // Retrieve reference to a blob named "myblob".
           CloudBlockBlob blockBlob = container.GetBlockBlobReference("myblob01");

           // Create or overwrite the "myblob" blob with contents from a local file.
           using (var fileStream = System.IO.File.OpenRead(@"C:\...\BlobFiles\myblob01.txt")) {
                      blockBlob.UploadFromStream(fileStream);
           }

}
```

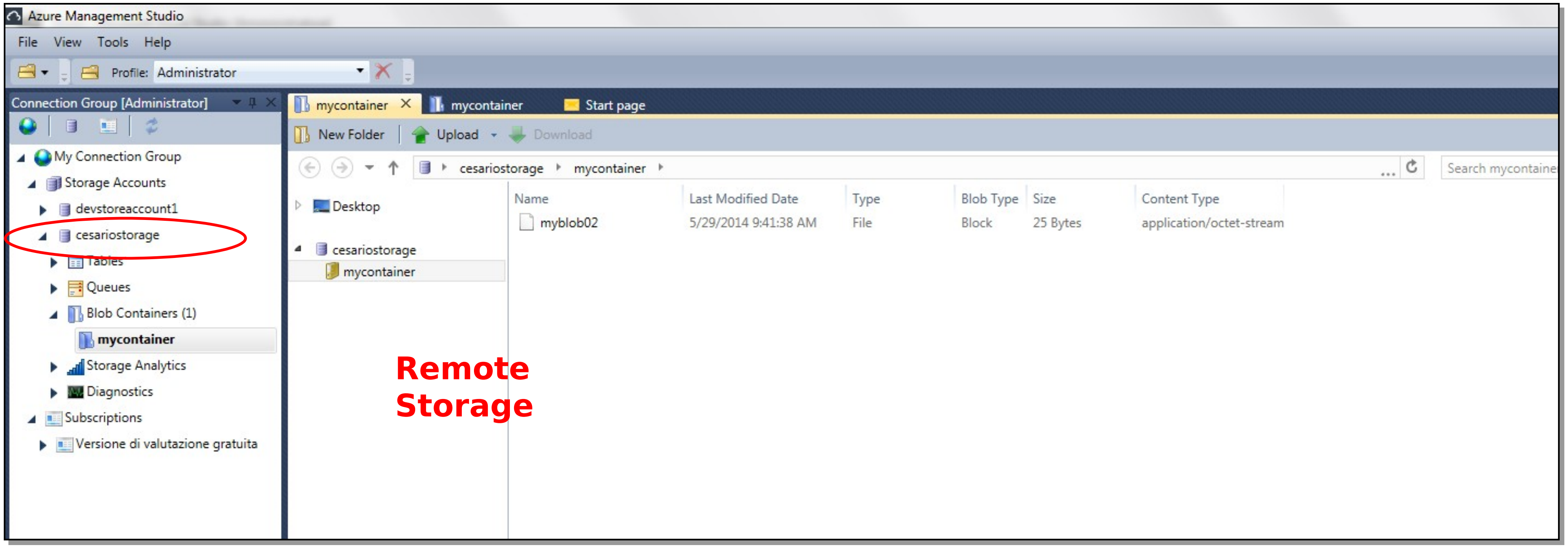
# View Blobs

- How to view blobs?
  - Azure Management Studio
  - Azure Storage Explorer



| Name | Last Modified Date | Type | Blob Type | Size | Content Type |
|---|---|---|---|---|---|
| myblob01 | 5/29/2014 8:29:04 AM | File | Block | 25 Bytes | application/octet-stream |

## View Blobs



| Name | Last Modified Date | Type | Blob Type | Size | Content Type |
|---|---|---|---|---|---|
| myblob02 | 5/29/2014 9:41:38 AM | File | Block | 25 Bytes | application/octet-stream |

## Demo

- Demo in classroom

- Local Storage
  - Fix connection string
  - Upload a blob
  - Check contents

- Remote Storage
  - Login the Azure Management Portal
  - Fix connection string
  - Upload a blob
  - Check contents

# Code snippets - List the blobs in a container (not flat listing)

```
 // Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("photos");

// Loop over items within the container and output the length and URI.
foreach (IListBlobItem item in container.ListBlobs(null, false)) {
    if (item.GetType() == typeof(CloudBlockBlob)) {
        CloudBlockBlob blob = (CloudBlockBlob)item;
        Console.WriteLine("Block blob of length {0}: {1}", blob.Properties.Length, blob.Uri);
    }
    else if (item.GetType() == typeof(CloudPageBlob)) {
        CloudPageBlob pageBlob = (CloudPageBlob)item;
        Console.WriteLine("Page blob of length {0}: {1}", pageBlob.Properties.Length,
pageBlob.Uri);
    }
    else if (item.GetType() == typeof(CloudBlobDirectory)) {
        CloudBlobDirectory directory = (CloudBlobDirectory)item;
        Console.WriteLine("Directory: {0}", directory.Uri);
    }
}
```

## Output: not flat listing

- Let us suppose the following set of block blobs in the container:
  - photo1.jpg
  - 2010/architecture/description.txt
  - 2010/architecture/photo3.jpg
  - 2010/architecture/photo4.jpg
  - 2011/architecture/photo5.jpg
  - 2011/architecture/photo6.jpg
  - 2011/architecture/description.txt
  - 2011/photo7.jpg

**Without using the «flat listing», CloudBlobDirectory and CloudBlockBlob objects are returned!**

- The resulting output could be:
  - Directory: https://<accountname>.blob.core.windows.net/photos/2010/
  - Directory: https://<accountname>.blob.core.windows.net/photos/2011/
  - Block blob of length 505623: https://<accountname>.blob.core.windows.net/photos/photo1.jpg

# Code snippets - List the blobs in a container (flat listing)

```
 // Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("photos");

// Loop over items within the container and output the length and URI.
foreach (IListBlobItem item in container.ListBlobs(null, true)) {
    if (item.GetType() == typeof(CloudBlockBlob)) {
        CloudBlockBlob blob = (CloudBlockBlob)item;
        Console.WriteLine("Block blob of length {0}: {1}", blob.Properties.Length, blob.Uri);
    }
    else if (item.GetType() == typeof(CloudPageBlob)) {
        CloudPageBlob pageBlob = (CloudPageBlob)item;
        Console.WriteLine("Page blob of length {0}: {1}", pageBlob.Properties.Length,
pageBlob.Uri);
    }
    else if (item.GetType() == typeof(CloudBlobDirectory)) {
        CloudBlobDirectory directory = (CloudBlobDirectory)item;
        Console.WriteLine("Directory: {0}", directory.Uri);
    }
}
```

## Output: flat listing

- Let us suppose the following set of block blobs in the container:
  - photo1.jpg
  - 2010/architecture/description.txt
  - 2010/architecture/photo3.jpg
  - 2010/architecture/photo4.jpg
  - 2011/architecture/photo5.jpg
  - 2011/architecture/photo6.jpg
  - 2011/architecture/description.txt
  - 2011/photo7.jpg

**Using the «flat listing», every blob in the container is returned as a CloudBlobBlock object!**

- The resulting output could be:
  - Block blob of length 4: https://<accountname>.blob.core.windows.net/photos/2010/architecture/description.txt
  - Block blob of length 314618: https://<accountname>.blob.core.windows.net/photos/2010/architecture/photo3.jpg
  - Block blob of length 522713: https://<accountname>.blob.core.windows.net/photos/2010/architecture/photo4.jpg
  - Block blob of length 4: https://<accountname>.blob.core.windows.net/photos/2011/architecture/description.txt
  - Block blob of length 419048: https://<accountname>.blob.core.windows.net/photos/2011/architecture/photo5.jpg
  - Block blob of length 506388: https://<accountname>.blob.core.windows.net/photos/2011/architecture/photo6.jpg
  - Block blob of length 399751: https://<accountname>.blob.core.windows.net/photos/2011/photo7.jpg
  - Block blob of length 505683

## Code snippets - Download blobs

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.
CloudBlobContainer container =
blobClient.GetContainerReference("mycontainer");

// Retrieve reference to a blob named "photo1.jpg".
CloudBlockBlob blockBlob =
container.GetBlockBlobReference("photo1.jpg");

// Save blob contents to a file.
using (var fileStream = System.IO.File.OpenWrite(@"path\myfile"))
{
    blockBlob.DownloadToStream(fileStream);
}
```

## Code snippets - Download blobs (as a text string)

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

// Retrieve reference to a blob named "myblob.txt"
CloudBlockBlob blockBlob2 = container.GetBlockBlobReference("myblob.txt");

string text;
using (var memoryStream = new MemoryStream())
{
    blockBlob2.DownloadToStream(memoryStream);
    text = System.Text.Encoding.UTF8.GetString(memoryStream.ToArray());
}
```

## Code snippets - Delete blobs

```
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve reference to a previously created container.
CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

// Retrieve reference to a blob named "myblob.txt".
CloudBlockBlob blockBlob = container.GetBlockBlobReference("myblob.txt");

// Delete the blob.
blockBlob.Delete();
```

## REST Web Services API example

- We want to create a simple application for storing contacts on a file database using JsonFile Database is a json file based database engine.

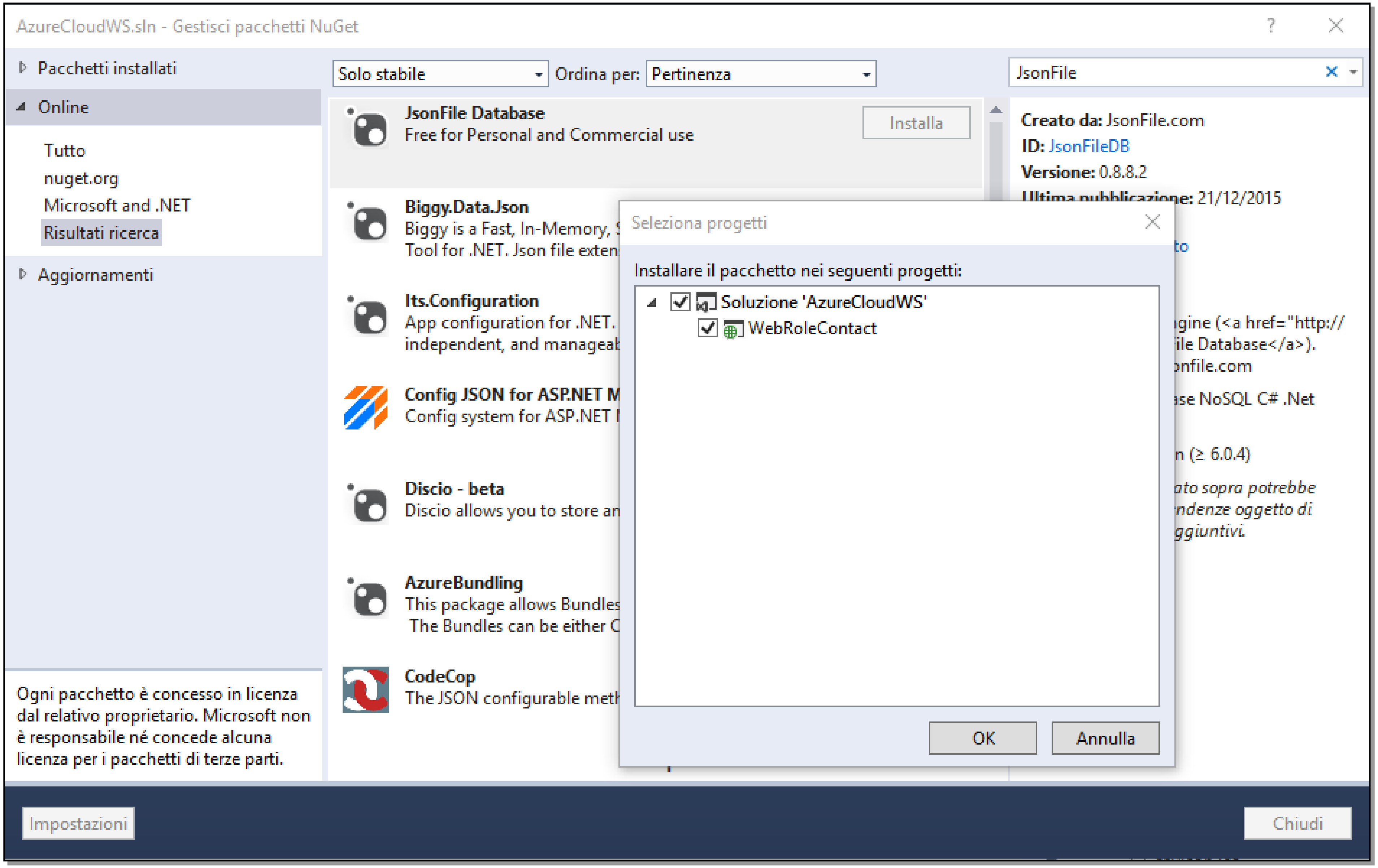
**What we need to start?**

- Create a new Microsoft Azure Cloud application using Visual Studio wizard

- Add just a WebRole (WebRoleContact) and choose Web API model with MVC template.

- Install packet **JsonFile Database (http://jsonfile.com/)** from NuGet and add reference to your solution

# Install JSONFile Database package



AzureCloudWS.sln - Gestisci pacchetti NuGet

Pacchetti installati
Online
Tutto
nuget.org
Microsoft and .NET
Risultati ricerca
Aggiornamenti

Solo stabile | Ordina per: Pertinenza | JsonFile

**JsonFile Database**
Free for Personal and Commercial use
Installa

**Biggy.Data.Json**
Biggy is a Fast, In-Memory, 
Tool for .NET. Json file exten

**Its.Configuration**
App configuration for .NET.
independent, and manageab

**Config JSON for ASP.NET M**
Config system for ASP.NET

**Discio - beta**
Discio allows you to store an

**AzureBundling**
This package allows Bundles
The Bundles can be either C

**CodeCop**
The JSON configurable meth

**Creato da:** JsonFile.com
**ID:** JsonFileDB
**Versione:** 0.8.8.2
**Ultima pubblicazione:** 21/12/2015

gine (<a href="http://
ile Database</a>).
nfile.com
se NoSQL C# .Net
n (≥ 6.0.4)
*ato sopra potrebbe*
*ndenze oggetto di*
*ggiuntivi.*

Seleziona progetti

Installare il pacchetto nei seguenti progetti:
- ☑ Soluzione 'AzureCloudWS'
  - ☑ WebRoleContact

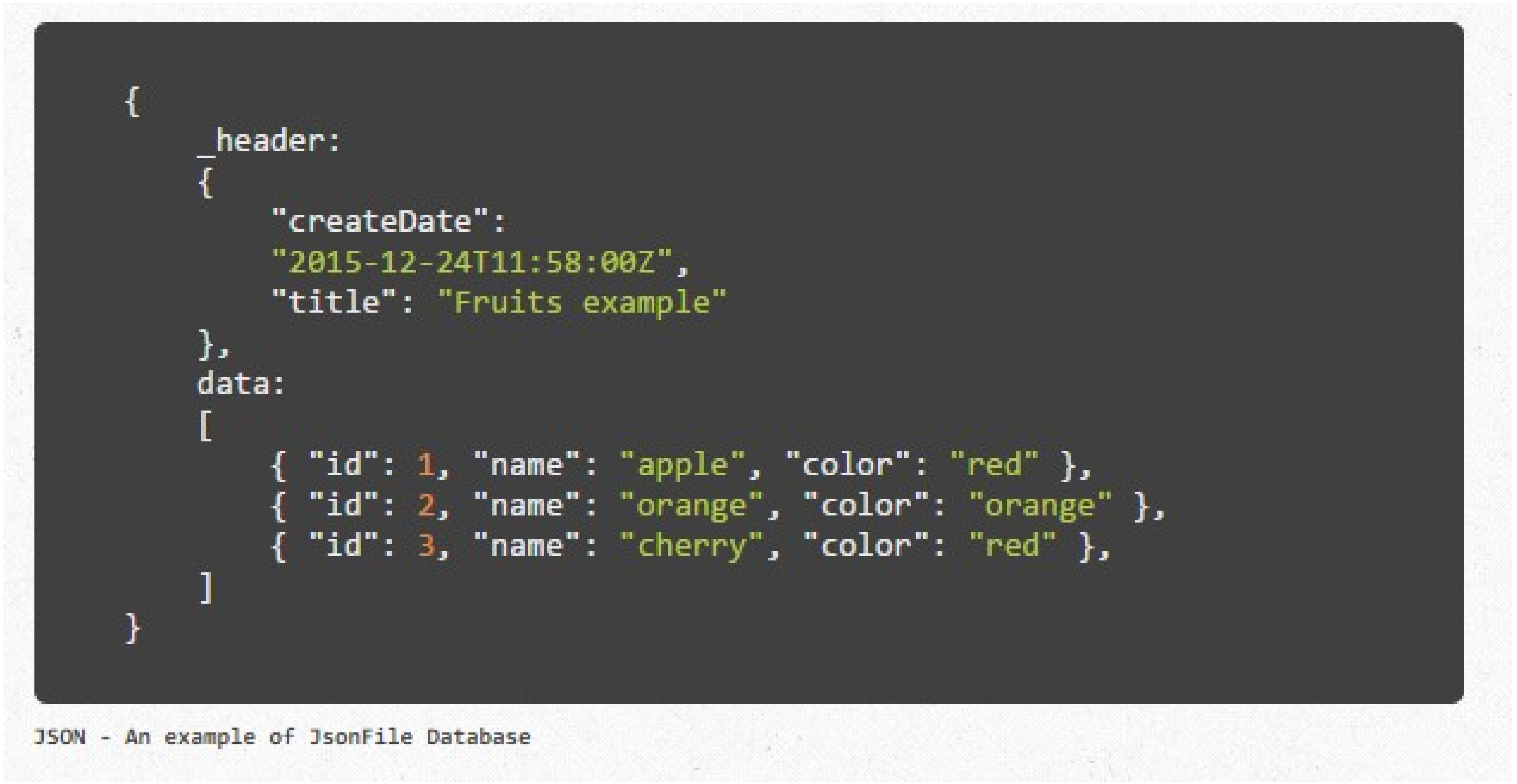
OK | Annulla

Ogni pacchetto è concesso in licenza dal relativo proprietario. Microsoft non è responsabile né concede alcuna licenza per i pacchetti di terze parti.

Impostazioni | Chiudi

## JSON Database package

- This package allows to create a database  to store data using JSON format
- It is easy to use
- Ideal to create very simple application.



```
{
    _header:
    {
        "createDate":
        "2015-12-24T11:58:00Z",
        "title": "Fruits example"
    },
    data:
    [
        { "id": 1, "name": "apple", "color": "red" },
        { "id": 2, "name": "orange", "color": "orange" },
        { "id": 3, "name": "cherry", "color": "red" },
    ]
}
```

JSON - An example of JsonFile Database

# Create database table structure

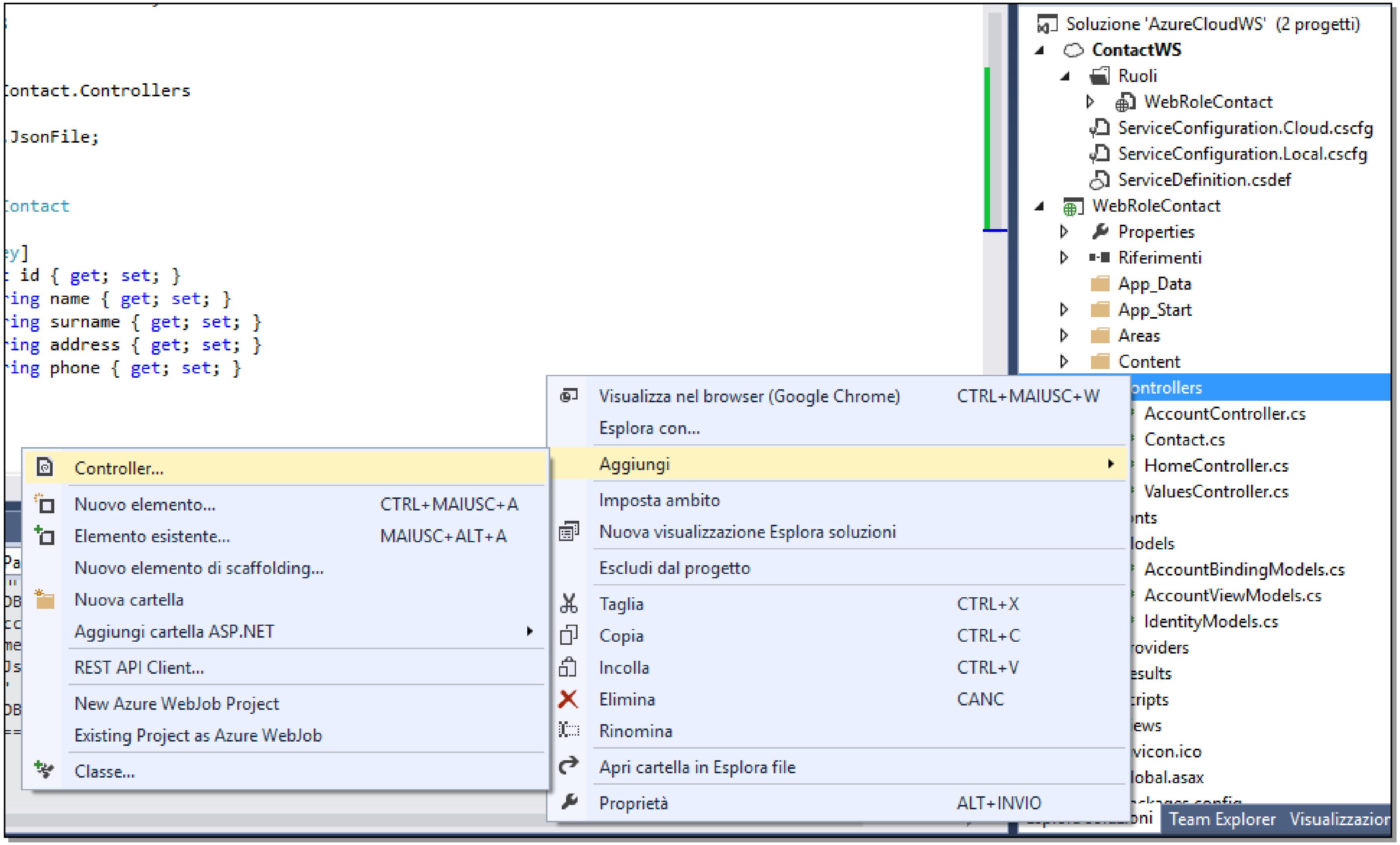
```
public class Contact
    {
        [PrimaryKey]
        public int id { get; set; }

    public string name { get; set; }
        public string surname { get; set; }
        public string address { get; set; }
        public string phone { get; set; }

    }
```

Defines primary key for the table

Other fields for the contact's table

# Add a controller for the Contact class

- Create a class ContactController



```
ontact.Controllers

.JsonFile;

ontact

y]
 id { get; set; }
ing name { get; set; }
ing surname { get; set; }
ing address { get; set; }
ing phone { get; set; }
```

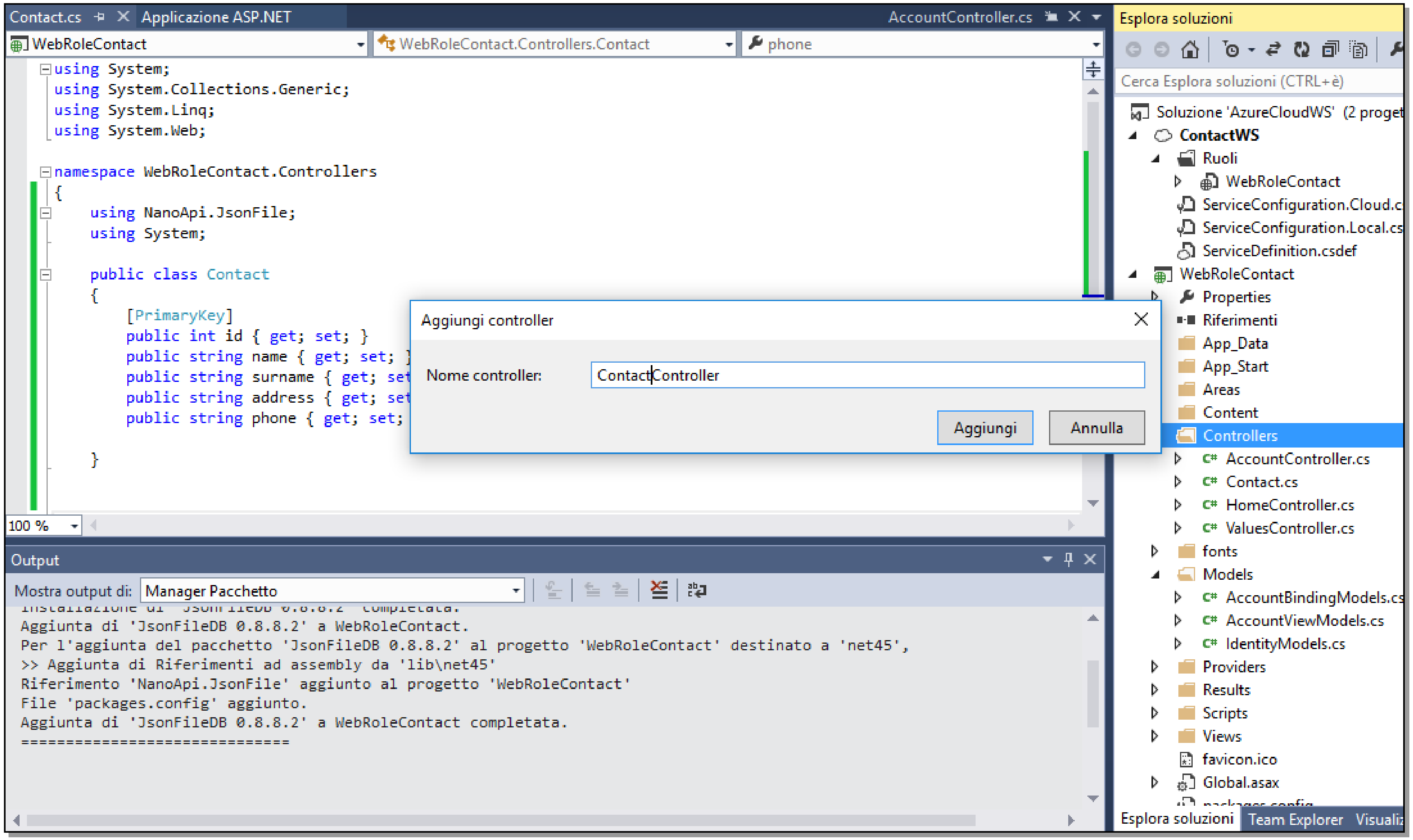
Soluzione 'AzureCloudWS' (2 progetti)
- ContactWS
  - Ruoli
    - WebRoleContact
  - ServiceConfiguration.Cloud.cscfg
  - ServiceConfiguration.Local.cscfg
  - ServiceDefinition.csdef
- WebRoleContact
  - Properties
  - Riferimenti
  - App_Data
  - App_Start
  - Areas
  - Content
  - ontrollers
    - AccountController.cs
    - Contact.cs
    - HomeController.cs
    - ValuesController.cs
  - nts
  - lodels
    - AccountBindingModels.cs
    - AccountViewModels.cs
    - IdentityModels.cs
  - roviders
  - esults
  - ripts
  - ews
  - vicon.ico
  - lobal.asax

| | |
|---|---|
| Visualizza nel browser (Google Chrome) | CTRL+MAIUSC+W |
| Esplora con... | |
| Aggiungi | |
| Imposta ambito | |
| Nuova visualizzazione Esplora soluzioni | |
| Escludi dal progetto | |
| Taglia | CTRL+X |
| Copia | CTRL+C |
| Incolla | CTRL+V |
| Elimina | CANC |
| Rinomina | |
| Apri cartella in Esplora file | |
| Proprietà | ALT+INVIO |

| | |
|---|---|
| Controller... | |
| Nuovo elemento... | CTRL+MAIUSC+A |
| Elemento esistente... | MAIUSC+ALT+A |
| Nuovo elemento di scaffolding... | |
| Nuova cartella | |
| Aggiungi cartella ASP.NET | |
| REST API Client... | |
| New Azure WebJob Project | |
| Existing Project as Azure WebJob | |
| Classe... | |

# Add a controller for the Contact class

- Create a class ContactController choosing Web API 2 model with write/read operations



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace WebRoleContact.Controllers
{
    using NanoApi.JsonFile;
    using System;

    public class Contact
    {
        [PrimaryKey]
        public int id { get; set; }
        public string name { get; set; }
        public string surname { get; set
        public string address { get; set
        public string phone { get; set;

    }
```

Aggiungi controller

Nome controller: ContactController

Aggiungi | Annulla

Mostra output di: Manager Pacchetto

```
Installazione di 'JsonFileDB 0.8.8.2' completata.
Aggiunta di 'JsonFileDB 0.8.8.2' a WebRoleContact.
Per l'aggiunta del pacchetto 'JsonFileDB 0.8.8.2' al progetto 'WebRoleContact' destinato a 'net45',
>> Aggiunta di Riferimenti ad assembly da 'lib\net45'
Riferimento 'NanoApi.JsonFile' aggiunto al progetto 'WebRoleContact'
File 'packages.config' aggiunto.
Aggiunta di 'JsonFileDB 0.8.8.2' a WebRoleContact completata.
==============================
```

# Add a controller for the Contact class

```
public class ContactController : ApiController
    {
        // GET: api/Contact
        public IEnumerable<string> Get() {
            return new string[] { "value1",
"value2" };
        }

        // GET: api/Contact/5
        public string Get(int id) {
            return "value";
        }

        // POST: api/Contact
        public void Post([FromBody]string value){}

        // PUT: api/Contact/5
        public void Put(int id, [FromBody]string
value)
        {}

        // DELETE: api/Contact/5
        public void Delete(int id) {}
    }
```

- A base template for the controller has been created
- All the basic operation (GET, POST, PUT, DELETE) are created and ready to be implemented

## Complete and customize the ContactController class

```
[RoutePrefix("api/Contact")] //  Used to define a base api path

public class ContactController : ApiController {

    // GET: api/Contact
    public IEnumerable<Contact> Get()
    {
        return Contact.list().ToArray();
    }
    // GET: api/Contact/5
    public Contact Get(int id)
    {
        return Contact.searchById(id);
    }
   // DELETE: api/Contact/5
    public void Delete(int id)
    {
        Contact.delete(id);
    }
   …
```

# Complete and customize the ContactController class

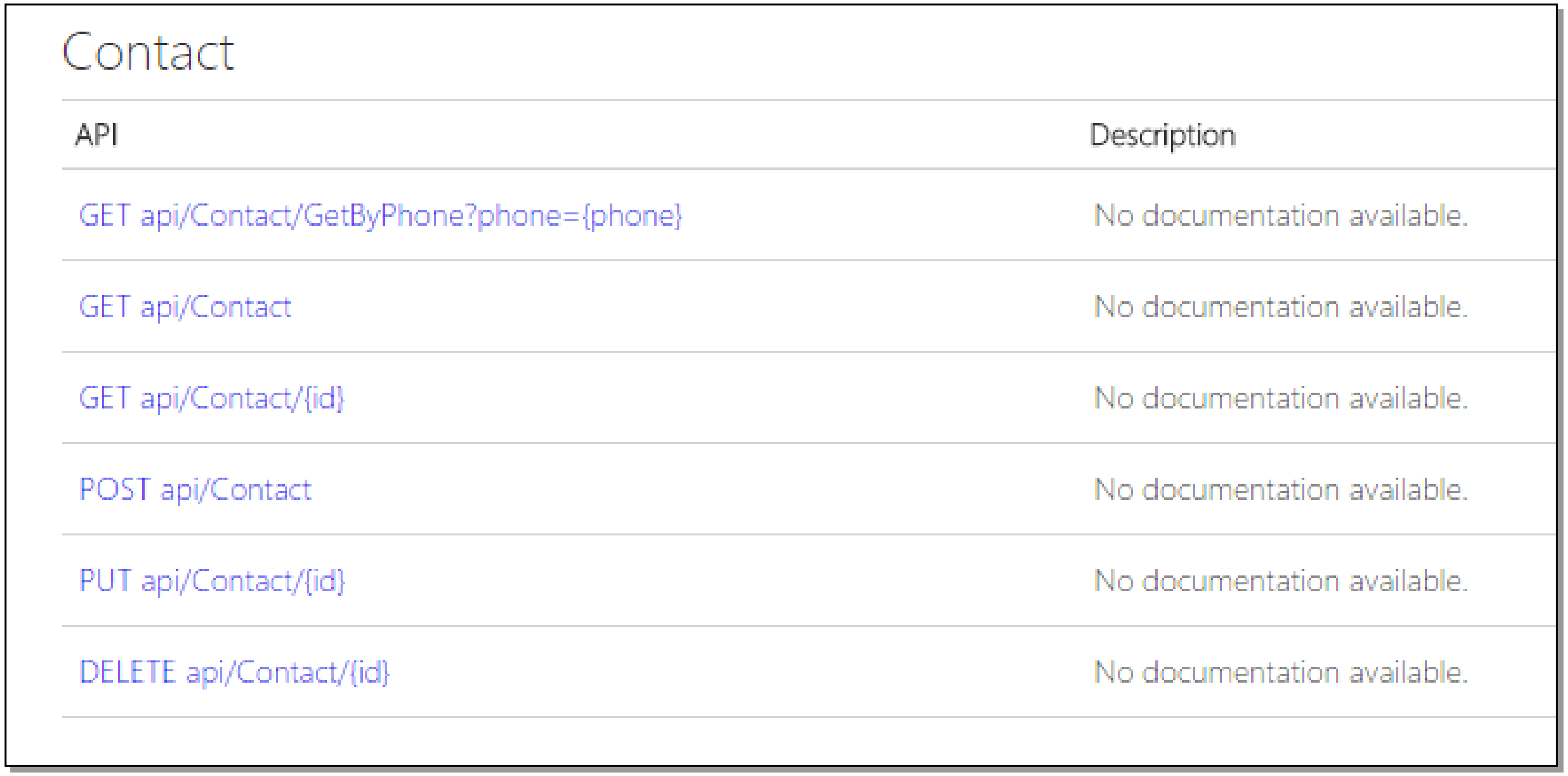
```
        …
    // POST: api/Contact <-- Used to add
        public void Post() {
            NameValueCollection postData = getQueryStringParams();
            String name = postData.Get("name");
            String surname = postData.Get("surname");
            String address = postData.Get("address");
            String phone = postData.Get("phone");
            Contact.create(name, surname, address,phone);
        }
        // PUT: api/Contact/5 <-- Used to update
        public void Put(int id) {
            NameValueCollection postData = getQueryStringParams();
            String name = postData.Get("name");
            String surname = postData.Get("surname");
            String address = postData.Get("address");
            String phone = postData.Get("phone");
            Contact.update(id, name, surname, address, phone);
        }
…
```

## Complete and customize the ContactController class

```
    …

    //Used to parse parameters from query string
    private NameValueCollection getQueryStringParams() {
        HttpContent requestContent = Request.Content;
        string queryString = requestContent.ReadAsStringAsync().Result;
        return HttpUtility.ParseQueryString(queryString);
    }

    [Route("GetByPhone")]       // □ Used to customize api path
    public IEnumerable<Contact> GetByPhone(string phone)
    {
        return Contact.searchByPhone(phone);
    }
}
```

# Add a basic controller for the Contact class

- After launching the application, the Contact APIs are automatically listed
- For each API method a detailed description is also generated

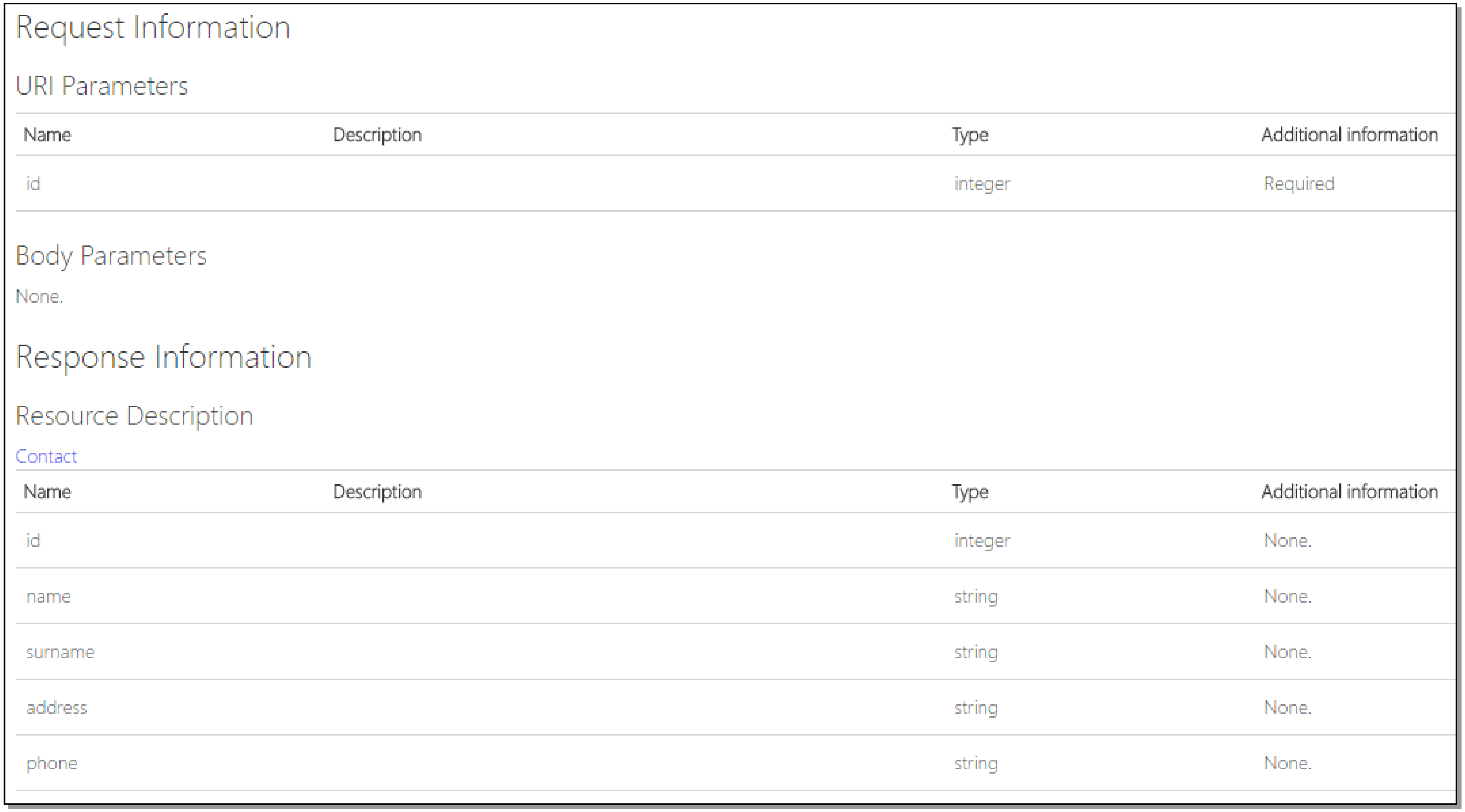


Contact

| API | Description |
|---|---|
| GET api/Contact/GetByPhone?phone={phone} | No documentation available. |
| GET api/Contact | No documentation available. |
| GET api/Contact/{id} | No documentation available. |
| POST api/Contact | No documentation available. |
| PUT api/Contact/{id} | No documentation available. |
| DELETE api/Contact/{id} | No documentation available. |

# Add a basic controller for the Contact class

- For each API method a detailed description is also generated



## Request Information

### URI Parameters

| Name | Description | Type | Additional information |
|---|---|---|---|
| id | | integer | Required |

### Body Parameters

None.

## Response Information

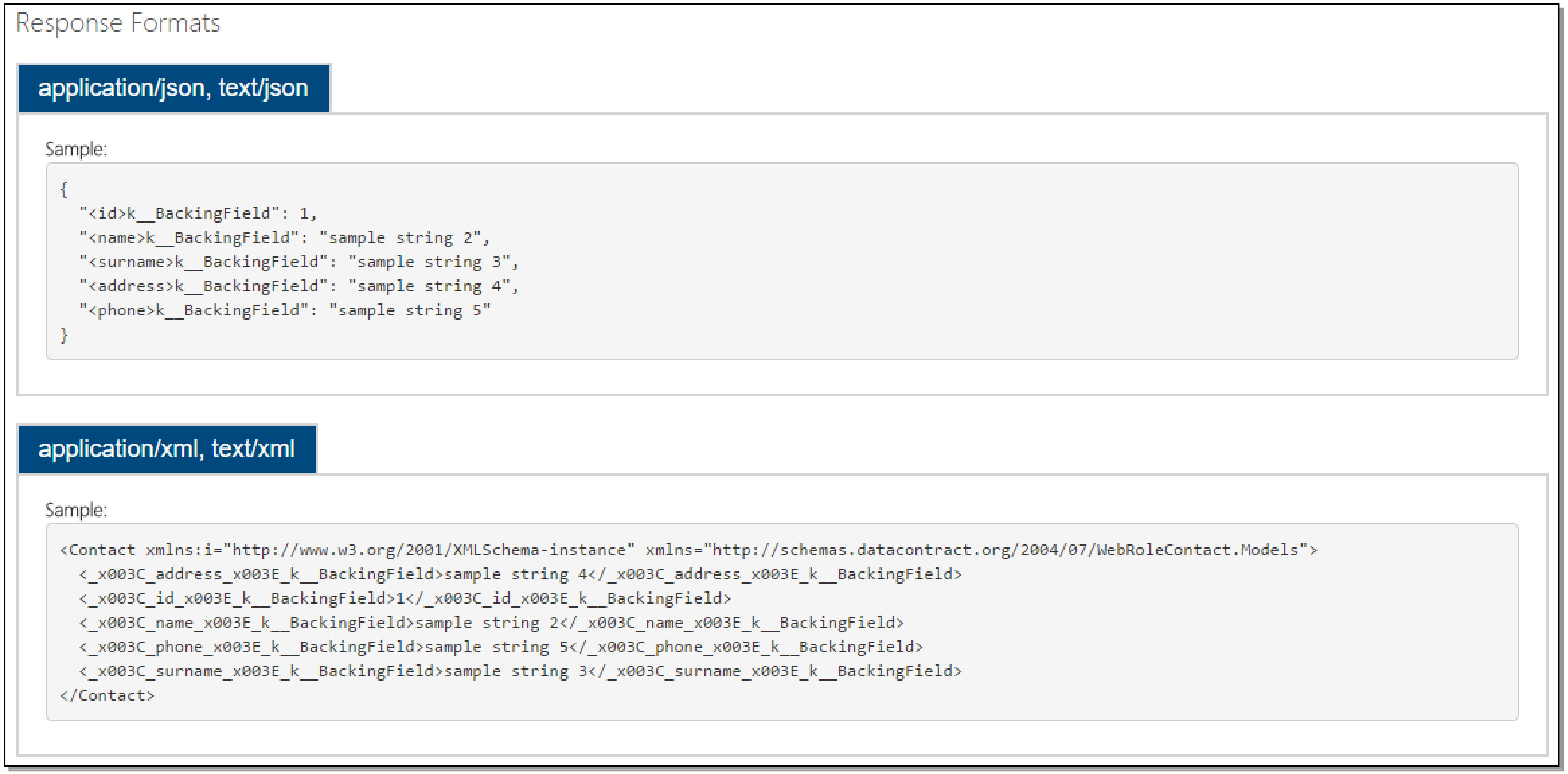
### Resource Description

Contact

| Name | Description | Type | Additional information |
|---|---|---|---|
| id | | integer | None. |
| name | | string | None. |
| surname | | string | None. |
| address | | string | None. |
| phone | | string | None. |

# Contact API – Example of return formats

- Return formats are JSON e XML



Response Formats

application/json, text/json

Sample:

```
{
  "<id>k__BackingField": 1,
  "<name>k__BackingField": "sample string 2",
  "<surname>k__BackingField": "sample string 3",
  "<address>k__BackingField": "sample string 4",
  "<phone>k__BackingField": "sample string 5"
}
```

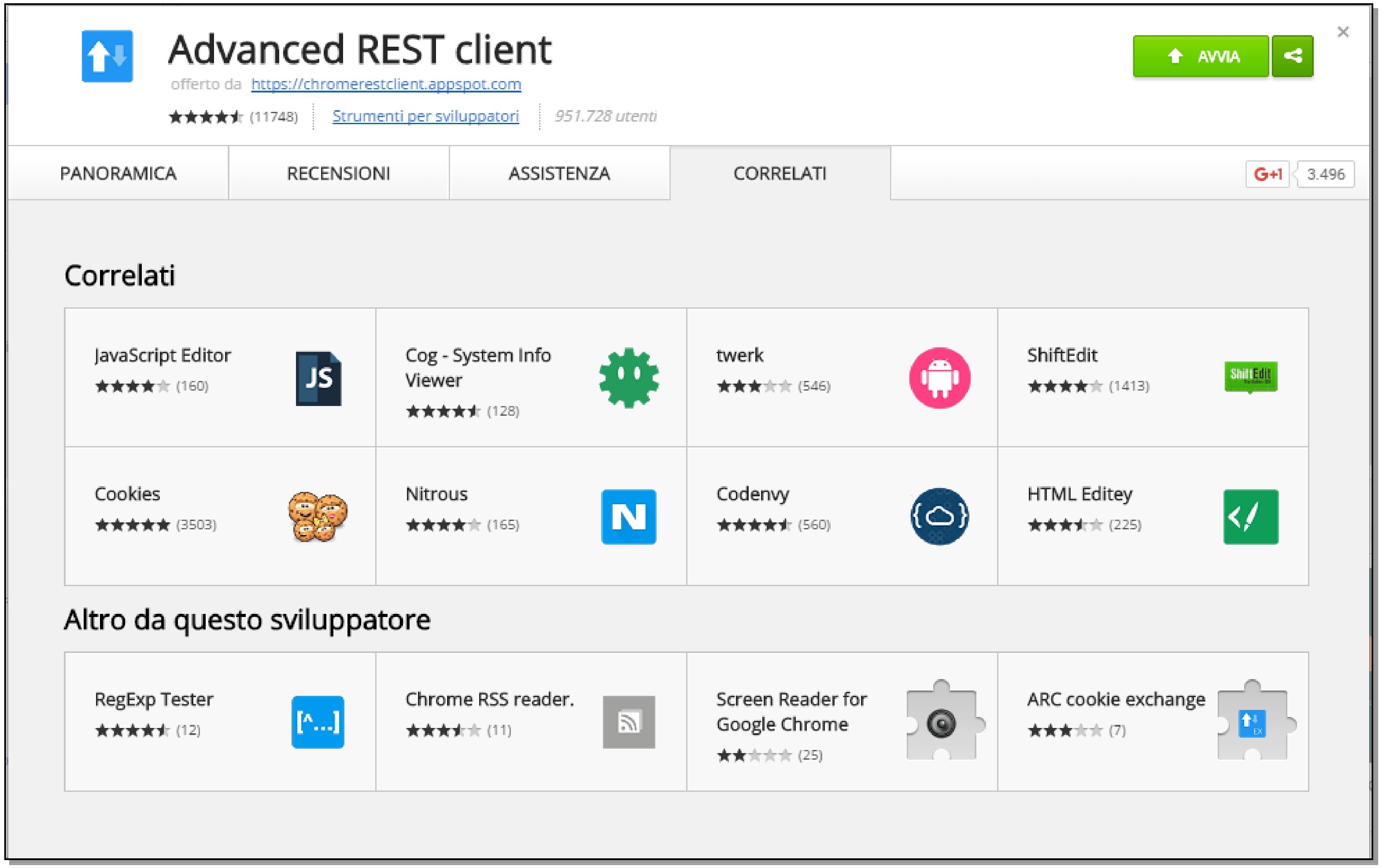
application/xml, text/xml

Sample:

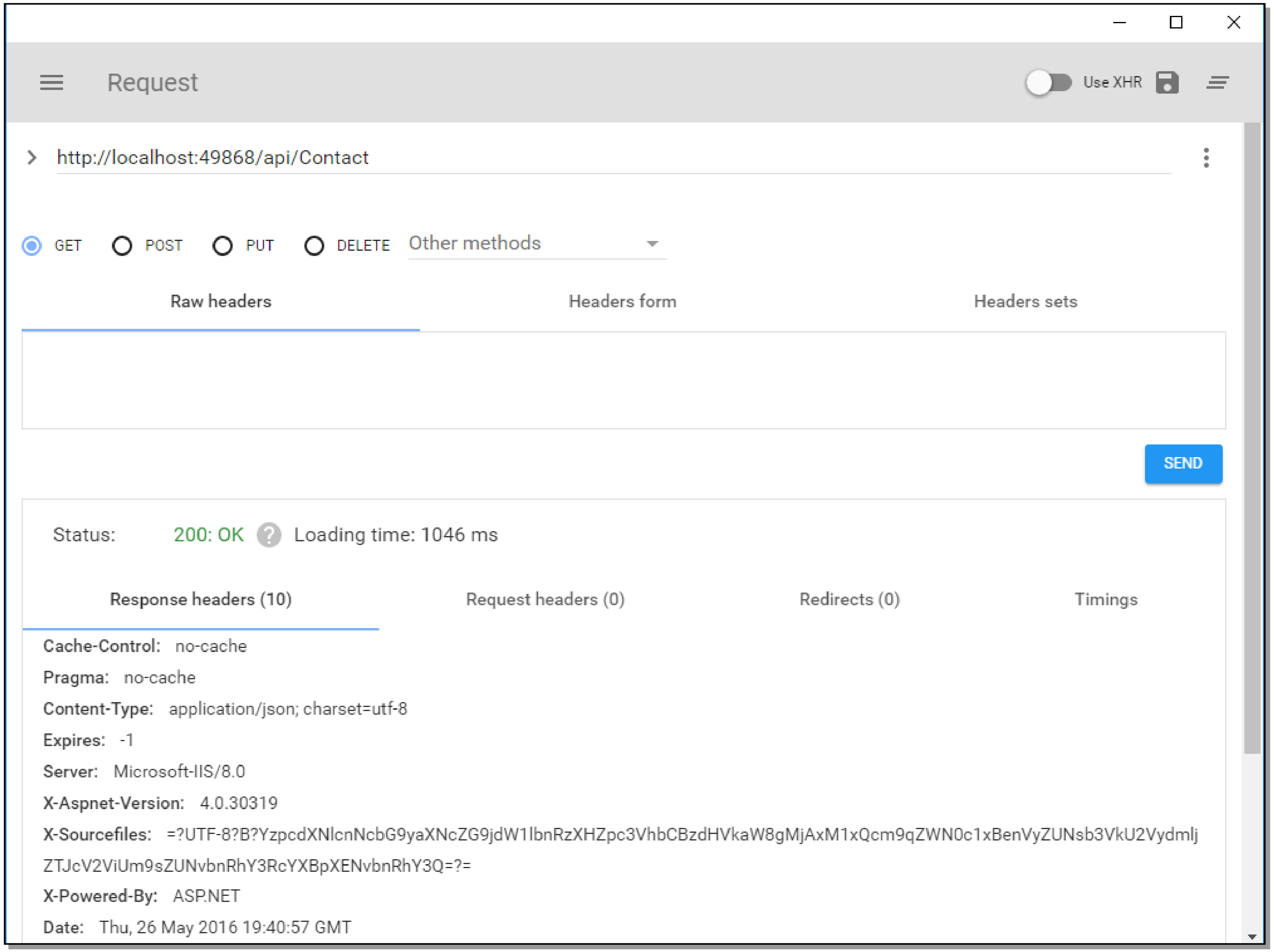
```
<Contact xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/WebRoleContact.Models">
  <_x003C_address_x003E_k__BackingField>sample string 4</_x003C_address_x003E_k__BackingField>
  <_x003C_id_x003E_k__BackingField>1</_x003C_id_x003E_k__BackingField>
  <_x003C_name_x003E_k__BackingField>sample string 2</_x003C_name_x003E_k__BackingField>
  <_x003C_phone_x003E_k__BackingField>sample string 5</_x003C_phone_x003E_k__BackingField>
  <_x003C_surname_x003E_k__BackingField>sample string 3</_x003C_surname_x003E_k__BackingField>
</Contact>
```

## How to make API calls

- To test our application we can use a REST client to make API calls


Advanced REST client
offerto da https://chromerestclient.appspot.com
★★★★★ (11748) Strumenti per sviluppatori 951.728 utenti
AVVIA
PANORAMICA
RECENSIONI
ASSISTENZA
CORRELATI
G+1 3.496
Correlati
JavaScript Editor ★★★★☆ (160)
Cog - System Info Viewer ★★★★★ (128)
twerk ★★★☆☆ (546)
ShiftEdit ★★★★☆ (1413)
Cookies ★★★★★ (3503)
Nitrous ★★★★☆ (165)
Codenvy ★★★★★ (560)
HTML Editey ★★★★☆ (225)
Altro da questo sviluppatore
RegExp Tester ★★★★★ (12)
Chrome RSS reader. ★★★★☆ (11)
Screen Reader for Google Chrome ★★☆☆☆ (25)
ARC cookie exchange ★★★☆☆ (7)

# Advanced REST client



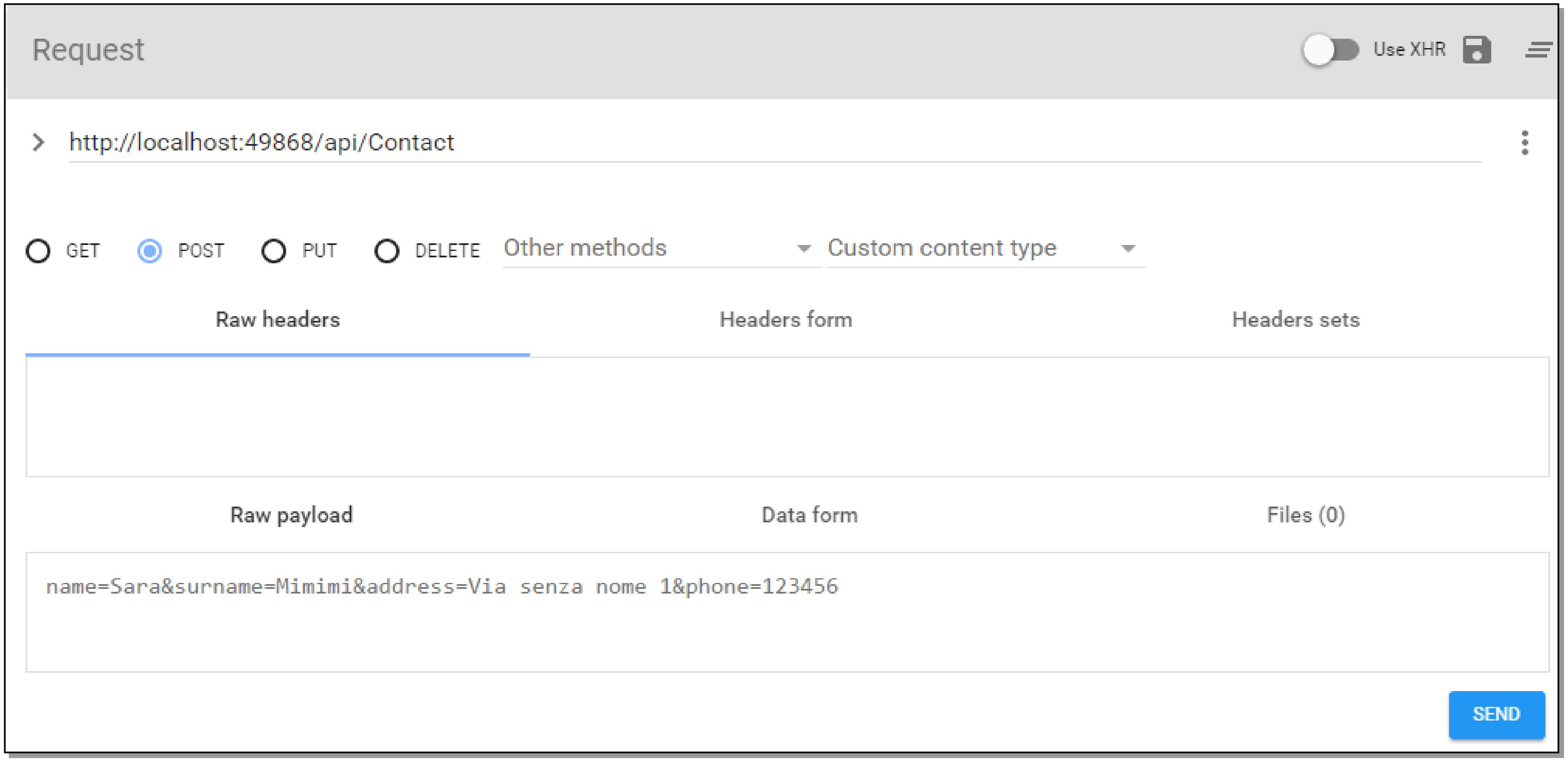
http://localhost:49868/api/Contact

[✓ GET] POST PUT DELETE

Status: 200: OK Loading time: 1046 ms

```
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-Aspnet-Version: 4.0.30319
X-Sourcefiles: =?UTF-8?B?YzpcdXNlcnNcbG9yaXNcZG9jdW1lbnRzXHZpc3VhbCBzdHVkaW8gMjAxM1xQcm9qZWN0c1xBenVyZUNsb3VkU2Vydmlj
ZTJcV2ViUm9sZUNvbnRhY3RcYXBpXENvbnRhY3Q=?=
X-Powered-By: ASP.NET
Date: Thu, 26 May 2016 19:40:57 GMT
```

# How to add a new contact trough API



Request

Use XHR

http://localhost:49868/api/Contact

GET [✓ POST] PUT DELETE Other methods Custom content type

Raw headers | Headers form | Headers sets

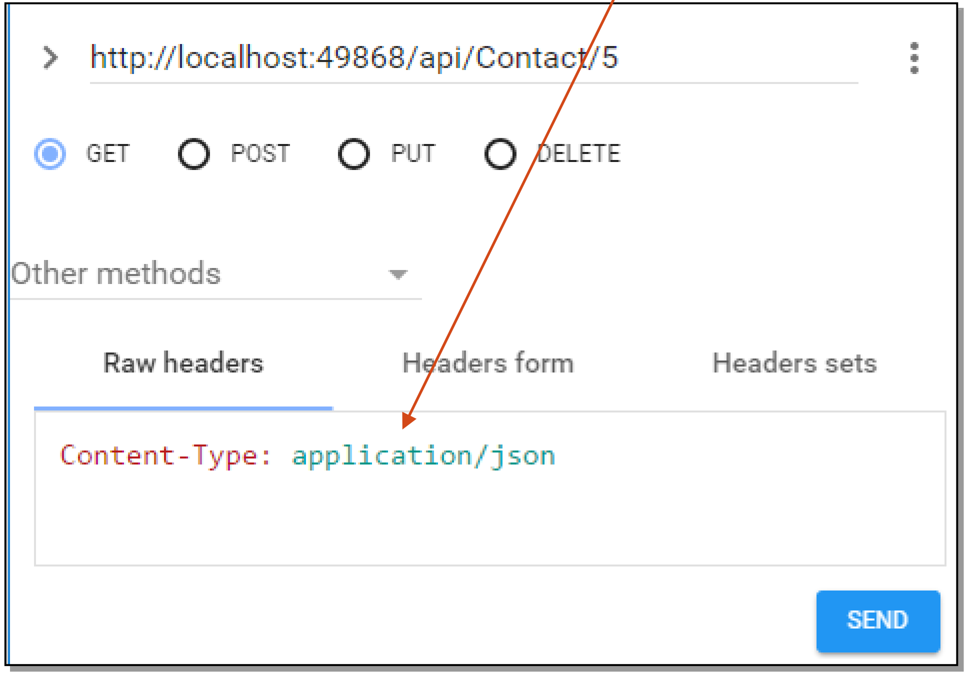
Raw payload | Data form | Files (0)

```
name=Sara&surname=Mimimi&address=Via senza nome 1&phone=123456
```

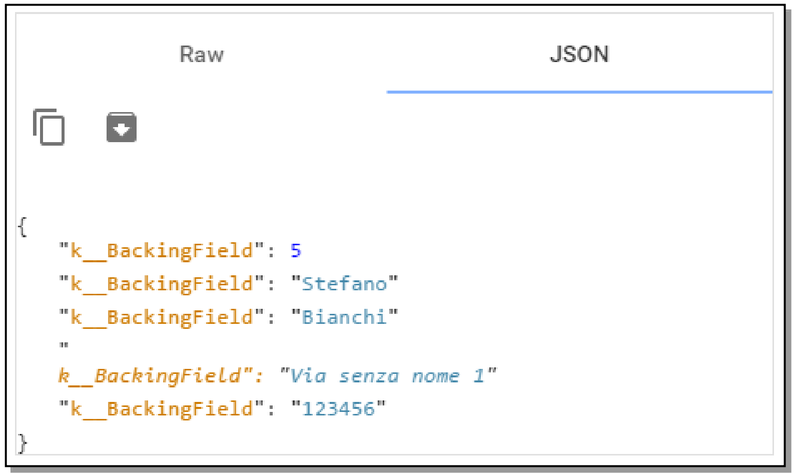
SEND

# How to add a get contact details trough API

The response message must be in json format

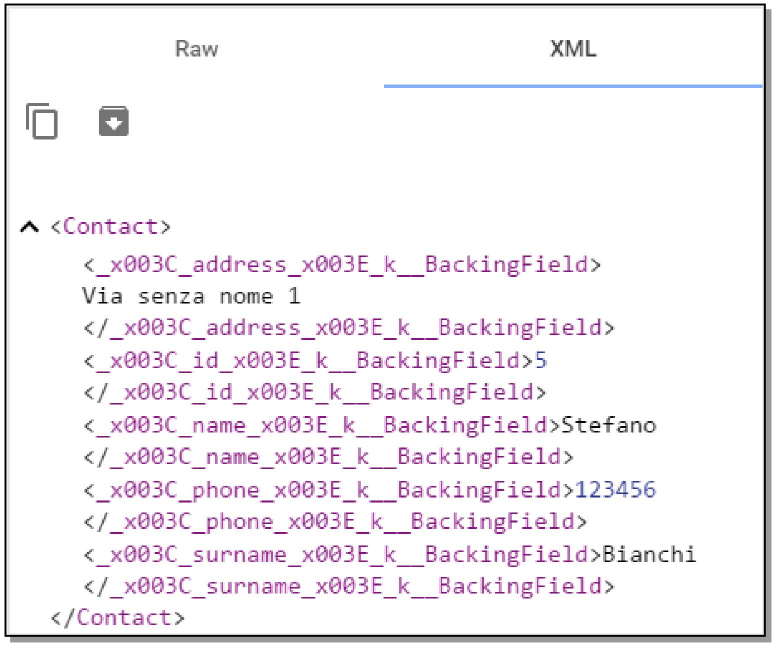


```
Content-Type: application/json
```



```
{
   "k__BackingField": 5
   "k__BackingField": "Stefano"
   "k__BackingField": "Bianchi"
   "
   k__BackingField": "Via senza nome 1"
   "k__BackingField": "123456"
}
```

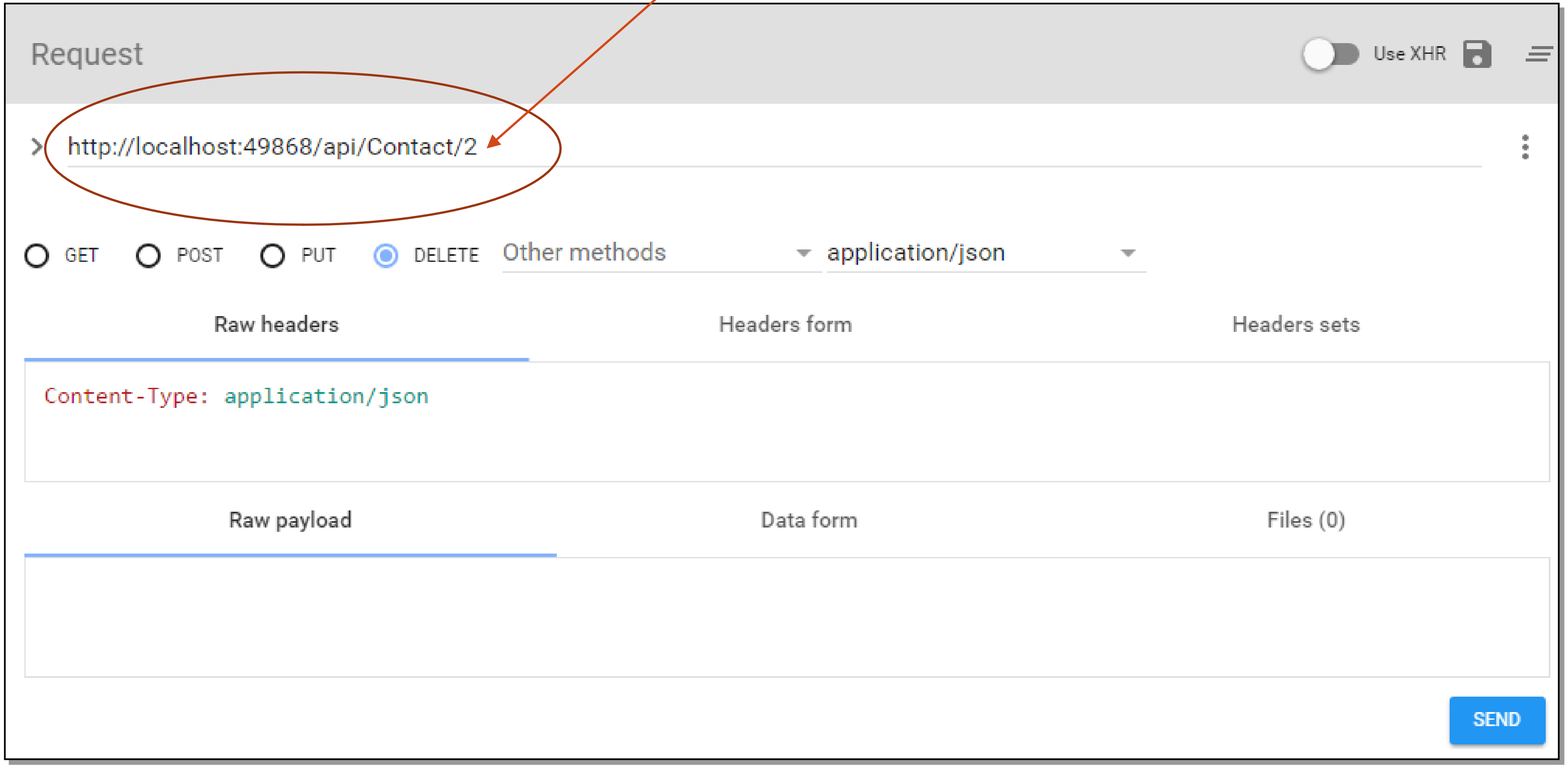
Instead, to get a response in xml format
Use «application/xml»



```
^ <Contact>
     <_x003C_address_x003E_k__BackingField>
     Via senza nome 1
     </_x003C_address_x003E_k__BackingField>
     <_x003C_id_x003E_k__BackingField>5
     </_x003C_id_x003E_k__BackingField>
     <_x003C_name_x003E_k__BackingField>Stefano
     </_x003C_name_x003E_k__BackingField>
     <_x003C_phone_x003E_k__BackingField>123456
     </_x003C_phone_x003E_k__BackingField>
     <_x003C_surname_x003E_k__BackingField>Bianchi
     </_x003C_surname_x003E_k__BackingField>
  </Contact>
```

# How to add delete a contact trough API

Delete contact with ID=2



Request

Use XHR

http://localhost:49868/api/Contact/2

GET [✓ DELETE] POST PUT

Other methods

application/json

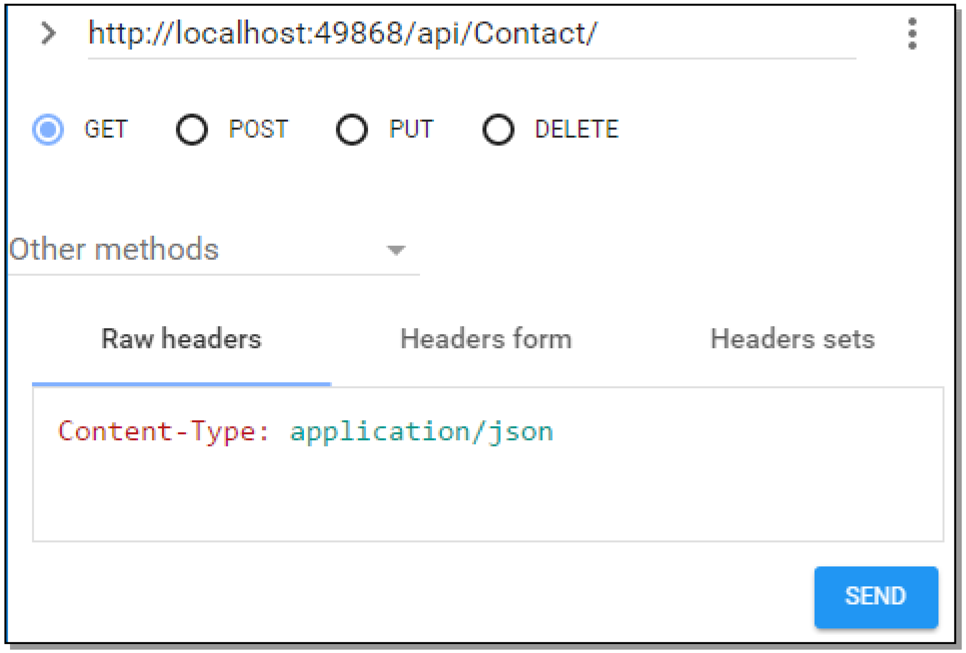
Raw headers | Headers form | Headers sets

```
Content-Type: application/json
```

Raw payload | Data form | Files (0)

SEND

# How to list contacts trough API



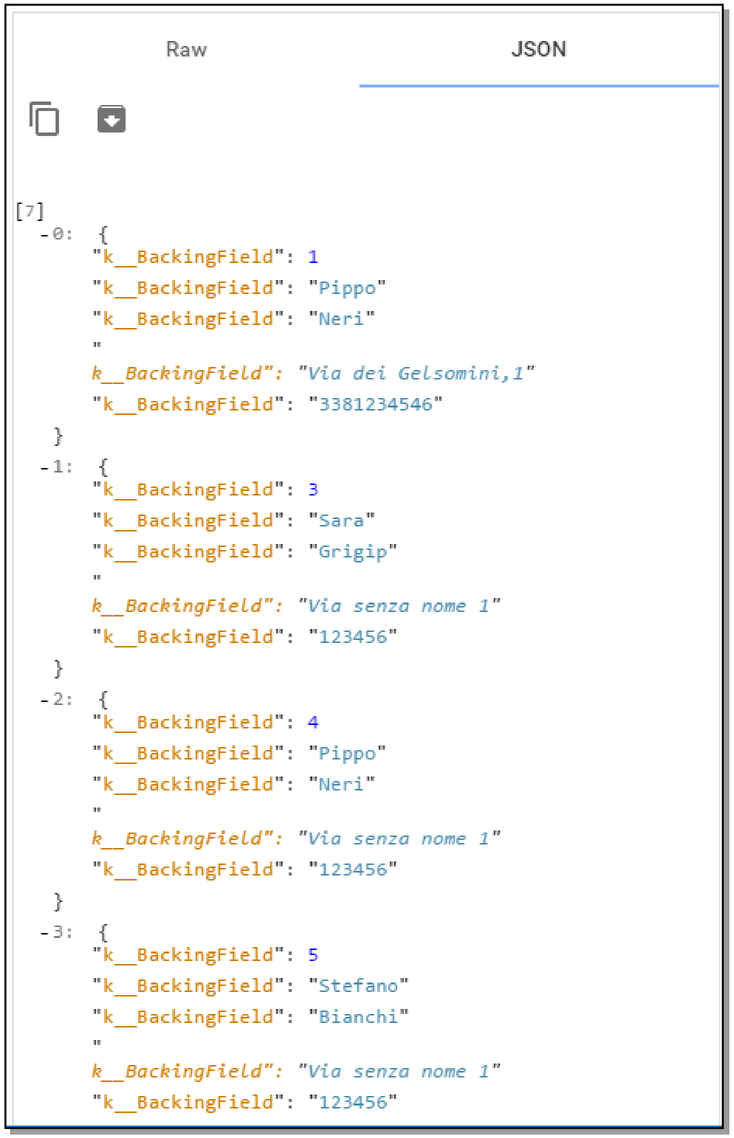
http://localhost:49868/api/Contact/

[✓ GET] POST PUT DELETE

Other methods

Raw headers | Headers form | Headers sets

```
Content-Type: application/json
```



```
[7]
 -0:  {
      "k__BackingField": 1
      "k__BackingField": "Pippo"
      "k__BackingField": "Neri"
      "
      k__BackingField": "Via dei Gelsomini,1"
      "k__BackingField": "3381234546"
   }
 -1:  {
      "k__BackingField": 3
      "k__BackingField": "Sara"
      "k__BackingField": "Grigip"
      "
      k__BackingField": "Via senza nome 1"
      "k__BackingField": "123456"
   }
 -2:  {
      "k__BackingField": 4
      "k__BackingField": "Pippo"
      "k__BackingField": "Neri"
      "
      k__BackingField": "Via senza nome 1"
      "k__BackingField": "123456"
   }
 -3:  {
      "k__BackingField": 5
      "k__BackingField": "Stefano"
      "k__BackingField": "Bianchi"
      "
      k__BackingField": "Via senza nome 1"
      "k__BackingField": "123456"
```