



Mocked Sensor Portal

Realisatie

Bachelor in de toegepaste informatica

Anton De Houwer

Academiejaar 2019-2020

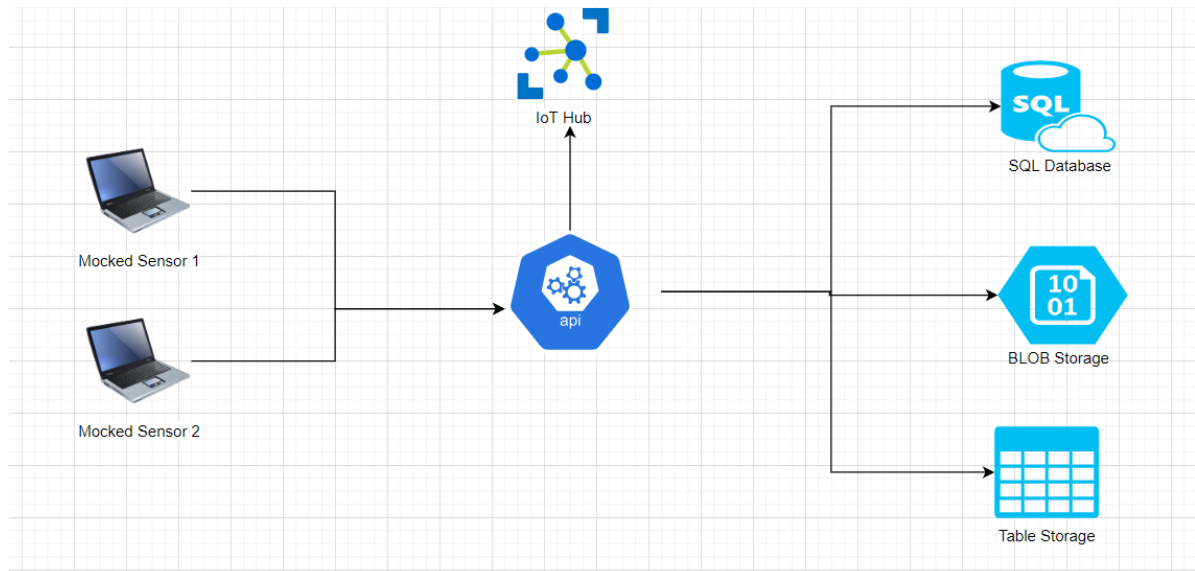
Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

INHOUDSTAFEL

INHOUDSTAFEL	2
INLEIDING	3
1 BACKEND API.....	4
1.1 SQL Database.....	4
1.1.1 Modellen	4
1.1.2 Database Context	5
1.1.3 Migration	6
1.2 BLOB storage	7
1.2.1 Welkom tekst	7
1.2.2 Payload.....	8
1.2.3 Flow payload ophalen per sensor	9
1.3 Table storage	10
1.4 Routes aanmaken	11
1.5 API-key	11
2 FRONT-END MOCKED SENSOR PORTAL.....	13
2.1 Startpagina	13
2.2 Connectie Pagina	14
2.2.1 Connectie toevoegen	16
2.2.2 Connectie wijzigen	17
2.2.3 Connectie verwijderen	18
2.3 Sensor pagina	19
2.3.1 Sensor toevoegen	20
2.3.2 Sensor wijzigen	21
2.3.3 Sensor verwijderen	21
3 'KEEP ALIVE'-SERVICE	24
3.1 IoT Hub	25
3.2 Hangfire.....	26
3.3 Recurring Jobs	26
3.4 CRON schedule	26
3.5 Hangfire dashboard	26
3.6 Monitoring messages	27

INLEIDING

In dit document wordt de realisatie van het Mocked Sensor portal besproken. De analyse die ik voor deze opdracht geschreven heb, kan je vinden in het bestand "Analyse Mocked Sensor Portal". Dit is een web portaal en het doel hiervan is om ervoor te zorgen dat de developer virtuele of mocked sensoren kan maken die gemanipuleerde waarden naar een Azure IoT Hub en/of IoT Central zal sturen.



Bij de ontwikkeling van deze opdracht heb ik de realisatie opgesplitst in drie zaken:

- **Backend API:** De API die door de Mocked Sensoren wordt aangesproken om enerzijds waarden te versturen naar de IoT Hub en anderzijds om logs of gegevens van de sensor op te slaan in een databank.
- **Front-End Mocked Sensor Portal:** Het web portaal waar de gebruiker sensoren kan gaan aanmaken.
- **'Keep Alive'-Service:** De service die ervoor gaat zorgen dat om de zoveel seconden of minuten de aanhangende payload van de sensor zal verstuurd worden naar de IoT Hub en/of Iot Central.

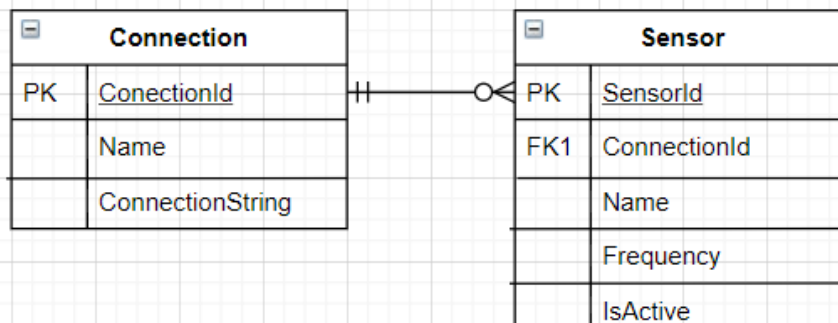
1 BACKEND API

Voor de Backend API van mijn portaal heb ik gebruikt gemaakt van asp.net core en Azure. .NET en Azure zijn beide van Microsoft en wordt bij VanRoey.be voortdurend gebruikt.



1.1 SQL Database

Allereerst heb ik een SQL Database toegevoegd op basis van het ERD dat ik maakte in mijn analyse. De database werd opgebouwd aan de hand van het Entity Framework. Ik maakte gebruik van "Code First". Dit wil zeggen dat tabellen voor de database automatisch gegenereerd worden op basis van geprogrammeerde klassen. Dit was mijn ERD-model:



1.1.1 Modellen

Om dergelijke tabellen aan te maken in de database, is er een model (klasse) nodig die met mijn code wordt aangemaakt:

```

16 references | Anton De Houwer, 29 days ago | 1 author, 4 changes
public class Sensor
{
    [Key]

    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    5 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
    public Guid SensorID { get; set; }
    [DataType(DataType.Text)]
    [Required]
    4 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
    public string Name { get; set; }
    1 reference | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
    public bool IsActive { get; set; }
    [Required]
    2 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
    public int Frequency { get; set; }
    3 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
    public Guid ConnectionID { get; set; }
    0 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
    public Connection Connection { get; set; }
}

```

```

public class Connection
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    5 references | Anton De Houwer, 36 days ago | 1 author, 2 changes | 0 exceptions
    public Guid ConnectionID { get; set; }
    [Required]
    [DataType(DataType.Text)]
    3 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
    public string Name { get; set; }
    [Required]
    2 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
    public string ConnectionString { get; set; }
    0 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
    public List<Sensor> Sensors { get; set; }
}

```

Zoals op de code te zien is, zijn de nodige relaties tussen tabellen (Foreign keys) ook aanwezig.

1.1.2 Database Context

Met "Code First" hebben we dus ook een database Context nodig. DbContext is een belangrijke klasse in Entity Framework API. Het is een brug tussen uw domein- of entiteitsklassen en de database.

```

1  using Microsoft.EntityFrameworkCore;
2  using VRA.IoT.MockedSensorPortal.DataAccess.Models;
3
4  namespace VRA.IoT.MockedSensorPortal.DataAccess.Data
5  {
6      9 references | Anton De Houwer, 16 days ago | 1 author, 4 changes
7      public class SensorContext : DbContext
8      {
9          0 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
10         public SensorContext(DbContextOptions<SensorContext> options) : base(options)
11         {
12             7 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
13             public DbSet<Connection> Connections { get; set; }
14             8 references | Anton De Houwer, 42 days ago | 1 author, 1 change | 0 exceptions
15             public DbSet<Sensor> Sensors { get; set; }
16         }
17     }
18 }

```

Hier voeg ik dan mijn twee modellen aan toe. Voor dat wij de database nu kunnen aanmaken moeten we eerst nog migrations toevoegen. Deze migrations zorgen ervoor dat onze database up to date blijft.

1.1.3 Migration

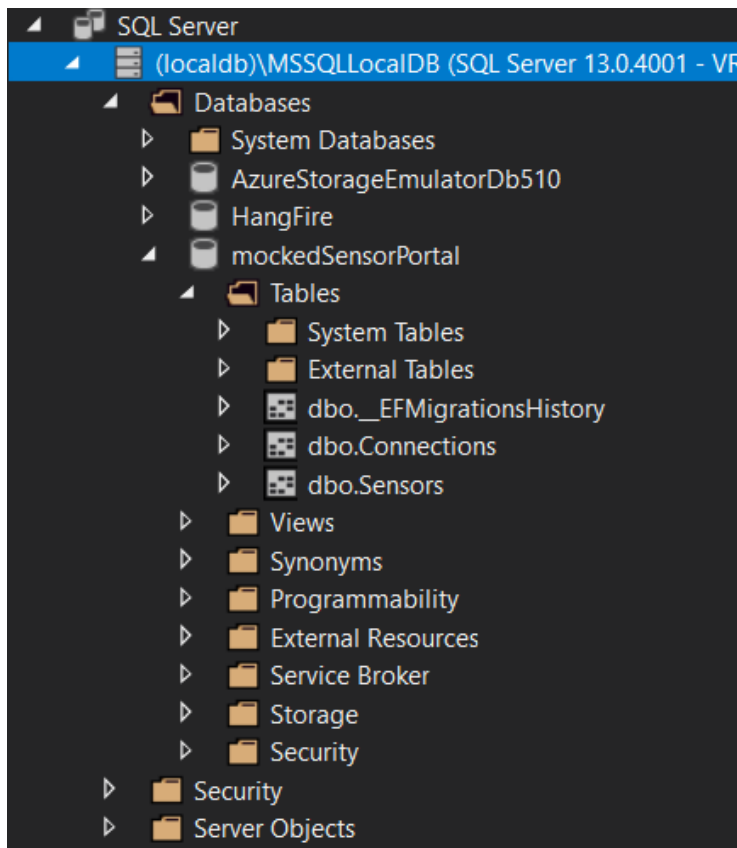
Toen alle modellen klaar waren, voerde ik een commando uit dat er voor zorgt dat er een *migration* wordt aangemaakt. Dat is een bestand waarin alle instructies staan om de geprogrammeerde database aan te maken/passen:

```

VRA.IoT.MockedSensorPortal  VRA.IoT.MockedSensorPortal.Migrations.Initial  Up(MigrationBuilder migrationBuilder)
1  using System;
2  using Microsoft.EntityFrameworkCore.Migrations;
3
4  namespace VRA.IoT.MockedSensorPortal.Migrations
5  {
6      1 reference | Anton De Houwer, 53 days ago | 1 author, 1 change
7      public partial class Initial : Migration
8      {
9          0 references | Anton De Houwer, 53 days ago | 1 author, 1 change | 0 exceptions
10         protected override void Up(MigrationBuilder migrationBuilder)
11         {
12             migrationBuilder.CreateTable(
13                 name: "Connections",
14                 columns: table => new
15                 {
16                     ConnectionID = table.Column<Guid>(nullable: false, defaultValueSql: "newsequentialid()"),
17                     Name = table.Column<string>(nullable: false),
18                     ConnectionString = table.Column<string>(nullable: false)
19                 },
20                 constraints: table =>
21                 {
22                     table.PrimaryKey("PK_Connections", x => x.ConnectionID);
23                 });
24             migrationBuilder.CreateTable(
25                 name: "Sensors",
26                 columns: table => new
27                 {
28                     SensorID = table.Column<Guid>(nullable: false, defaultValueSql: "newsequentialid()"),
29                     Name = table.Column<string>(nullable: false),
30                     IsActive = table.Column<bool>(nullable: false),
31                     Frequency = table.Column<int>(nullable: false),
32                     ConnectionID = table.Column<Guid>(nullable: false)
33                 }
34             );
35         }
36     }
37 }

```

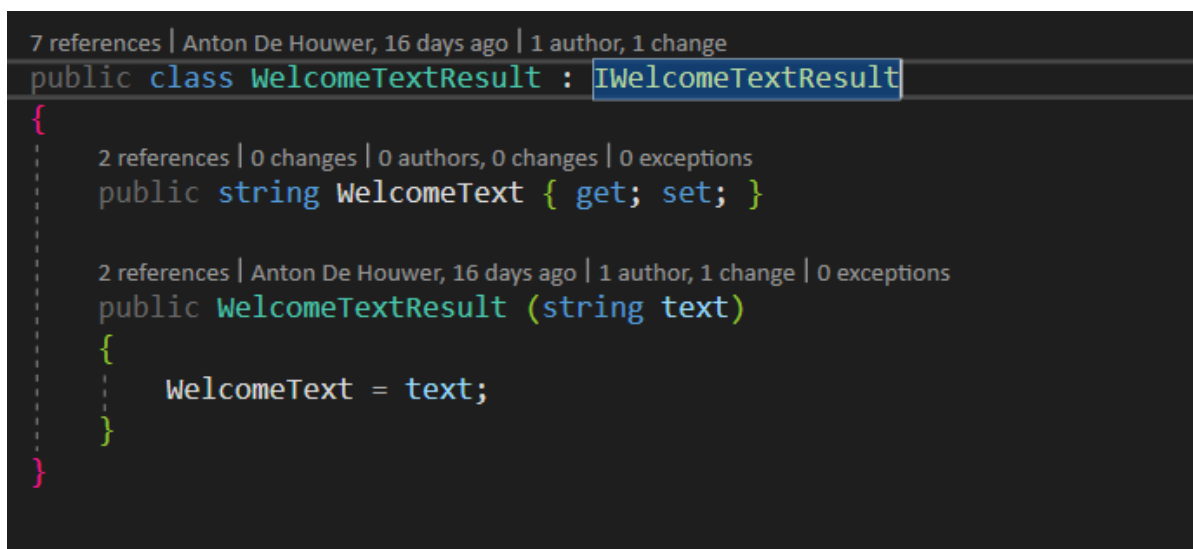
Wanneer ik het commando "Update-Database" uitvoerde in PowerShell, werd de *migration* uitgevoerd en de database aangemaakt:

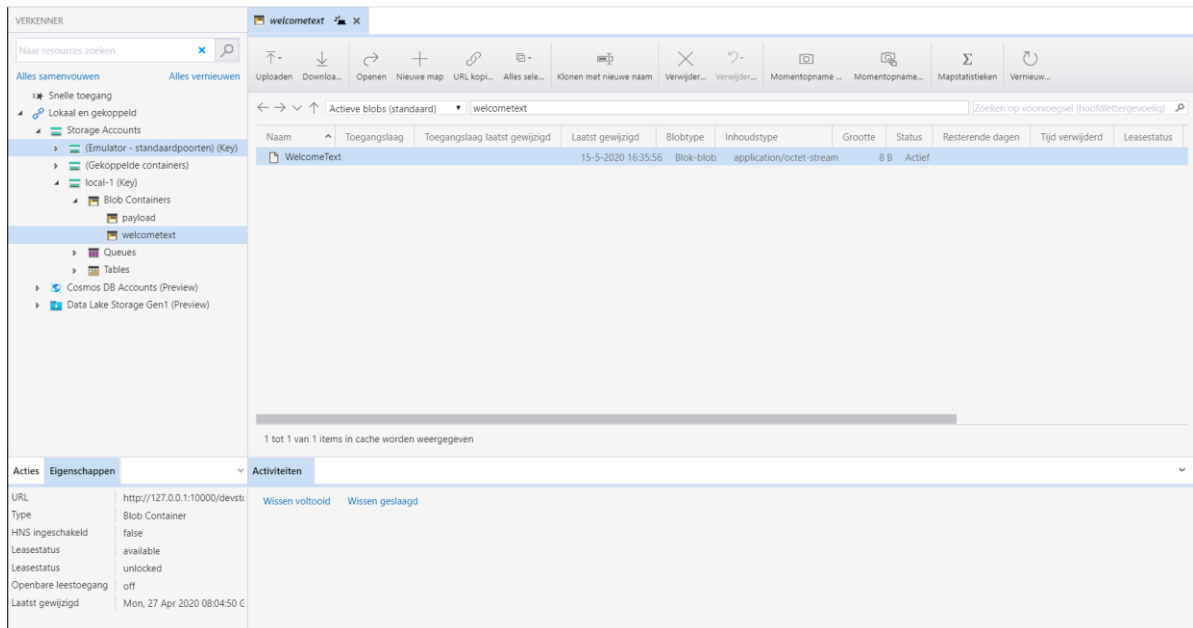


1.2 BLOB storage

De welkom tekst van het portaal en de payload van de sensor worden opgeslagen in een BLOB storage. Een BLOB storage is zeer handig voor het opslagen van een grote hoeveelheid ongestructureerde data. Doordat er geen relaties bestaan tussen de data is de BLOB storage redelijk snel met het opslaan en ophalen van zijn data.

1.2.1 Welkom tekst





VERKENNER

Naar resources zoeken

Alles samenvoegen Alles vernieuwen

Snelle toegang

Lokaal en gekoppeld

Storage Accounts

Emulator - standaardpoorten (Key)

(Gekoppelde containers)

local-1 (Key)

Blob Containers

payload

welcometext

Queues

Tables

Cosmos DB Accounts (Preview)

Data Lake Storage Gen1 (Preview)

Uploaden Download... Openen Nieuwe map URL kopi... Alles sele... Klonen met nieuwe naam Verwijder... Verwijder... Momentopname... Momentopname... Mapstatistieken Vernieuw...

Actieve blobs (standaard) | welcometext

Naam Toegangslaag Toegangslaag laatst gewijzigd Laatst gewijzigd Blobtype Inhoudstype Grootte Status Resterende dagen Tijd verwijderd Leasestatus

Naam	Toegangslaag	Toegangslaag laatst gewijzigd	Laatst gewijzigd	Blobtype	Inhoudstype	Grootte	Status	Resterende dagen	Tijd verwijderd	Leasestatus
WelcomeText			15-5-2020 16:35:56	Blok-blob	application/octet-stream	8 B	Actief			

1 tot 1 van 1 items in cache worden weergegeven

Acties Eigenschappen Activiteiten

URL http://127.0.0.1:10000/devst

Type Blob Container

HNS ingeschakeld false

Leasestatus available

Leasestatus unlocked

Openbare leesttoegang off

Laatst gewijzigd Mon, 27 Apr 2020 08:04:50 C

Wissen voltooid Wissen geslaagd

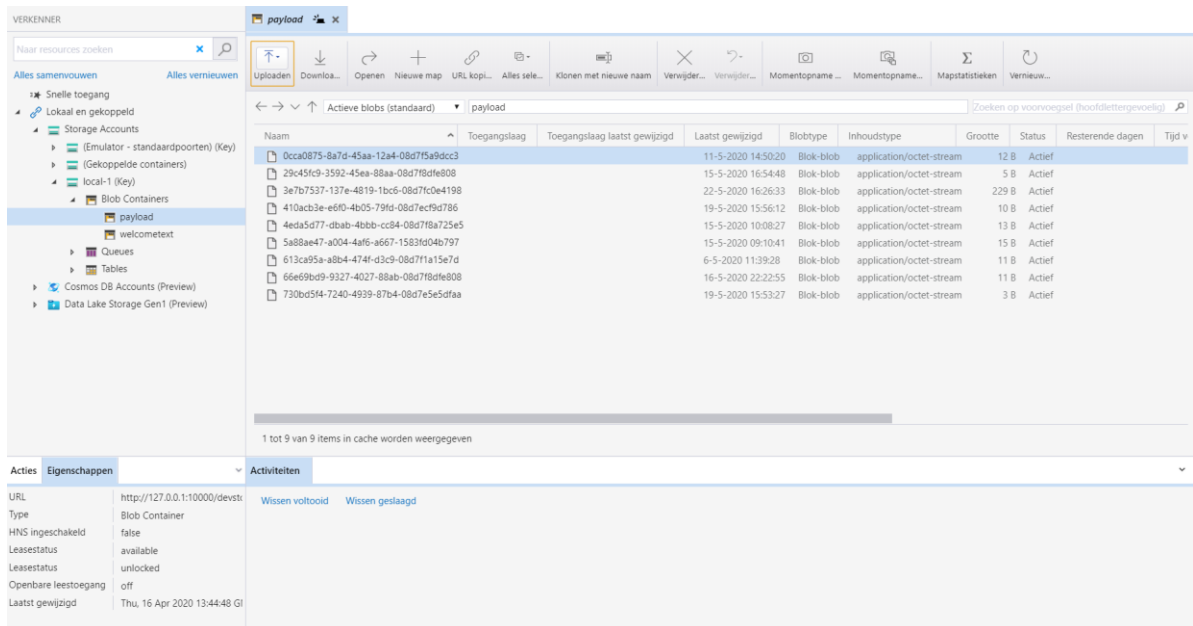
Zoals je hier kan zien wordt de welkom tekst in de "welcometext" Blob Container opgeslagen. Deze kan ik via Azure storage explorer bekijken.

1.2.2 Payload

```
public class SensorPayloadResult : ISensorPayloadResult
{
    3 references | Anton De Houwer, 16 days ago | 1 author, 1 change | 0 exceptions
    public Guid SensorId { get; set; }

    8 references | Anton De Houwer, 16 days ago | 1 author, 1 change | 0 exceptions
    public string Payload { get; set; }

    5 references | Anton De Houwer, 16 days ago | 1 author, 1 change | 0 exceptions
    public SensorPayloadResult(Guid sensorId, string payload)
    {
        SensorId = sensorId;
        Payload = payload;
    }
}
```

Naam	Toegangsleeg	Toegangsleeg laatste gewijzigd	Laatste gewijzigd	Blobtype	Inhoudstype	Grootte	Status	Resterende dagen	Tijd v
0cca0875-8a7d-45aa-12a4-08d7f5a9dcd3			11-5-2020 14:50:20	Blob-blob	application/octet-stream	12 B	Actief		
29c45fc9-3592-45ea-88aa-08d7f8dfe808			15-5-2020 16:54:48	Blob-blob	application/octet-stream	5 B	Actief		
3e7b7537-137e-4819-1bc5-08d7f0e4198			22-5-2020 16:26:33	Blob-blob	application/octet-stream	229 B	Actief		
410act3e-e6f0-4b05-79fd-08d7ec9d786			19-5-2020 15:56:12	Blob-blob	application/octet-stream	10 B	Actief		
4eda5d77-dbab-4bbb-cc84-08d7f8a725e5			15-5-2020 10:08:27	Blob-blob	application/octet-stream	13 B	Actief		
5a88ae47-a0d4-4af6-a667-1583f04b797			15-5-2020 09:10:41	Blob-blob	application/octet-stream	15 B	Actief		
613ca95a-a8d4-474f-d3c9-08d7f1a15e7d			6-5-2020 11:39:28	Blob-blob	application/octet-stream	11 B	Actief		
66e69bd9-9327-4027-88ab-08d7f8dfe808			16-5-2020 22:22:55	Blob-blob	application/octet-stream	11 B	Actief		
730bd5d4-7240-4939-87b4-08d7e5e5dfaa			19-5-2020 15:53:27	Blob-blob	application/octet-stream	3 B	Actief		

De payload van elke sensor wordt opgeslagen in de “payload” Blob container. Elke payload hangt aan een sensor. Dit zijn twee verschillende plaatsen waar zaken worden opgeslagen. Om ervoor te zorgen dat toch voor elke sensor de juiste payload wordt opgehaald, is de naam van elke payload het id van de bijhorende sensor.

1.2.3 Flow payload ophalen per sensor

```
public async Task<IApiResponse<IEnumerable<ISensorResult>>> GetAllSensorsAsync()
{
    var sensors = await _context.Sensors.ToListAsync();
    if (SensorsListIsEmpty(sensors))
        return ApiResponse<IEnumerable<ISensorResult>>.Success(new List<ISensorResult>());

    var results = _mapper.Map<IEnumerable<Sensor>, IEnumerable<ISensorResult>>(sensors);
    var payloads = await _sensorPayloadRepository.GetAllPayloadsAsync();
    AttachPayloadToSensors(ref results, payloads);

    return ApiResponse<IEnumerable<ISensorResult>>.Success(results);
}
```

```
1 reference | Anton De Houwer, 19 days ago | 1 author, 1 change | 0 exceptions
private void AttachPayloadToSensors(ref IEnumerable<ISensorResult> sensors, IEnumerable<ISensorPayloadResult> payloads)
{
    foreach (var sensor in sensors)
    {
        var payload = payloads.FirstOrDefault(x => x.SensorId.Equals(sensor.SensorID));
        if (payload == null)
            continue;

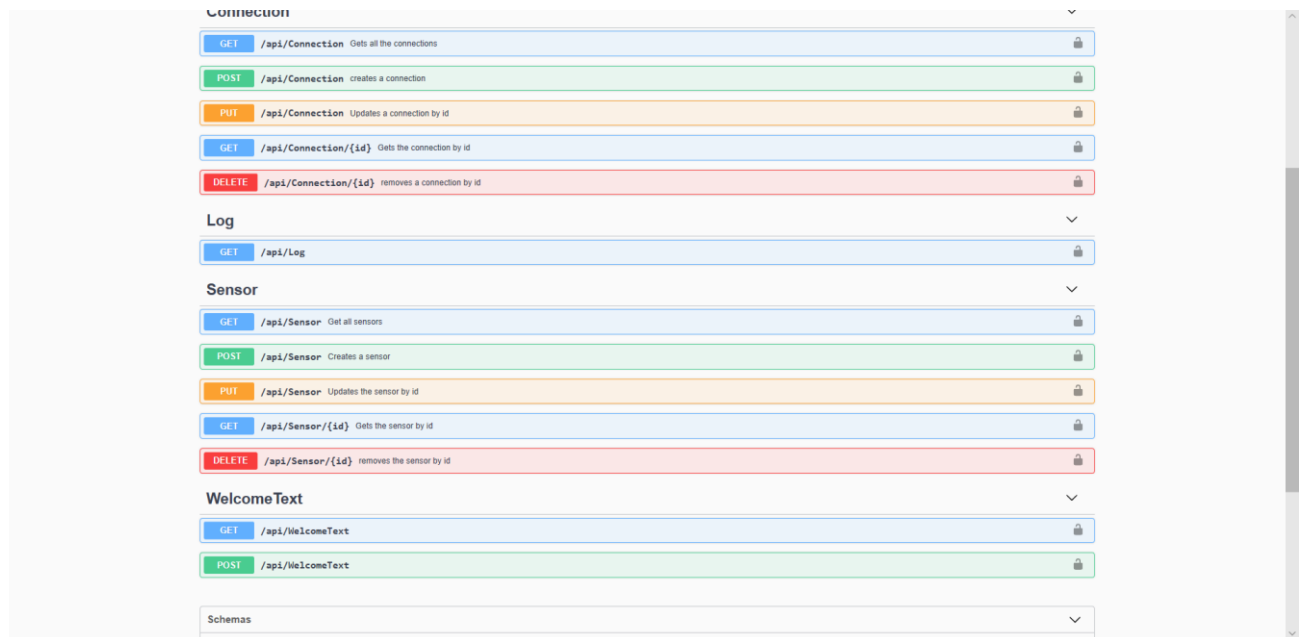
        sensor.Payload = payload.Payload;
    }
}
```

```
8 references | Anton De Houwer, 30 days ago | 1 author, 1 change
public class Log : TableEntity
{
    0 references | Anton De Houwer, 30 days ago | 1 author, 1 change | 0 exceptions
    public string FullMessage { get; set; }
    0 references | Anton De Houwer, 30 days ago | 1 author, 1 change | 0 exceptions
    public string Level { get; set; }
    0 references | Anton De Houwer, 30 days ago | 1 author, 1 change | 0 exceptions
    public string Message { get; set; }
}
```

Alle logs die mijn API doet worden hier in de "Logs" Table storage opgeslagen. Hier kan je dan makkelijk alle errors terug gaan herbekijken.

1.4 Routes aanmaken

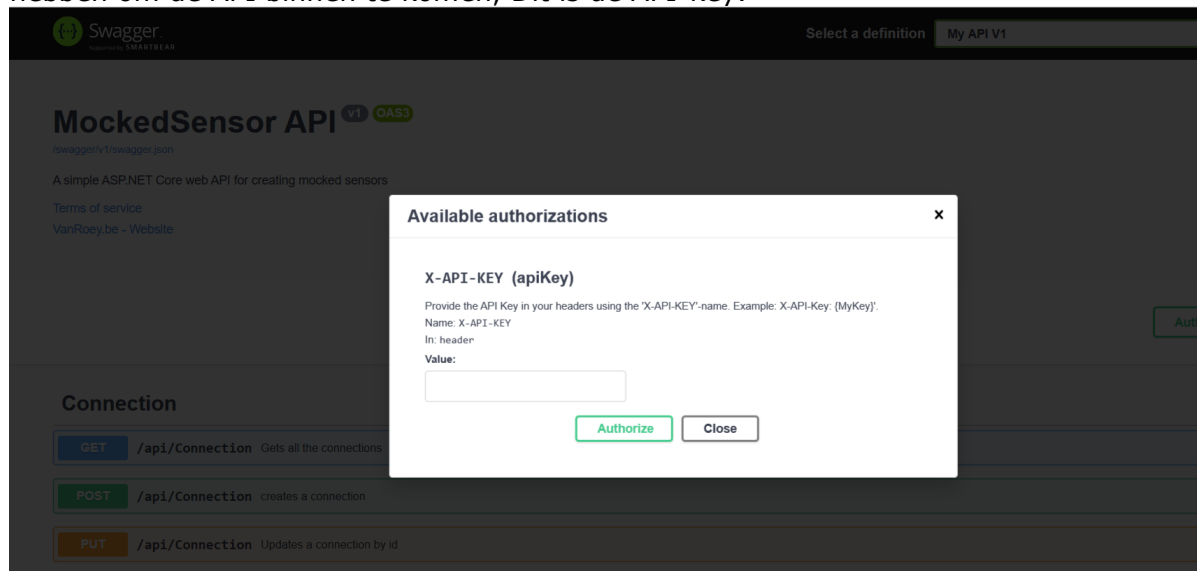
Om CRUD-operaties uit te voeren op de nieuwe tabellen, voorzag ik enkele routes in de API. Via deze routes kan men de nodige data toevoegen:

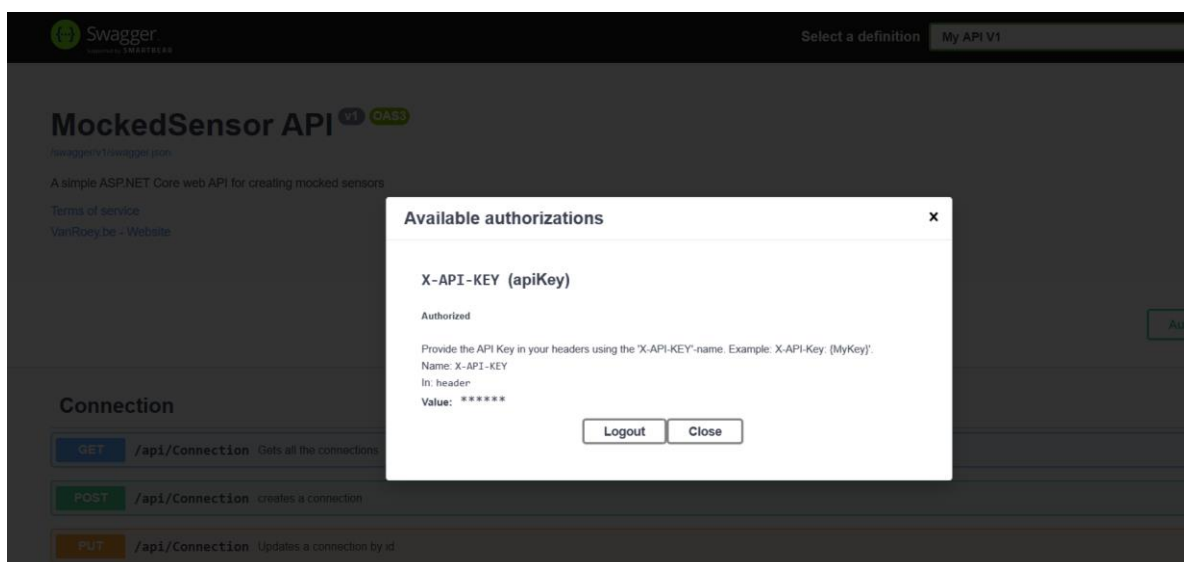


Met deze geïmplementeerde routes, is het nu mogelijk sensoren, connecties en een welkom tekst toe te voegen aan de database.

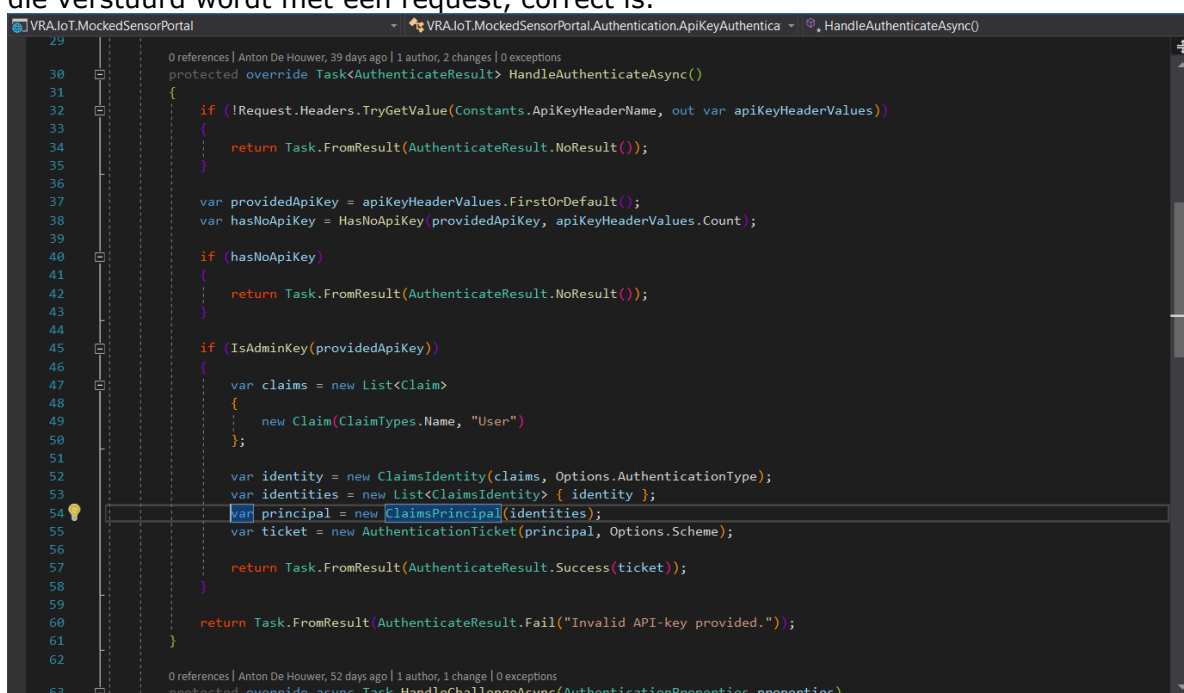
1.5 API-key

Om mijn API te beveiligen heb ik ervoor gezorgd dat elke request een sleutel moet hebben om de API binnen te komen, Dit is de API-key.





Via een ApiKeyAuthenticationHandler haal ik mijn API-Key op die in mijn appsettings.json file wordt opgeslagen. In deze handler wordt gekeken of de API-Key die verstuurd wordt met een request, correct is.



2 FRONT-END MOCKED SENSOR PORTAL

Op het Mocked Sensor Portal zijn er drie pagina's aanwezig:

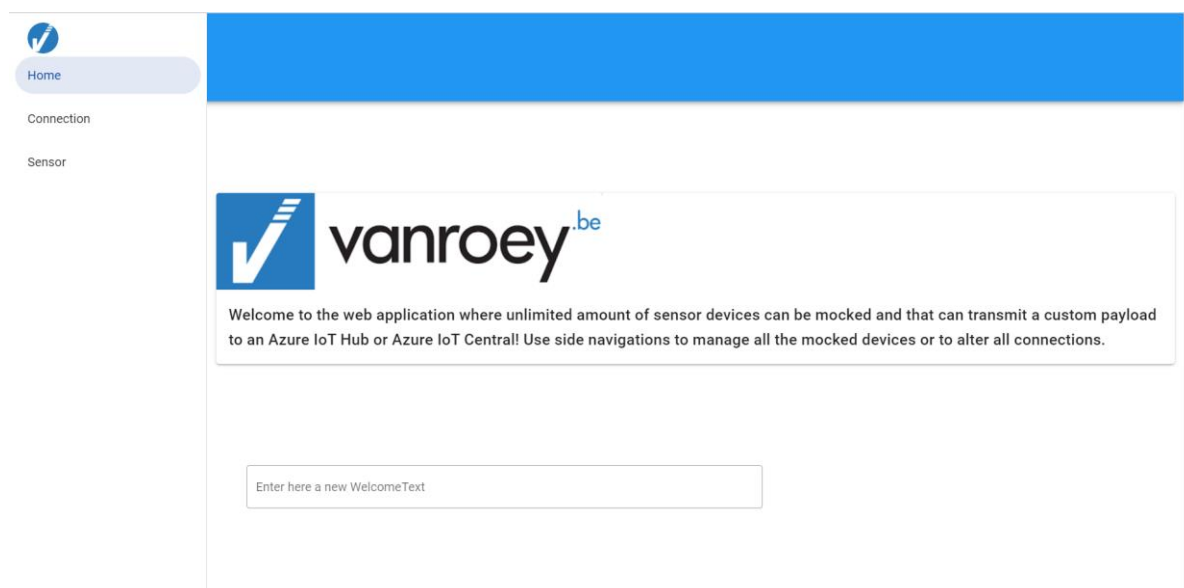
- De startpagina: Hier kan je de welkom tekst bekijken en wijzigen.
- Connectie pagina: Hier kan men alle connecties bekijken, een connectie wijzigen, aanmaken en verwijderen.
- Sensor pagina: Hier kan men alle sensoren bekijken, een sensor wijzigen, aanmaken en verwijderen.

Voor mijn Front-End maakte ik gebruik van Vue.js en typescript. Vue.js is een open-source framework voor het maken van single-page applications en interfaces. In Vue.js maakte ik gebruik van Vuetify. Vuetify is een material component framework waarmee ik heel gemakkelijk mijn web portaal kan ontwikkelen. Als ik alle Vue frameworks met elkaar vergeleek had Vuetify toch de meeste voordelen. Ik hield dan vooral rekening met de lange termijn voordelen en updates.



2.1 Startpagina

Dit is de startpagina van het portaal. De tekst van de startpagina wordt opgehaald uit de BLOB Storage die ik eerder al aangehaald heb. Er kan onderaan de pagina een nieuwe welkom tekst naar keuze ingevoerd worden. De welkom tekst wordt automatisch veranderd na het klikken op de enter toets of als er gewoon ergens uit het invoervak geklikt wordt.



2.2 Connectie Pagina

Op deze pagina zijn alle connecties te zien, deze connecties worden opgehaald van de SQL-Database. Via de zoekbalk rechts boven kan men filteren op naam van de connectie. Ook is er beneden aan de pagina paginatie toegevoegd om het overzichtelijk te maken voor de gebruiker.

Home

Connection

Sensor

+

Connections

Search

Connection Name	Connection String	actions
Connectie12	string	
123	string77	
se3	string	
string78999	string	
stringffff	stringqqqss	
strinsssg	striddng	
strinddsssg	striddng	
striddng	string	
string77774	string	
string77775	string	

Rijen per pagina: 10 1-10 van 18

Home

Connection

Sensor

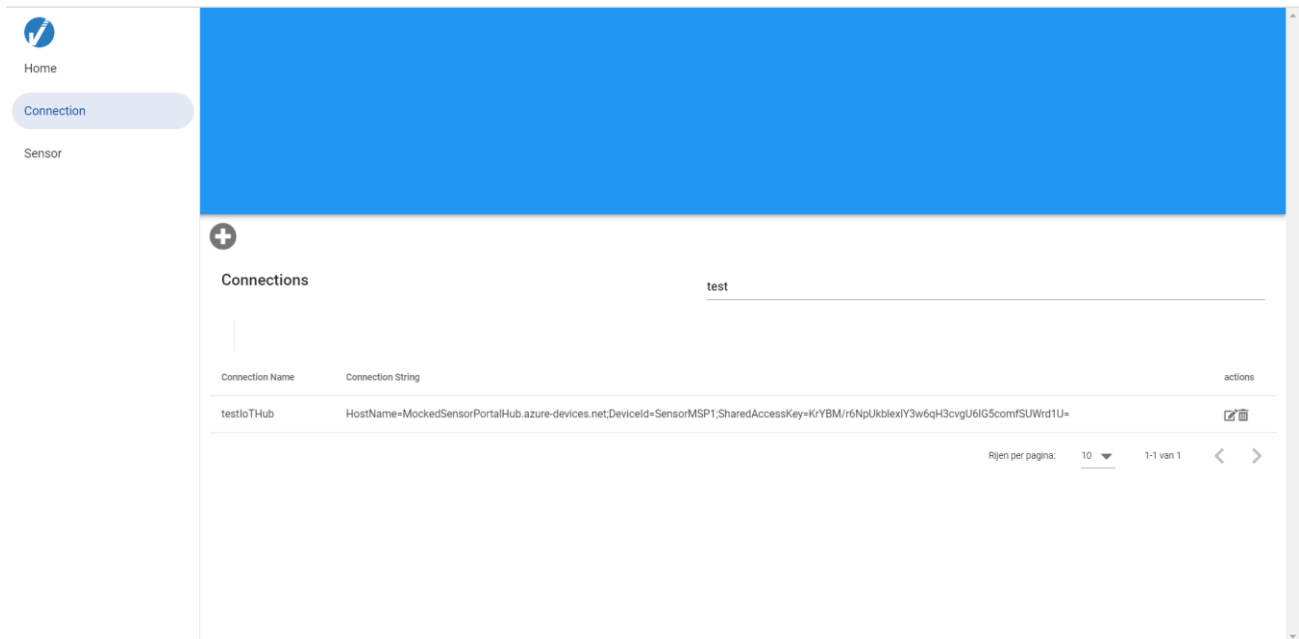
+

Connections

Search

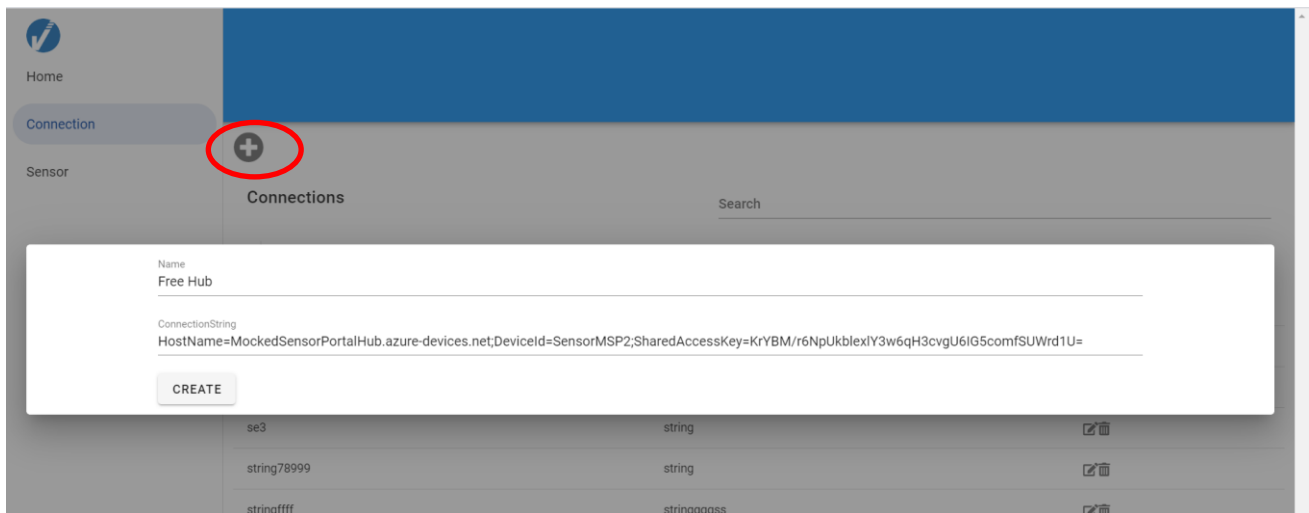
Connection Name	Connection String	actions
Connectie12	string	
123	string77	
se3	string	
string78999	string	
stringffff	stringqqqss	

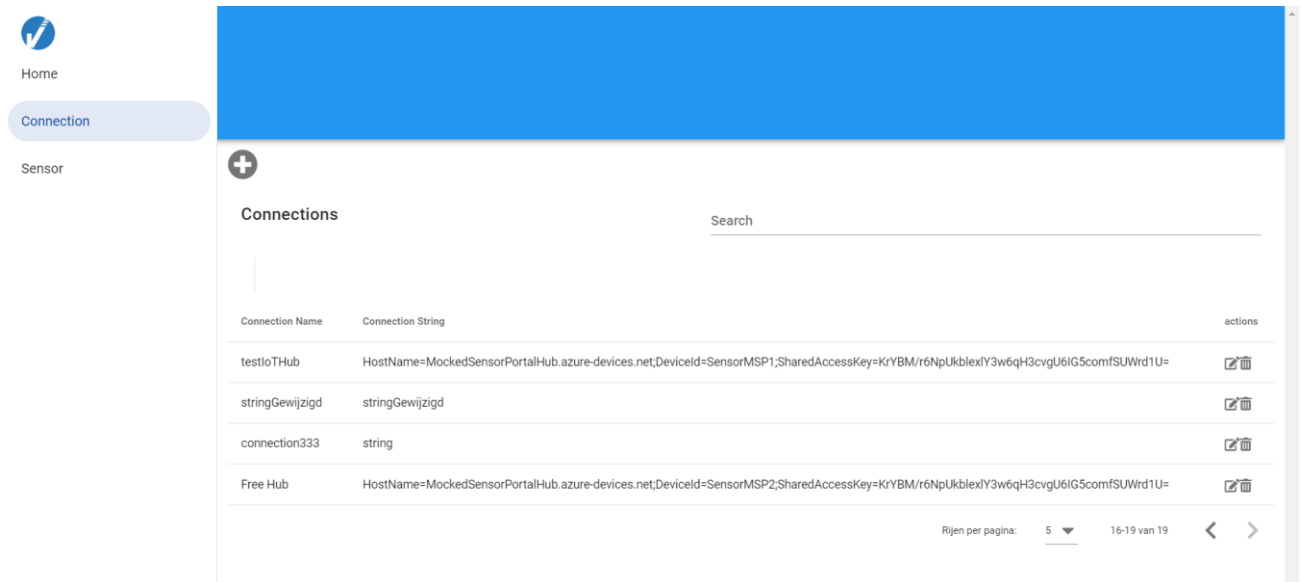
Rijen per pagina: 5 1-5 van 18



2.2.1 Connectie toevoegen

Door links boven op het plus icoontje te klikken, kan er een nieuwe connectie worden aangemaakt. Er komt een dialog tevoorschijn waar men de naam en de connectie string van de nieuwe connectie kan invoeren.

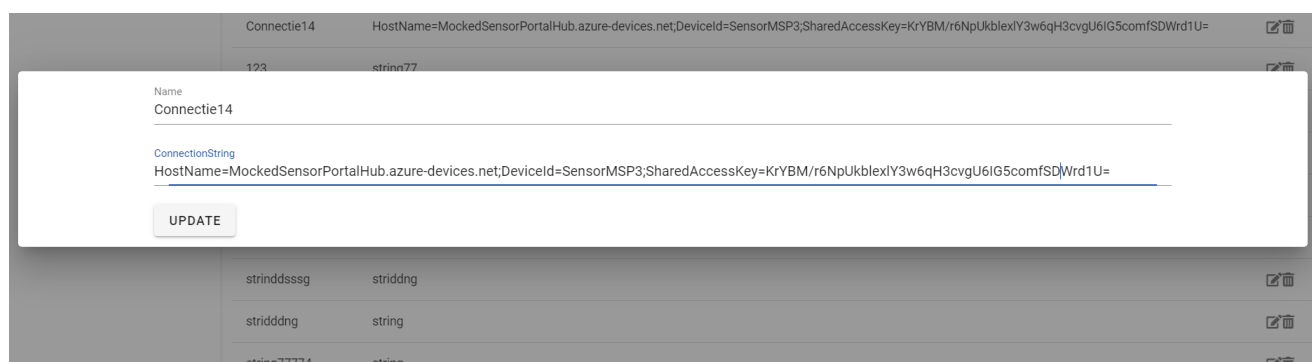
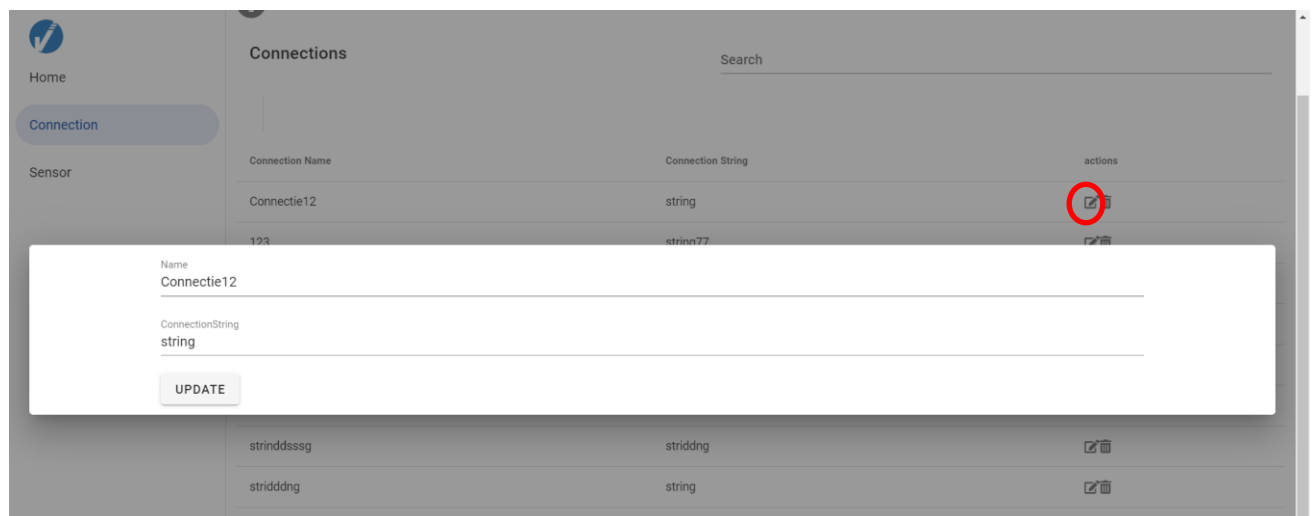


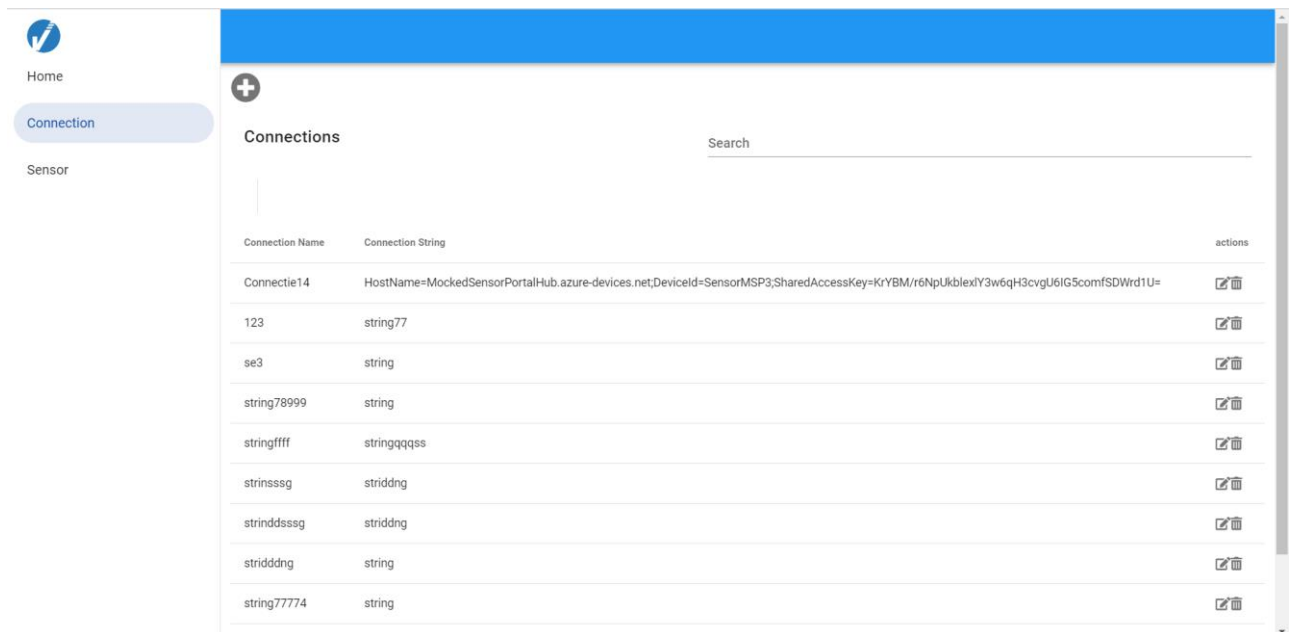


Nadat er op create wordt geklikt zal de connectie meteen worden toegevoegd aan de lijst.

2.2.2 Connectie wijzigen

Door op het potlood icoontje te klikken, kan een connectie worden gewijzigd. Dat potlood icoontje is telkens rechts van de connectie te vinden. Er zal een dialog tevoorschijn komen waar de gegevens van de gekozen connectie in kunnen gewijzigd worden.





Home



















Connection

Sensor

+

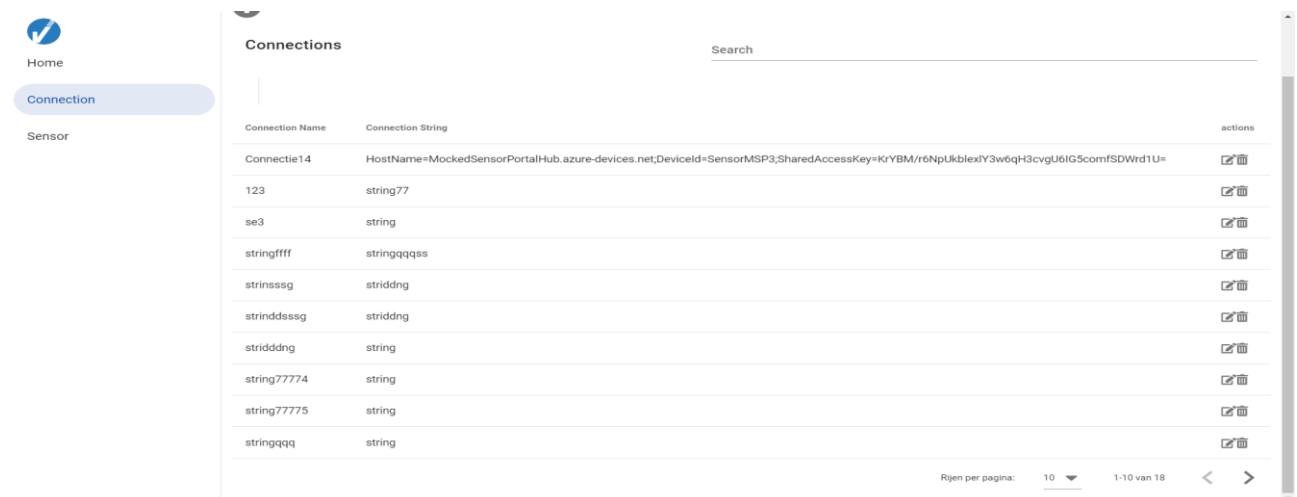
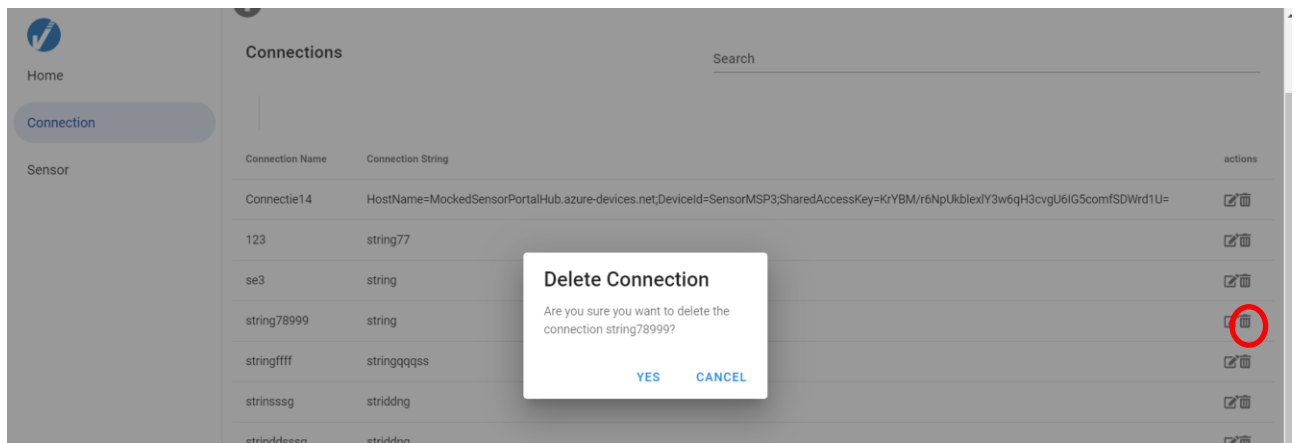
Connections

Search

Connection Name	Connection String	actions
Connectie14	HostName=MockedSensorPortalHub.azure-devices.net;DeviceId=SensorMSP3;SharedAccessKey=KrYBM/r6NpUkbleY3w6qH3cvGU6IG5comfSDWrd1U=	 
123	string77	 
se3	string	 
string78999	string	 
stringffff	stringqqqs	 
strinsssg	stridding	 
strinddsssg	stridding	 
striddng	string	 
string77774	string	 

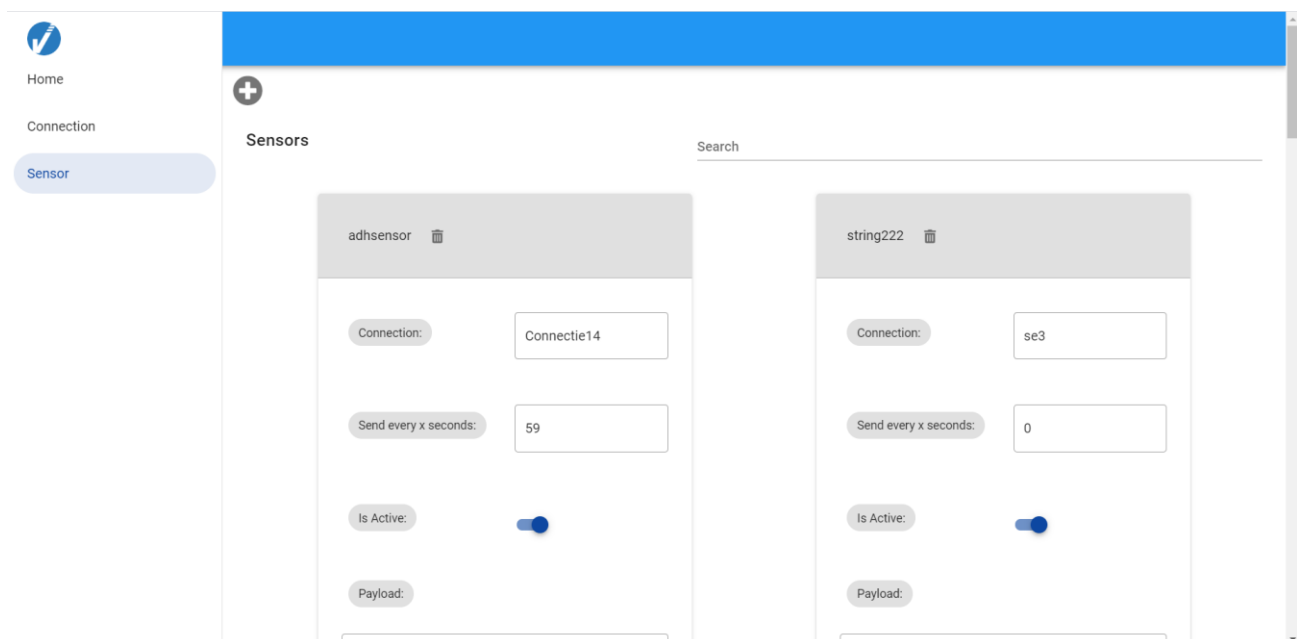
2.2.3 Connectie verwijderen


Door het op het vuilbak icoontje te klikken, kan er een connectie verwijderd worden. Het vuilbak icoontje is telkens rechts van de connectie te vinden. Als dit gebeurt, zal er eerst gevraagd worden aan de gebruiker of hij wel zeker is dat hij de gekozen connectie wil verwijderen. Zodra hij daarop bevestiging geeft zal de gekozen connectie verwijderd worden.



2.3 Sensor pagina

Op deze pagina zijn alle sensoren te zien, deze sensoren worden opgehaald van de SQL-Database. Via de zoekbalk rechts boven kan men filteren op naam van de sensor. Om een mooi overzicht te tonen, wordt elke sensor telkens in een card getoond.



sensorTest 

Connection:

testIoTHub

Send every x seconds:

42


Is Active:

☐

Payload:

```
{
  "name": "The Anton
Device",
  "latitude": "{func(random(10,
20))}",

```


Home


Connection

Sensor

+

Sensors

ad

adhsensor 

Connection:

Connectie14


Send every x seconds:

59

Is Active:

☒

Payload:

adh14 

Connection:

123

Send every x seconds:

12

Is Active:

☐

Payload:

2.3.1 Sensor toevoegen

Door links boven op het plus icoontje te klikken, kan er een nieuwe sensor worden aangemaakt. Er komt een dialog tevoorschijn waar men de naam, frequentie, connectie

en de payload voor de nieuwe sensor kan invoeren. De connecties worden aan de hand van een keuzelijst getoond om het zo simpel mogelijk te maken voor de gebruiker.

The screenshot shows a web application interface for managing sensors. On the left is a sidebar with a checkmark icon and links for 'Home', 'Connection', and 'Sensor'. The main area is titled 'Sensors' and contains a form to create a new sensor. The form has the following fields:

- Name:** Mocked temperature and GPS
- Frequency:** 25
- Connection:** Free Hub (selected from a dropdown menu)
- Payload:** `{"id": "temp device", "tmp": "{func(rand(10,25))}"}`

At the bottom of the form is a 'CREATE' button. Below the form, there is a table of existing sensors. The first sensor is shown with a toggle switch labeled 'Is Active:' which is currently turned on.

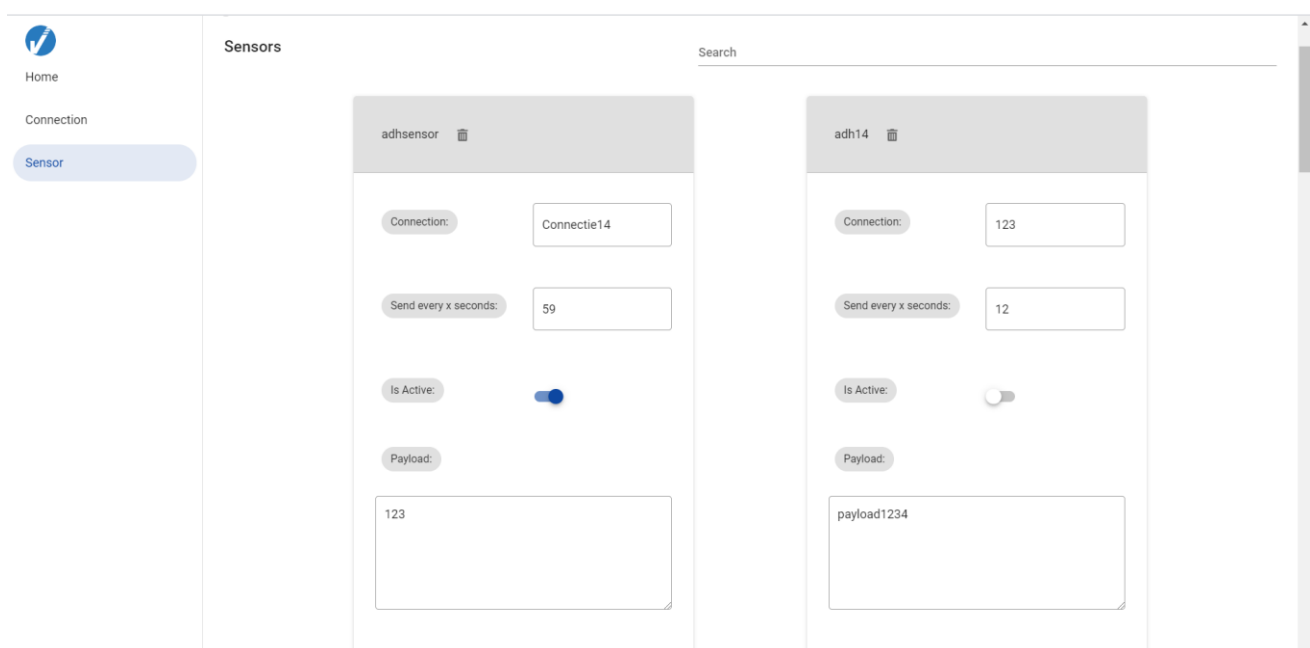
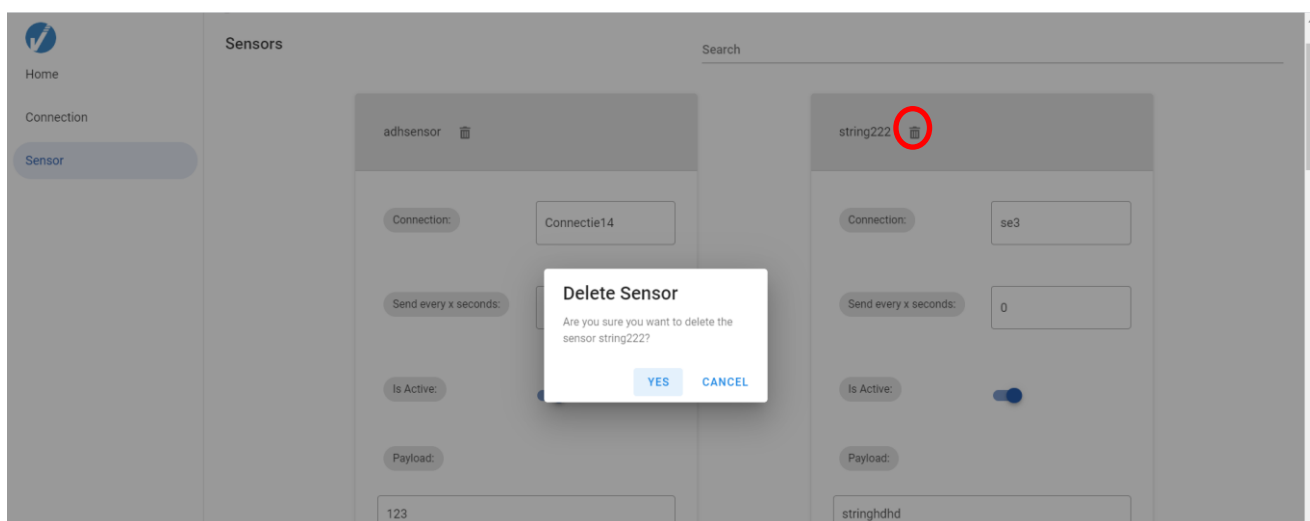
Zodra er op create wordt geklikt zal de sensor meteen in een nieuwe card van onder aan de lijst te zien zijn. Nieuwe sensoren zullen ook default op niet-actief staan, de gebruiker moet hierna nog de sensor op actief zetten voor deze zijn payload kan versturen.

2.3.2 Sensor wijzigen

Door de gegevens van de sensor in de card te veranderen zullen deze meteen ook gewijzigd worden. Hierdoor kan de gebruiker op snelle en eenvoudige manier een sensor wijzigen.

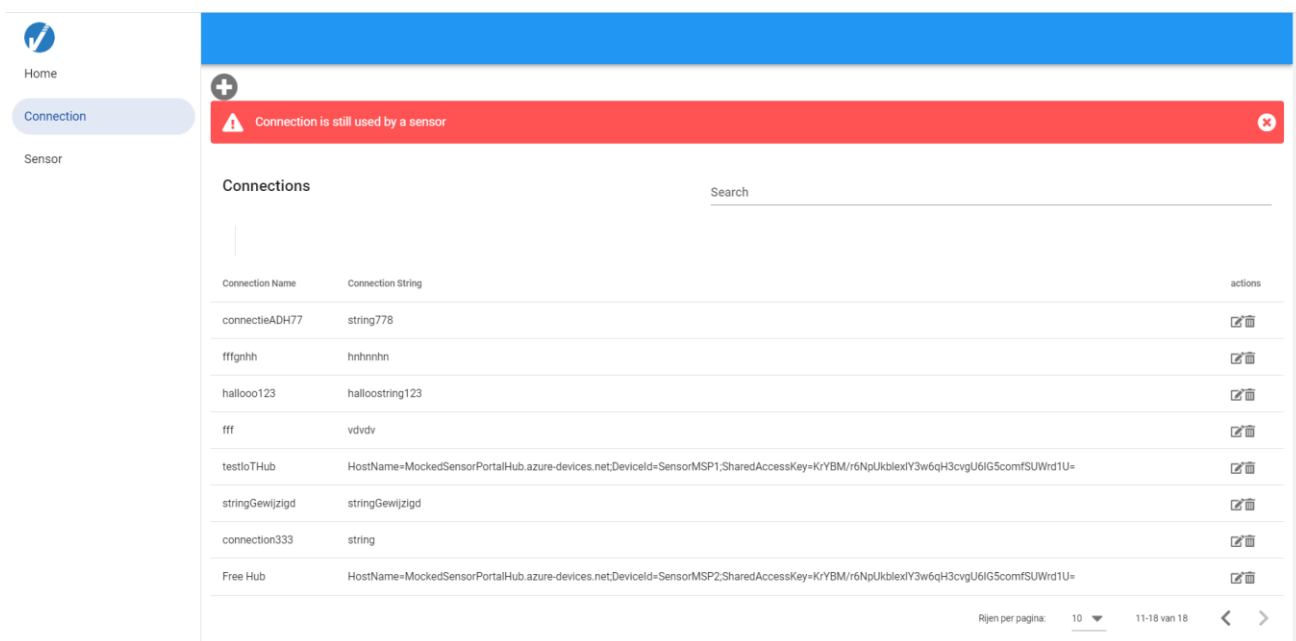
2.3.3 Sensor verwijderen

Door op het vuilbak icoontje te klikken, kan er een sensor verwijderd worden. Dit vuilbak icoontje is telkens rechts van de naam van de sensor te vinden. Als dit gebeurt, zal er eerst gevraagd worden aan de gebruiker of hij wel zeker is dat hij de gekozen sensor wil verwijderen. Zodra hij dit bevestigt, zal de gekozen sensor verwijderd worden.



2.4 Error opvangen

Errors die worden gemaakt door het systeem of fouten die de gebruiker maakt, worden op een gebruiksvriendelijke manier getoond. Op het voorbeeld hieronder zie je dat als de gebruiker een connectie wil verwijderen die nog in gebruik is, error alert getoond wordt met daarbij de uitleg.



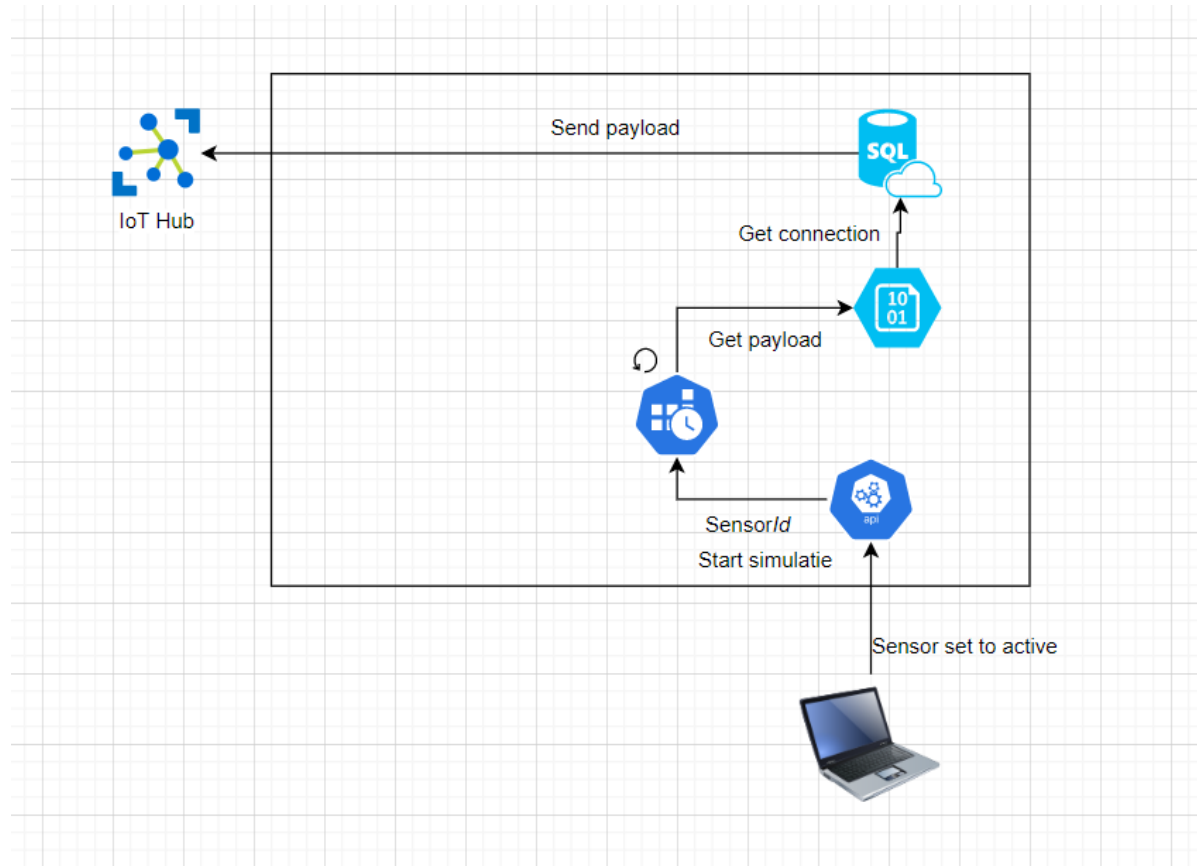
2.5 Vue observable

Met Vue observable kan ik objecten gaan controleren en updaten zonder de pagina opnieuw te laden. Ik gebruik dit op alle drie de pagina's zodat de gebruiker meteen het resultaat ziet van de acties die uitgevoerd worden.

```
export const sensorState = Vue.observable(new SensorState(sensorApiService));
export const welcomeTextState = Vue.observable(new WelcomeTextState(welcomeTextApiService));
export const connectionState = Vue.observable(new ConnectionState(connectionApiService));
```

3 'KEEP ALIVE'-SERVICE

Om alle actieve sensoren in leven te houden, maak ik een 'keep Alive'-Service dat om de aantal seconden zijn aanhangende payload gaat versturen naar de geregistreerde connectie. Voor deze Service maak ik gebruik van Hangfire dat later in dit document nog zal worden uitgelegd.



3.1 IoT Hub

IoT Hub is een cloud service van Azure dat waarden van een IoT device kan opslagen en doorsturen. Voor mijn opdracht heb ik voor het testen in Azure een IoT Hub aangemaakt. Hier heb ik dan een nieuw IoT device geregistreerd, waar ik later dan mijn sensor payload kan naar toe sturen. De connectiestring van IoT device dat juist geregistreerd is, moet dan later worden toegevoegd als nieuwe connectie in het portaal.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the 'Microsoft Azure' logo and a search bar. Below the navigation bar, there are sections for 'Azure services' (with icons for various services like Resource groups, IoT Hub, App Services, etc.) and 'Recent resources' (a table listing recently viewed resources).

The main content area displays the 'MockedSensorPortalHub | IoT devices' page. On the left, there is a sidebar with navigation options like 'Export template', 'Query explorer', 'IoT devices', 'Automatic Device Management', 'Messaging', 'Security', and 'Monitoring'. The main pane shows a table of IoT devices with the following data:

Device ID	Status	Last Status Update (UTC)	Authentication Type	Cloud to Device Message Count
SensorMSP1	Enabled	--	Sas	0

Below the table, the details for the 'SensorMSP1' device are shown. The 'Primary Connection String' field is highlighted with a red circle and contains the following text:

```
HostName=MockedSensorPortalHub.azure-devices.net;DeviceId=SensorMSP1;SharedAccessKey=KriYBM/r6NpUkblexY3w6qH3cvgU6lGScomFSUWrd1U=
```

Other fields visible include 'Device ID' (SensorMSP1), 'Primary Key', 'Secondary Key', 'Secondary Connection String', 'Enable connection to IoT Hub' (set to 'Enable'), and 'Parent device' (set to 'No parent device').

3.2 Hangfire

Een gemakkelijke manier om achtergrondverwerking uit te voeren in .NET- en .NET Core-applicaties. Hangfire is een open source software waar er verschillende manieren van achtergrondtaken aanwezig zijn. Ik maakte gebruik van recurring jobs, dit wil zeggen dat een taak bepaalde keren kan uitgevoerd worden aan de hand van een CRON schedule.

3.3 Recurring Jobs

Zoals eerder al aangehaald is, kan ik met Recurring jobs taken verschillende keren uitvoeren aan de hand van een CRON schedule. De sensor krijgt een frequentie mee die ik ga gebruiken in het CRON schedule van het RecurringJob om zo de 'Keep Alive'-Service op het juiste moment te laten uitvoeren. Ik maakte hiervoor in mijn API een aparte class library aan, dat enkel dient om de payload op te halen uit de BLOB Storage en te versturen naar de correcte connectie.

```
if (IsSensorActive(command.IsActive))
{
    var querySensor = GetSensorByIdQuery.FromUpdateSensorCommand(command.Id);
    var queryConnection = GetConnectionIdFromUpdateSensorCommand(command.ConnectionId);
    RecurringJob.AddOrUpdate(recurringJobId:"Cron-Job",methodCall: () => _cronJobService.RunTask(querySensor, queryConnection), $"{command.Frequency} * * * *");
    var sensor = await _sensorRepository.UpdateSensorAsync(command);
    var result = Mapper.Map<IApiResult<ISensorResult>, ApiResultDto<SensorDto>>(sensor);
    return Ok(result);
}
else
{
    RecurringJob.RemoveIfExists("Cron-Job");
    var sensor = await _sensorRepository.UpdateSensorAsync(command);
    var result = Mapper.Map<IApiResult<ISensorResult>, ApiResultDto<SensorDto>>(sensor);
    return Ok(result);
}
```

Hier wordt er dus eerst gekeken of de sensor op actief staat en indien dit zo is, wordt er RecurringJob uitgevoerd om de zoveel seconden.

3.4 CRON schedule

Met het CRON schedule kunnen wij ervoor zorgen dat een opdracht wordt uitgevoerd om de zoveel tijd. Bekijk hieronder de code die ik gemaakt heb voor het uitvoeren van mijn cron job.

```
RecurringJob.AddOrUpdate(recurringJobId:"Cron-Job",methodCall: () =>
_cronJobService.RunTask(querySensor, queryConnection), $"{command.Frequency} * * * *");
```

Het laatste deel dat groen gekleurd is, bevat het CRON-schedule waar ik de frequentie van de sensor mee aangeef zodat de taak telkens op het juiste tijdstip verstuurt wordt.

3.5 Hangfire dashboard

Via het Hangfire dashboard kan ik kijken welke taken er op dat moment worden uitgevoerd en of ze succesvol waren of niet.

Met Visual Studio Code kan ik de berichten die worden verstuurd van mijn Mocked sensor naar de IoT Hub bekijken. Door de IoT Hub extensie te downloaden kan ik enkele superhandige commando's uitvoeren voor het monitoren van mijn IoT Hub. Deze waardes krijgt de Developer dan binnen en zo kan hij de logica van zijn applicatie verder opbouwen.

```

[IoTHubMonitor] Start monitoring message arrived in built-in endpoint for all devices ...
[IoTHubMonitor] created partition receiver [0] for consumerGroup [$default]
[IoTHubMonitor] created partition receiver [1] for consumerGroup [$default]
[IoTHubMonitor] [2:01:25 PM] Message received from [SensorMSP1]:
{"name": "[The Anton Device]", "latitude": [13], "longitude": [4], "id": [44049609-c990-4a44-9fc0-87413896abb9]}
[IoTHubMonitor] [2:01:25 PM] Message received from [SensorMSP1]:
{"name": "[The Anton Device]", "latitude": [17], "longitude": [2], "id": [ad7dd161-80b6-403a-9a28-3971d5ff13d3]}
[IoTHubMonitor] [2:01:26 PM] Message received from [SensorMSP1]:
{"name": "[The Anton Device]", "latitude": [16], "longitude": [4], "id": [9d66ba4d-81be-4ade-a9cc-6a48018d66a8]}
[IoTHubMonitor] [2:01:26 PM] Message received from [SensorMSP1]:
{"name": "[The Anton Device]", "latitude": [12], "longitude": [2], "id": [cd20e435-37ee-4817-a68d-9fc3744fc0ab]}
[IoTHubMonitor] [2:01:26 PM] Message received from [SensorMSP1]:
{"name": "[The Anton Device]", "latitude": [16], "longitude": [4], "id": [9d66ba4d-81be-4ade-a9cc-6a48018d66a8]}
[IoTHubMonitor] [2:01:26 PM] Message received from [SensorMSP1]:
{"name": "[The Anton Device]", "latitude": [13], "longitude": [4], "id": [44049609-c990-4a44-9fc0-87413896abb9]}
[IoTHubMonitor] [2:01:26 PM] Message received from [SensorMSP1]:
{"name": "[The Anton Device]", "latitude": [17], "longitude": [2], "id": [ad7dd161-80b6-403a-9a28-3971d5ff13d3]}
[IoTHubMonitor] [2:01:26 PM] Message received from [SensorMSP1]:

```