Labo: Garbage App: Deel 1

Doelstellingen

- Dit labo heeft als doel een volledige opdracht uit te werken met behulp van Azure Functions. De gemaakte services zal u daarna aanspreken in de lessen rond Android.
- Kennismaken met ADO.NET voor het wegschrijven van data naar SQL Azure

Opdrachtstelling

We wensen een mobile app te ontwikkelen waarmee mensen zwerfvuil kunnen registreren. Via een Android App is het de bedoeling dat mensen bij het zien van zwerfvuil dit kunnen doorsturen naar een database in de cloud. Daarnaast moeten ze al hun zwerfvuil registraties terug kunnen opvragen, wijzigen of verwijderen. Een registratie bestaat uit volgende gegevens: naam en email, van de persoon die registratie invult (beide verplicht), omschrijving van het zwerfvuil (verplicht), categorie van het zwerfvuil (verplicht), de straat (verplicht), de gemeente (verplicht), geschatte hoeveelheid in kg (verplicht), de latitude en longitude (niet verplicht).

*De verschillende categorieën zijn: **tuinafval, klein huishoudelijk afval, gevaarlijk afval en ik weet het niet**. We moeten ook het tijdstip van registratie opslaan. **Technische vereisten**: maak gebruik van **SQL Azure** voor de database. De webservices schrijf je in **Azure Functions**. De mobile app zal u maken in de lessen Android*

Stap 1: ER Diagram

Probeer het **ER diagram** te maken met de nodige tabellen en kolommen. Indien het ER-diagram klaar is gelieve dit aan te maken in de **SQL Azure Database**. Maak hiervoor een nieuwe database aan met bv als naam **GarbageApp**. Voeg daarna de nodige tabellen toe. Zorg er ook voor dat de categorieën aanwezig zijn in de database, steek ook een aantal steden in. U kan dit doen via de **SQL Management** Studio door daar de nodige **INSERT** statement te schrijven. Vergeet ook niet de **firewall** te configureren. Sla ook de **connectionstring** op. We hebben deze straks nodig. **Maak gebruik van GUID voor de nodige ID's**.

Stap 2: Visual Studio project

- Maak een nieuwe Azure Function App in de portal
- Maak een lege Private Git Repo
- Maak een nieuw Azure Function App in Visual Studio en voeg toe aan GIT
- Koppel de GIT repo aan de Azure Function zodat alles automatisch online komt.

Stap 3: Welke functies hebben we nodig?

Probeer eerst zelf op basis van de opdrachtstelling te bepalen welke functies je allemaal nodig zal hebben. Plaats er ook het juiste **HTTP Verb** bij. Hieronder kan u mijn keuze zien:

- Ophalen steden (GET)
- Ophalen afval categorieën (GET)
- Toevoegen Registraties (POST)
- Ophalen van AL MIJN registraties (GET)
- Ophalen van één registratie
- Verwijderen van één Registratie (DELETE)
- Wijzigen van één registratie (PUT)

Stap 4: Ophalen van Steden service

De eerste service die we samen ontwikkelen is het **GetCities** service. De URL die we daarvoor wensen te gebruiken is **garbage/cities**. Voeg een Azure Function toe aan uw project. De functie ziet er bij mij als volgt uit:

```
[FunctionName("GetCities")]
public static HttpResponseMessage GetCities([HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "garbage/cities")]HttpRequestMessage req, TraceWriter log)
```

Voeg nu ook een map **Models** toe waarin we de modellen zullen plaatsen die nog zijn. Voeg een klasse City toe aan deze map. Deze klassen zullen we gebruiken om de steden uit de database te halen en daarna om te zetten naar JSON. Deze ziet er als volgt uit:

```
public class City
{
    [JsonProperty("cityid")]
    public Guid CityId { get; set; }
    [JsonProperty("name")]
    public string Name { get; set; }
}
```

Op dit moment missen we nog één belangrijk stuk kennis om deze service te maken. We hebben reeds gezien hoe je vanuit Python kan praten met SQL Azure maar we hebben dit nog niet gezien vanuit C#. Gelukkig zijn de concepten ongeveer hetzelfde.

1. We zullen een connectie (**SqlConnection**) maken via de **connectionstring** met de database.
2. Als we een connectie hebben dan kunnen wie een **SqlCommand** het SQL Statement(SELECT,INSERT,…) opstellen en uitvoeren
3. Bij het ophalen van data hebben we nog een **SqlDataReader** nodig om de data uit te lezen en in een object te plaatsen.

BELANGRIJK

De connectionstring slaan we op in de **local.settings.json**. Deze file zal **NIET** in GIT zitten.

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "",
    "AzureWebJobsDashboard": "",
    "BlobAccount": "nmctcloud1storage",
    "BlobAccountKey": "eCGLbBfVWMffCcUKhODtRZRkrUixQ
    "ConnectionString": "Server=tcp:cloud1-nmct.data
  }
}
```

Het ophalen uit de **local.settings.json** file doen we via volgende code:

```
Environment.GetEnvironmentVariable("Connectionstring");
```

Hieronder kan u de volledige code zien voor het ophalen van steden. We declareren eerst een lijst van City objecten die we daarna zullen opvullen.

We maken via het C# **using** statement een **SqlConnection** object aan. In de **constructor** geven we de **connectionstring** mee (die kan u vinden in de portal). Daarna openen we de connectie met de database via de methode **Open()** van het **connection** object. Nu is de verbinding open en kunnen we SQL Statement sturen.

We maken een **SqlCommand** object aan. Via dit object kunnen we SQL Statement doorsturen. We stellen het SQL Statement op "**SELECT * FROM City**". We steken dit in de property **CommandText** van het command object. Nu moeten we nog het commando uitvoeren. We roepen de methode ExecuteReader() aan van het **command** object. Het resultaat is een **SqlDataReader**.

Via deze reader halen we nu één voor één de rijen uit de database. Voor iedere rij maken we een nieuw City object aan. We plaatsen iedere object in de lijst.

Wanneer we klaar zijn met lezen (**dus uit de while lus**) dan moeten we lijst omzetten in **JSON** en sturen we deze terug. We zorgen ook voor de juiste **statuscode**.

Merk ook op dat alle code in een **try/catch** staat. Bij een **exception** sturen we een **internal server error** terug. Naar de client. **Stuur NOOIT de exceptie zelf door naar de client**.

Als de compilatie lukt dan kan je het project opstarten en test je via Postman de URL uit.

```
try
{
    List<City> cities = new List<City>();
    using (SqlConnection connection = new SqlConnection(CONNECTIONSTRING))
    {
        connection.Open();
        using (SqlCommand command = new SqlCommand())
        {
            command.Connection = connection;
            string sql = "SELECT * FROM City";
            command.CommandText = sql;
            SqlDataReader reader = command.ExecuteReader();
            while (reader.Read())
            {
                City city = new City();
                city.CityId = Guid.Parse(reader["CityId"].ToString());
                city.Name = reader["Name"].ToString();
                cities.Add(city);
            }
        }
    }

    return req.CreateResponse(HttpStatusCode.OK, cities);
}
catch (Exception ex)
{
    return req.CreateResponse(HttpStatusCode.InternalServerError);
}
```

Stap 5: Ophalen afvaltypes

Op basis van stap 4 moet u volledige zelfstandig stap 5 kunnen maken.

Stap 6: Toevoegen van een registratie

We kunnen nu zowel steden als afvaltypes opvragen. Een laatste iets moeilijker stap is het toevoegen van een registratie. Via een HTTP Post ontvangen we een JSON String met daarin de registratiegegevens. Deze kan er als volgt uitzien:

```
{
 "GarbageRegistrationId" : "29d77476-021b-41ec-8792-38728aa591b4",
  "Description": "gras zakken maar ook snoei",
  "GarbageTypeId": "677939FC-F63D-4078-9DD4-73B289E20E5B",
  "CityId": "836FB0EB-E825-4F71-A056-7CC51ECBE914",
  "Weight": 1,
  "Lat": 0.0,
  "Long": 0.0,
  "User": "dieter"
}
```

Maak nu zelf het model aan voor registraties (zal overeenkomen met uw database tabel). Bij mij ziet dit er als volgt uit:

```
public class GarbageRegistration
{
    public Guid GarbageRegistrationId { get; set; }
    public string Description { get; set; }
    public Guid GarbageTypeId { get; set; }
    public Guid CityId { get; set; }
    public int Weight { get; set; }
    public float Lat { get; set; }
    public float Long { get; set; }
    public string User { get; set; }
    public DateTime Timestamp { get; set; }
}
```

De volgende code is het toevoegen van een record aan een database. De meeste zaken zijn hetzelfde, we maken connectie aan, we maken command aan. We stellen het INSERT-statement op.

Een belangrijk verschil is hier het gebruik van parameters. Via deze duiden we aan via **@naam**. .NET zal deze zelf invullen met de juiste waarden afkomst van **command.Parameter.AddWithValue**. U moet deze manier van werken gebruiken omwille van security. **PLAK NOOIT SQL STATEMENTS AAN ELKAAR VIA STRINGS**.

```
try
{
    var content = await req.Content.ReadAsStringAsync();
    var registration = JsonConvert.DeserializeObject<GarbageRegistration>(content);

    using (SqlConnection connection = new SqlConnection(CONNECTIONSTRING))
    {
        connection.Open();
        using (SqlCommand command = new SqlCommand())
        {
            command.Connection = connection;
            string id = Guid.NewGuid().ToString();
            string sql = "INSERT INTO GarbageRegistration VALUES (@id,@user,@description,@GarbageTypeId,@CityId,@weight,@lat,@long,@timstamp)";
            command.CommandText = sql;
            command.Parameters.AddWithValue("@id", id);
            command.Parameters.AddWithValue("@user", registration.User);
            command.Parameters.AddWithValue("@description", registration.Description);
            command.Parameters.AddWithValue("@GarbageTypeId", registration.GarbageTypeId);
            command.Parameters.AddWithValue("@CityId", registration.CityId);
            command.Parameters.AddWithValue("@weight", registration.Weight);
            command.Parameters.AddWithValue("@lat", registration.Lat);
            command.Parameters.AddWithValue("@long", registration.Long);
            command.Parameters.AddWithValue("@timstamp", DateTime.Now);
            command.ExecuteNonQuery();
            registration.GarbageRegistrationId = new Guid(id);
        }
    }
    var json = JsonConvert.SerializeObject(registration);
    return req.CreateResponse(HttpStatusCode.OK, json);
}
catch (Exception ex)
{
    return req.CreateResponse(HttpStatusCode.InternalServerError);
}
```

Via de code **ExectuteNonQuery** zullen we nu het SQL Statement toevoegen aan de database. Test dit nu uit via Postman.

Stap 7: Afwerking

Probeer nu alle andere nodig functies af te werken en test met Postman.