Documentation

# Oppgaven

Oppgaven går ut på å lage en løsning som kan overvåke løringsmiljøet slik at man på sikt kan få bedre oversikt over skolen sin tilstand.

Video: <https://www.youtube.com/watch?v=nkPM_7XQKms>

# Valg av enhet

Løsningen er bygget opp med Particle Photon som tar i mot data og sender dem til Microsoft sitt skytjeneste. Grunnen til at det ble valgt Particle Photon over LoRa radio er fordi løsningen kan lett bli satt opp uten å måtte konfigurere radiofrekvens etter hvor man befinner seg. Dette betyr at løsningen kan enkelt settes opp i en skole i Kina uten å måtte gjøre noe ekstra spesial tilpasninger.

LoRa frekvenser

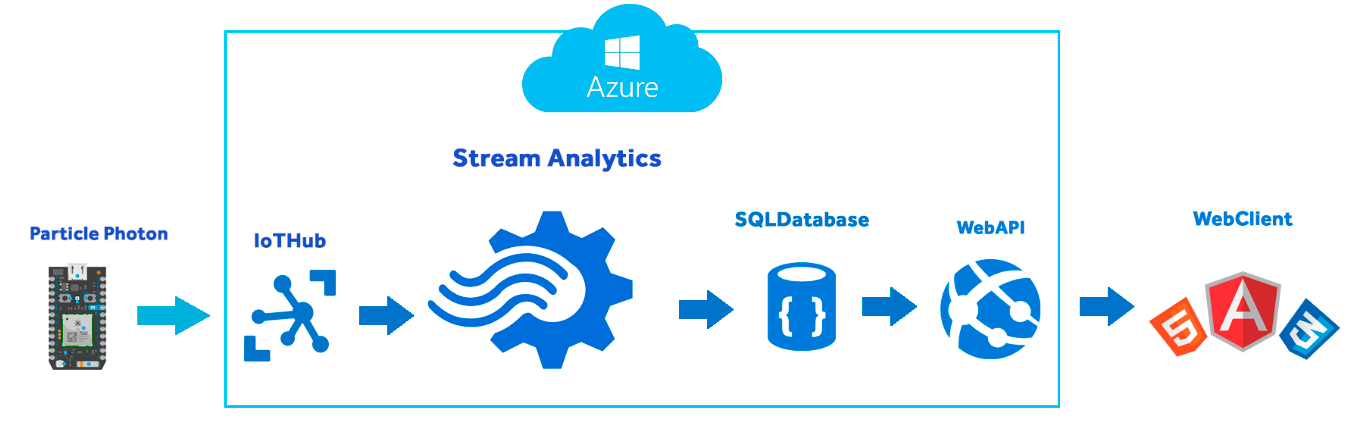
* EU: 863-870 Mhz
* AU: 915-928 Mhz
* US: 902-928 Mhz
* CN: 779-787 Mhz

I tillegg til dette må man planlegge hvordan man skal sette opp infrastrukturen til LoRa. Man må stille spørsmål som:

* Hvilken rekkevidde får vi innendørs?
* Klarer vi å nå helt ned i kjelleren?
* Hvor bør Gateaway-radion stå?

Med Particle Photon, som har tilgang til wifi slipper man å tenke på alt dette. Her trenger man bare å tenke på hvor i rommet man skal plassere sensor-noden. I tillegg til dette har Particle en veldig bra dokoumentasjon på hvordan man kan sende data over HTTP. Dette gjør det lettere å kunne koble seg til en skytjeneste (i dette tilfellet Microsoft Azure).

# Arkitektur



Her er en figur som illustrer hvordan dataen flyter fra enheten til nettsiden. IoT hub ligner veldig mye på EventHub, men den støtter 2-veis kommunikasjon over protokoller som MQTT eller AMQP. I tillegg til dette har Azure IoT hub «device management> der man kan registrere og forvalte alle sine enheter fra ett sted. MQTT gir for eksempel fordelen av å ha mindre «overhead» i protokollen, samt at den støtter callback funksjonalitet, dvs at du kan trigge metoder direkte på device. På den andre siden har Particle Cloud API også veldig god støtte for å snakke med enheten din på sin side. Les mer her om Azure IoT hub vs Azure Event Hub: <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-compare-event-hubs>

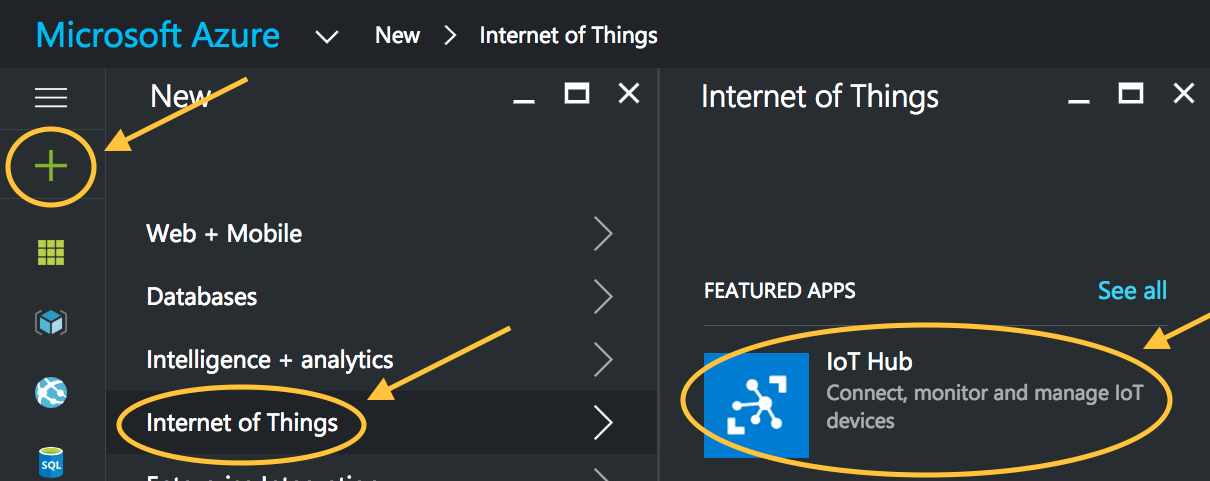
## Setup Azure IoT Hub

For å kunne sette opp Azure IoT Hub, trenger du først en microsoft konto(outlook.com). Deretter kan du registrere Azure kontoen din her: <https://azure.microsoft.com/nbno/free/?WT.srch=1&wt.mc_id=AID_SEM_>

Particle og Microsoft nå partnere, så dette gjør det mye enklere å kunne integrere enheten mot microsoft sine produkter. Dette gjør det mulig for oss å bruke andre tjenester i Azure som:

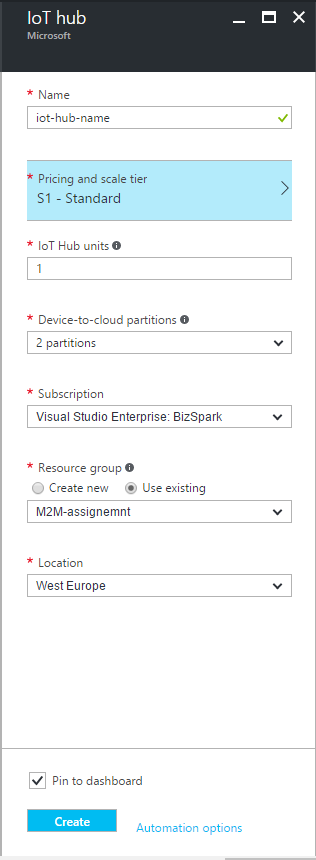
* Azure Machine Learning (Predikere sensordata?)
* Notification Hub
* Stream Aanalytics
* PowerBI

Gjennom hele prossessen I opprettelse av applikasjoner i Microsoft Azure kan det lønne seg å være tålmodig da det tar tid før ting er ferdig opprettet.

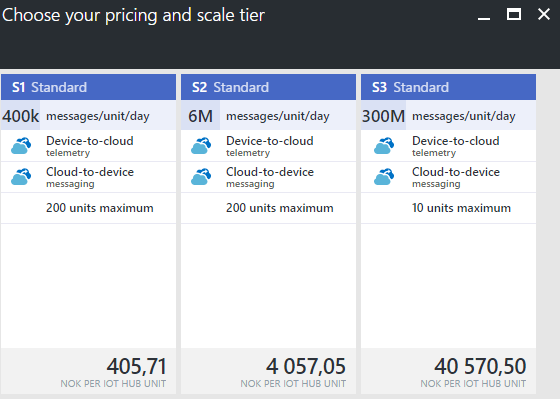
For å kunne sette opp Azure IoT Hub må du navigere deg til Azure portalen: [www.portal.azure.com](http://www.portal.azure.com) Trykk på **+** ikonen på sidebaren, deretter trykk på **Internet of Things > IoT Hub**  


(Hvis du har vanskeligheter med å finne den, kan du også søke på IoT Hub i søkebaren)

Deretter vil en ny fane poppe opp, bruk disse instillingene hvis du vil teste løsningen fort og eksperimentere uten at ting skal koste penger. Det vil ta litt tid før IoT Huben din er oppe og kjører.



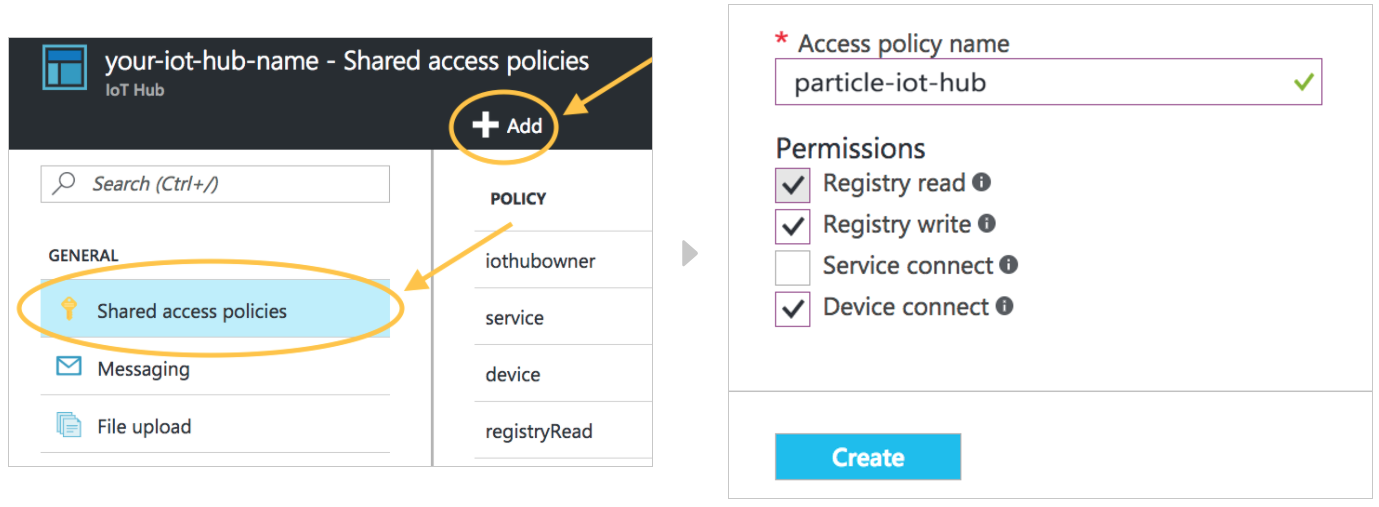
Hvis du velger Free-plan vil det ikke være mulig å oppgradere den til en av de andre tier i senere tidspunkt. Velger du Free vil det bare være for testing. Navnet til IoT-huben må du huske til senere for å kunne lykkes med integreringen. Bruk også resource group for å lett kunne gruppere dine applikasjoner under en gruppe. Dette gjør det lettere hvis du skal slette den til senere tidspunkt.



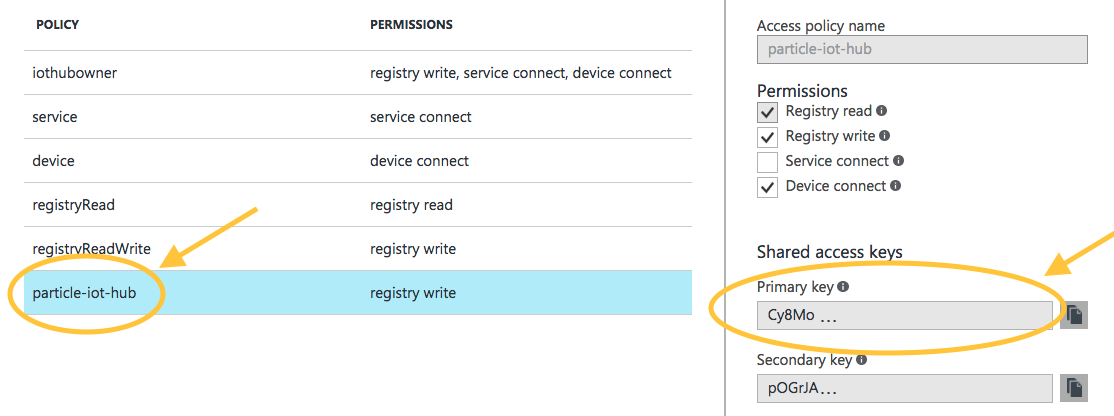
Det er ganske mye man kan gjøre for bare 405 kr i mnd (prisene endrer seg etter som valutakursene endrer seg). Dette avhenger selvfølgelig på hvor ofte du sender meldinger og hvor mange enheter som sender meldinger til IoT Huben.

**Legg til Shared Access Policy**

For å gjøre det mulig for Particle å snakke med IoT huben din, må du legge til en shared access policy. Dette er også på grunn av sikkerhetsmessige årsaker, med shared access policy lager du en regel som bare gjelder for din particle integrasjon. Når IoT Huben er ferdig spunnet opp, klikk på ikonet fra dashboardet, deretter klikk på **shared access policies** > **+ Add.** Gi din policy **write**, **read** og **device connect permissions.**



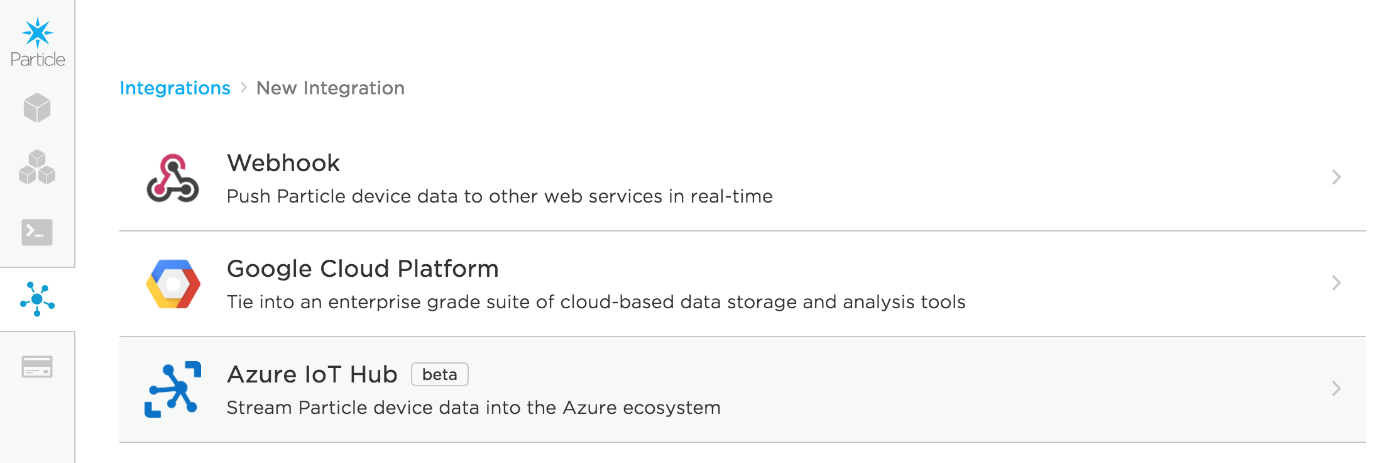
For at din particle enhet skal kunne ha vellykket integrasjon trenger du begge **policy name** og **policy primary key**. Du kan finne dette ved å klikke på policy navnet som du nettop har opprettet.



Alle nøkkler fra applikasjonene i Azure har ending med « = », hvis du er usikker på at du har kopiert hele koden, kan du se på slutten av strengen om den har « = ». Ellers kan du klikke på ikonet for å kopiere nøkkelen. Når dette er ferdig kan vi begynne å konfigurere instillingene i Particle.

## Particle Console

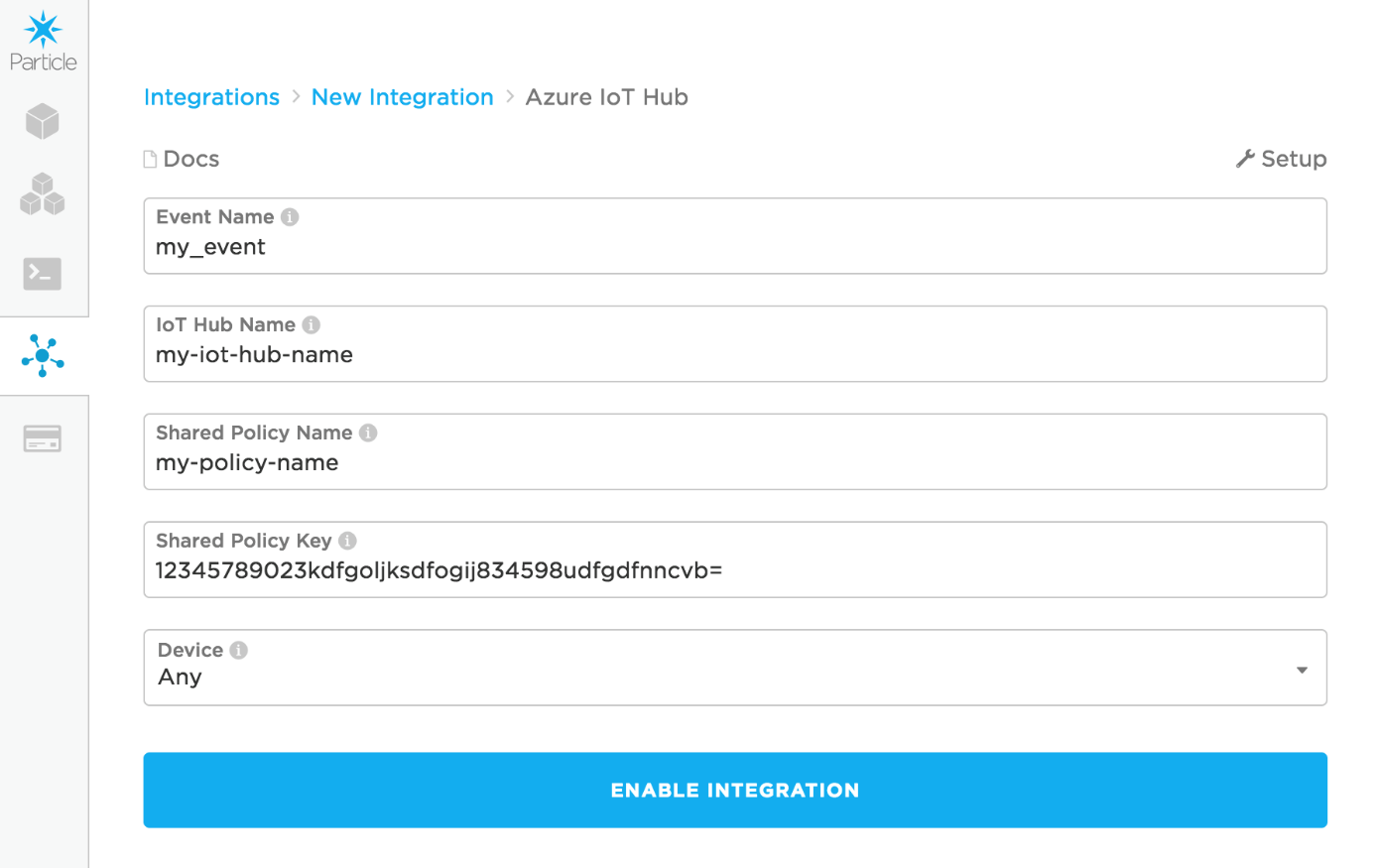
Naviger deg inn til [www.console.particle.io](http://www.console.particle.io) for å aktivere **Azure IoT Hub Integration.**



(Den er fortsatt under beta, så noe kan være litt ustabilt)

Videre i konfigurasjonen vil du støte på disse feltene:

* **Event Name:** Dette er navnet på eventet som vil trigge datapublisering til Azure IoT Hub. Dette vil bli trigget når du kaller på Particle.publish() metoden fra Particle.
* **IoT Hub Name:** Navnet som du ga IoT Huben. I dette tilfellet: *iot-hub-name*
* **Shared Access Policy Name:** Navnet til Shared Access Policy. I dette tilfellet: particle-iot-hub
* **Shared Access Policy Key:** Nøkkelen fra shared access policy name
* **Device:** Hvis du vil at dette skal gjelde alle particle enheter, velg: “Any”, hvis du vil at det bare skal gjelde din enhet må du velge enheten med respective navn.



Videre klikker du på Advanced settings for å kunne sende din spesial tilpasset JSON. Hvordan webhook variablene fungere kan du lese mer om her: <https://docs.particle.io/guide/tools-and-features/webhooks/#webhook-variables>

Kode:

{

"carbondioxide": "{{C2}}",

"temperature": "{{t}}",

"humidity": "{{h}}",

"light": "{{l}}",

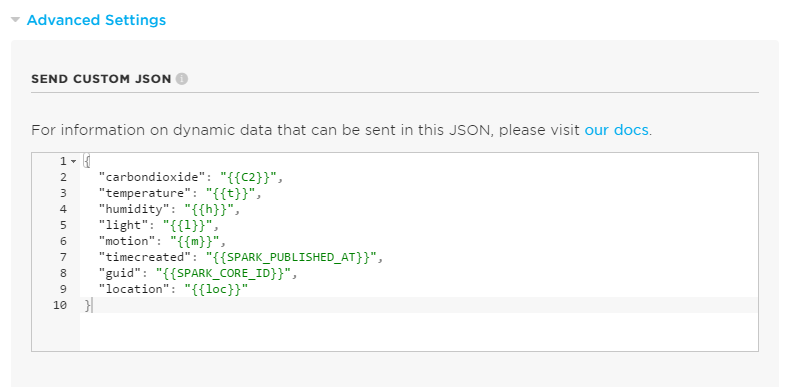
"motion": "{{m}}",

"timecreated": "{{SPARK\_PUBLISHED\_AT}}",

"guid": "{{SPARK\_CORE\_ID}}",

"location": "{{loc}}"

}



# Particle Photon

Utstyrsliste for Particle Photon:

* MH-Z19
* DHT11
* Photoresistor
* PIR

Før du i det heletatt begynner å koble opp hele systemet og laste opp kode, kan det lønne seg å bare koble hvert komponent hver for seg og teste hver for seg. Dette kan du f.eks gjøre ved å teste gjennom *analogRead(PIN).*

Set up Particle Photon: <https://docs.particle.io/guide/getting-started/start/core/>

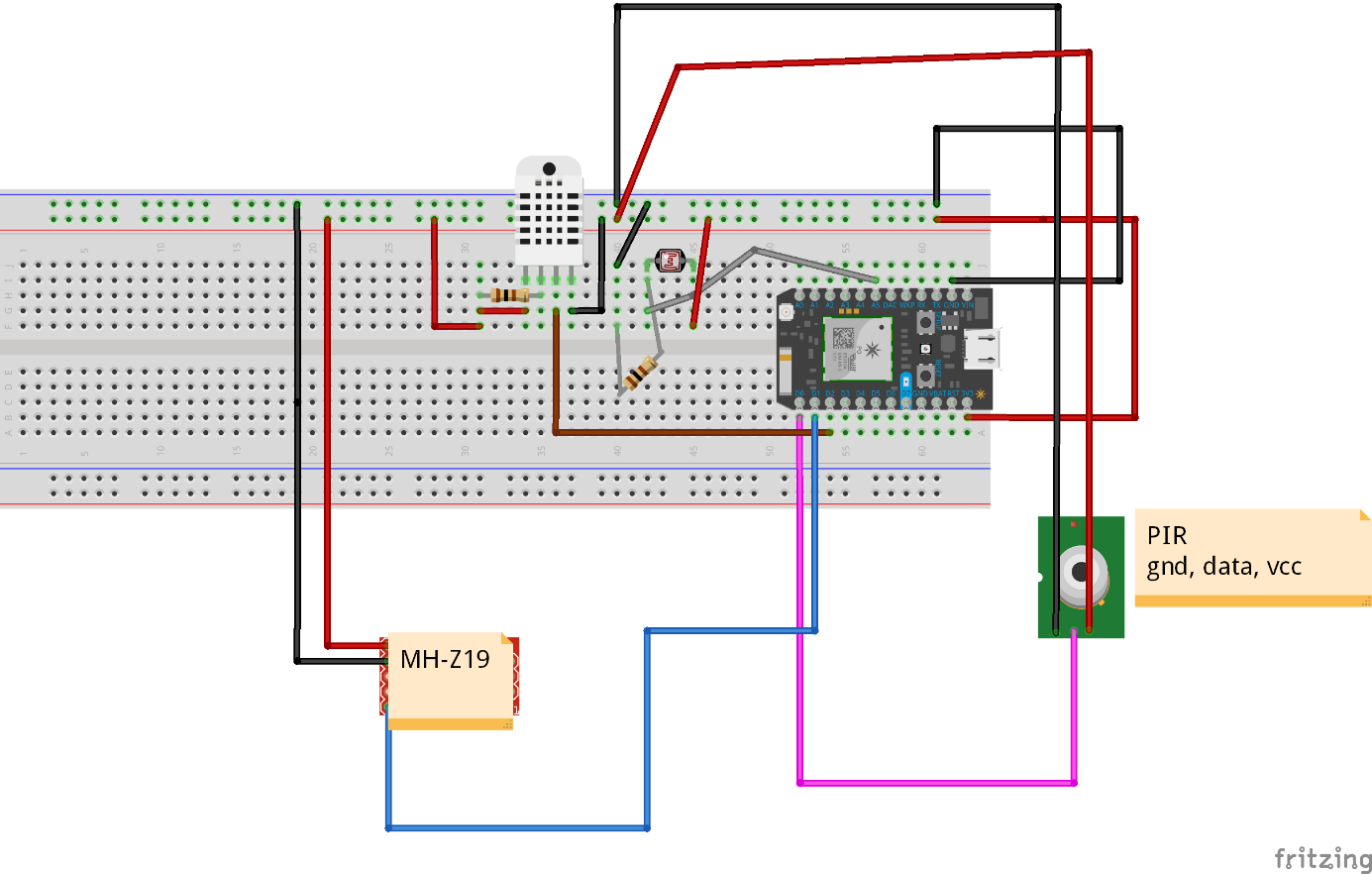
## Koblingsskjema

|  |  |  |
| --- | --- | --- |
| PIR | Particle Photon | Data |
| VCC | VCC | Bevegelse |
| Data | D0 |
| GND | GND |

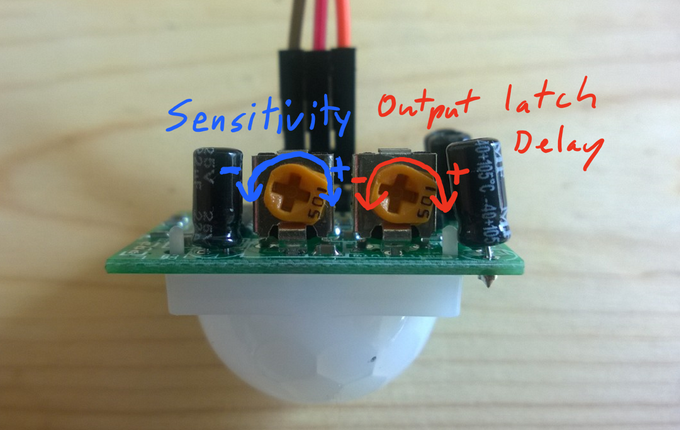
|  |  |  |
| --- | --- | --- |
| DHT11 | Particle Photon | Data |
| VCC | VCC | Temperatur: Celcius & Farenheit Fuktighet |
| GND | GND |
| D-OUT | D2 |
| A-OUT | Ingen |

|  |  |  |
| --- | --- | --- |
| Photosensor | Particle Photon | Data |
| A-OUT | A5 | Lys |
| GND | GND |

|  |  |  |
| --- | --- | --- |
| MH-Z19 | Particle Photon | Data |
| PWM | D1 | CO2 (Carbondioksid) |
| GND | GND |
| VCC | VCC |



Jeg fant ikke delen for MH-z19 i fritzing, men det holder nok å forholde seg til koblingsskjema også. Vær oppmerksom på at PIR-sensoren kan oppføre seag litt merkelig i starten. Da hjelper det å justere Sensetivity og delay.



## Kode

Til dette prosjektet lastet jeg ned local IDE for å kunne jobbe med filer. Fordelen med dette er når man har så mange sensorer som nå, blir det lettere å lage objekter av sensorene. IDE kan lastes ned her: <https://www.particle.io/products/development-tools/particle-local-ide>

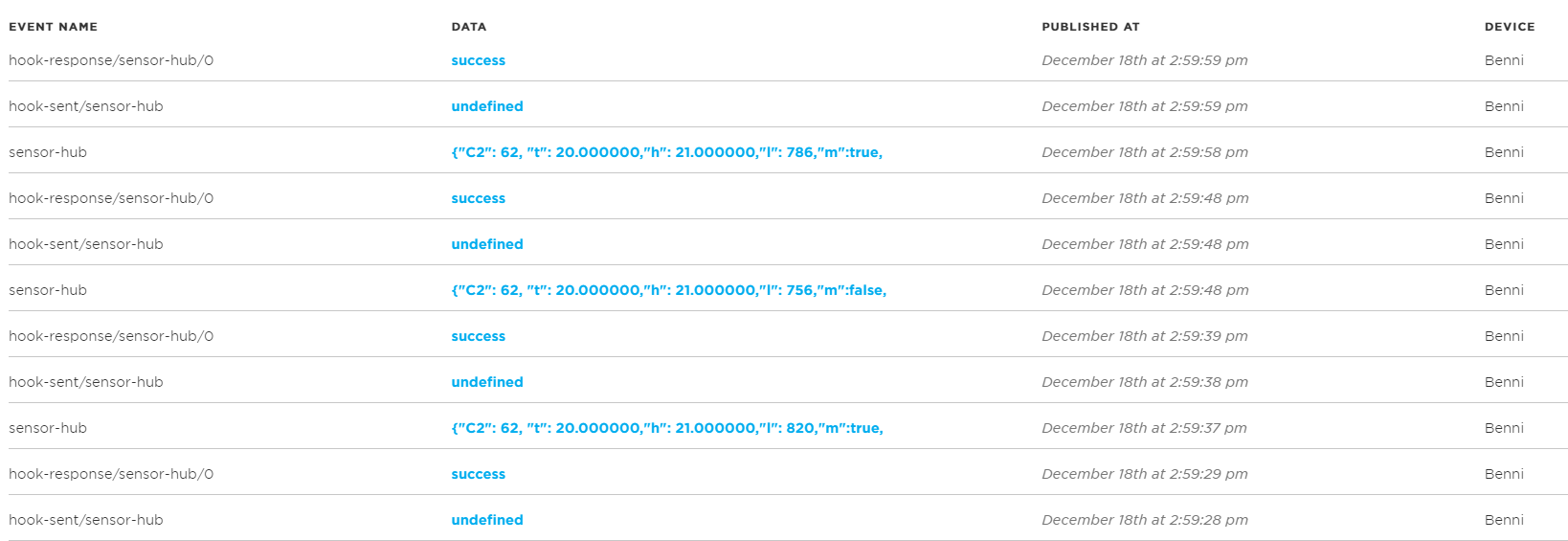
Ulempen med IDE-et er at man ikke kan inkludere biblioteker på like lett måte som online IDE-et. Dette må du selv laste ned og legge til i prosjektet. En løsning på dette (som jeg gjorde) var å gå inn på online IDE, finne det biblioteket jeg ville ha og replikere filene i local IDE.

Deretter, for at alle .h filer skal fungere må du inkludere #include "application.h" i respektive .filer. Dette er for å kunne bruke Particle sitt bibliotek.

Det er også mulig å bruke [www.build.particle.io](http://www.build.particle.io) men da må du være oppmerksom på at oppdelingen av sensorene ikke vil fungere. Du må legge alle koden i en fil.

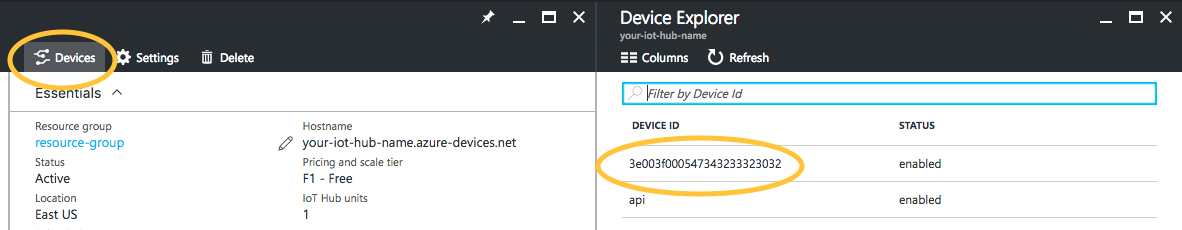
## Test om data blir sendt

Det finnes flere måter å se om dataen blir sendt. Fra particle sin side holder det å sjekke ut consoll loggen fra nettleseren eller ved å kjøre particle subscribe mine fra CLI. Jeg har forholdt meg til consollen nettleseren.

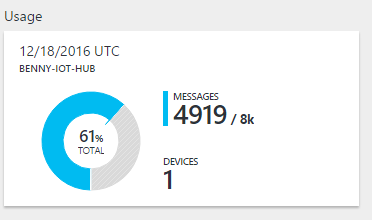


Fra Azure sin side kan det lønne seg å sjekke ut iothub-explorer som er en npm-pakke. <https://github.com/Azure/iothub-explorer>

Når alt er på plass og meldingene blir sendt uten error vil particle enheten din bli registrert i IoT Huben.



Du vil også kunne se om kvoten din per dag blir brukt opp eller ikke.



## Stream Analytics

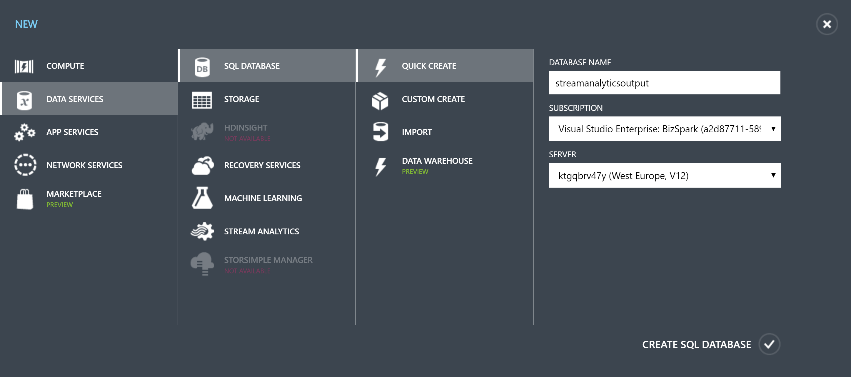
På grunn av tid og erfaring vil denne delen skje gjennom den gamle portalen. Det er ingenting i veien å gjøre dette gjennom den nye portalen, men forrige oppgave var jeg avhengig av å gjøre alt i gamle portalen på grunn av støtte for Service bus og eventhub. Ingenting i den denne løsningen er egentlig avhengig av den gamle portalen.

Naviger til [www.manage.windowsazure.com](http://www.manage.windowsazure.com)

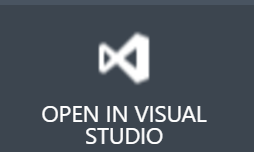
Stream Analytics er det verktøyet vi vil bruke for å kunne sveise sammen data fra enheten og skrive til en database. Den fungerer som en publisher og en subscriber. I dette tilfellet vil den ta inputstream fra IoT Hub og deretter output til en SQL-Database. Meldingene som blir sendt til IoT Hub vil bli slettet over tid og derfor trenger vi SQL-Database til å kunne lagre historiske data.

For å opprette database trykker du på **SQL Databases** og trykk på new

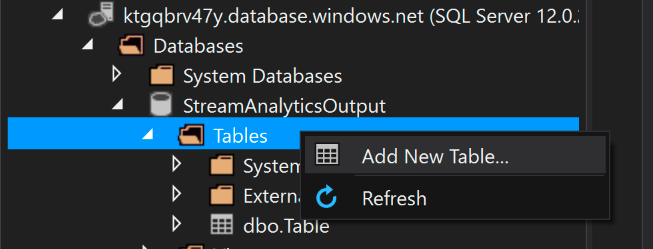




Når databasen er ferdig opprettet må vi legge til tabeller og attributter. For å gjøre dette lett, klikk inn på **Open in Visual Studio**



Let etter SQL serveren som databasen ligger i og opprett ny tabell.



Det går ann å bruke Databasedesign vinduet til å legge til i databasen, men lim inn dette for å spare tid.

CREATE TABLE [dbo].[Table] (

[Id] INT IDENTITY (1, 1) NOT NULL,

[temperature] FLOAT (53) NULL,

[humidity] FLOAT (53) NULL,

[light] INT NULL,

[motion] NVARCHAR (MAX) NULL,

[timecreated] DATETIME NULL,

[guid] NVARCHAR (MAX) NOT NULL,

[carbondioxide] INT NULL,

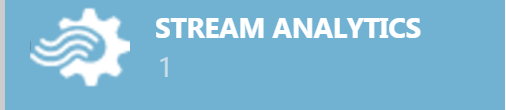
[location] NVARCHAR (MAX) NULL,

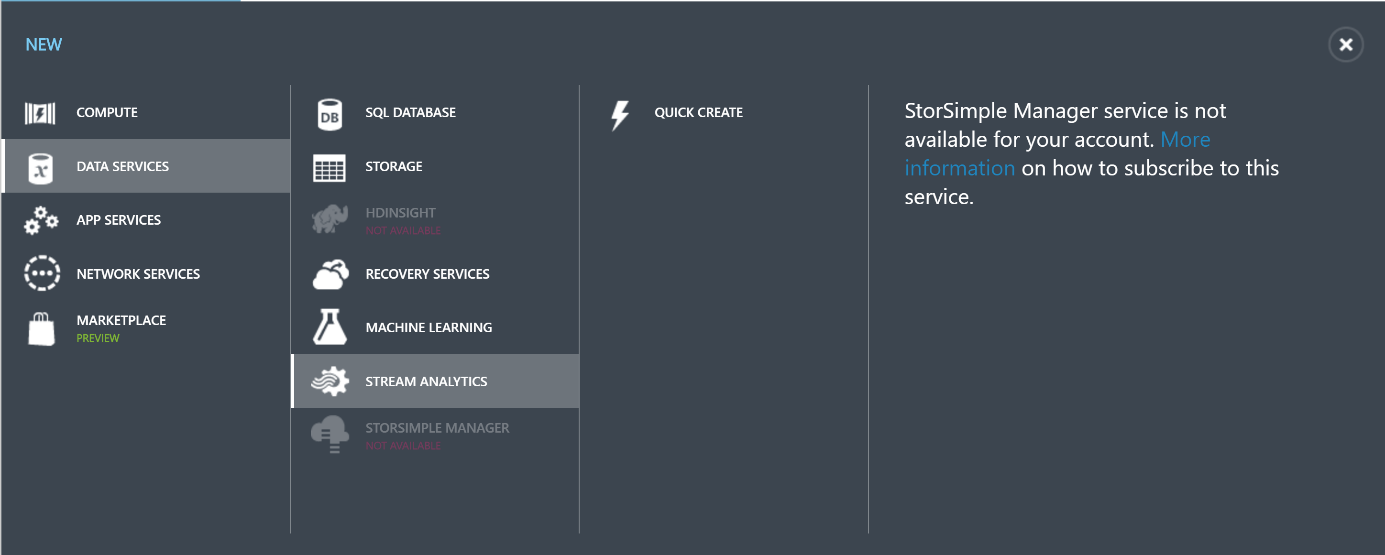
PRIMARY KEY CLUSTERED ([Id] ASC)

);

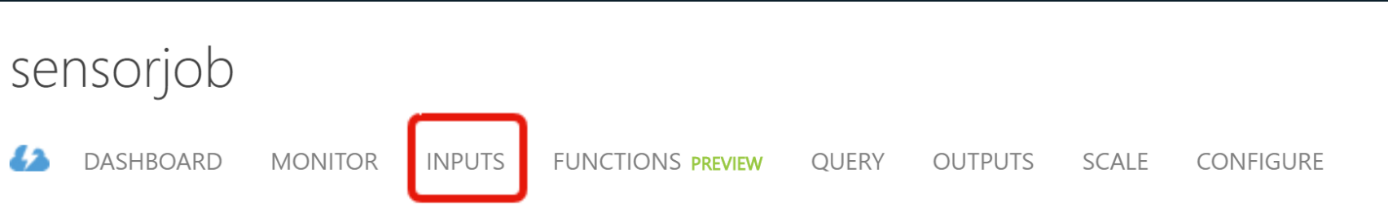
**Opprette Stream Analytics**

På samme måte som database leter du etter en lignende ikon bare for stream analytics





Vi vil først gå inn på inputs for å si til stream analytics hvor datakildene våres kommer i fra. Dette finner du i topmenyen



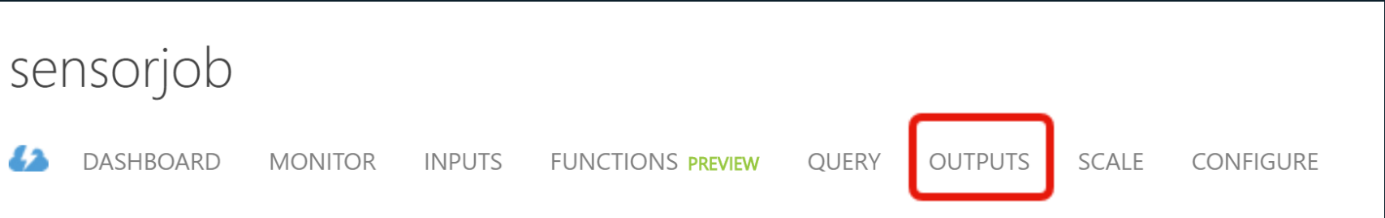
Trykk deretter på add input som befinner seg i footeren. 

## Input

Step 1. Velg data stream

Step 2. Velg IoT Hub

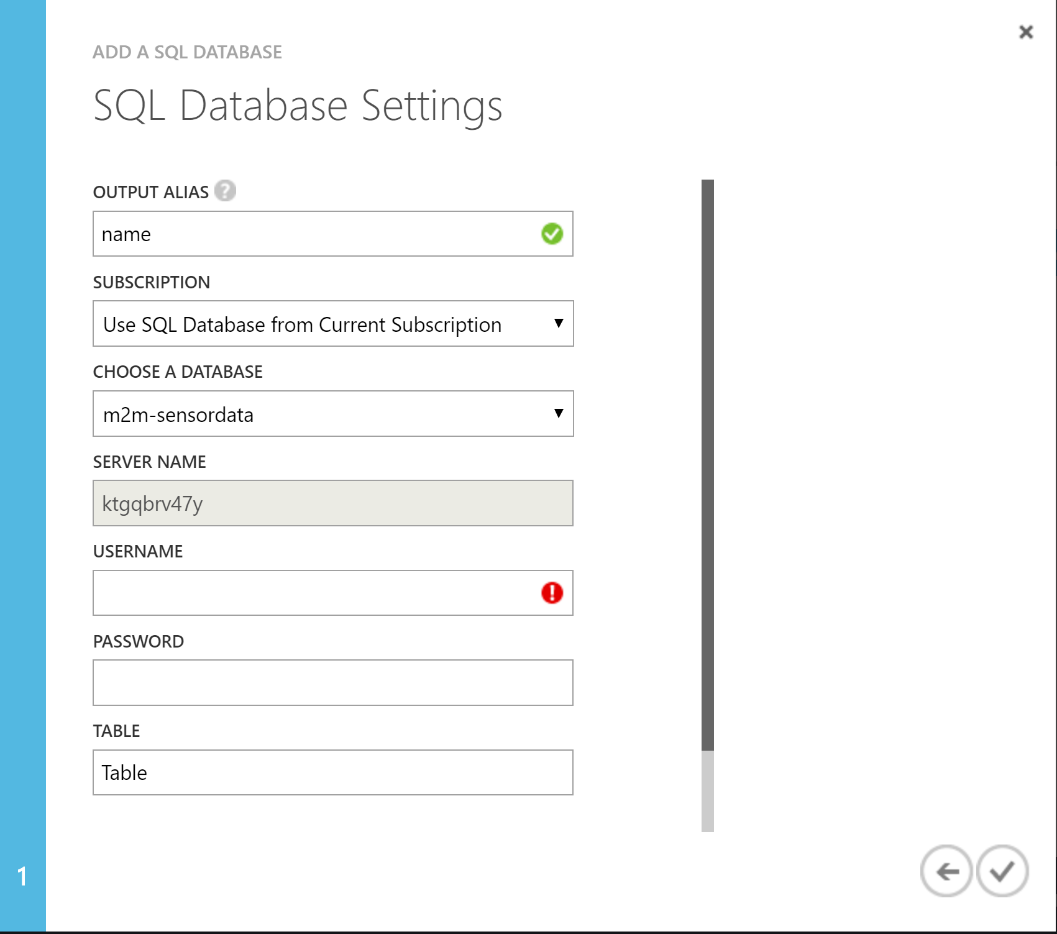
## Output



Klikk på Add output og deretter velg SQL-Database



Fyll inn feltene med det du har configurert databasen din med. Her er det viktig at du referer til samme tabell. I mitt tilfelle kalte jeg bare min for Table.



Videre går vi til Query som er i samme meny som Input og Output. Her vil vi kunne kjøre spørringer slik at det som kommer fra input datastrømmen vil bli lagt inn i output som er vår database. Legg så inn dette scriptet i vinduet

SELECT

temperature, humidity, light, motion, timecreated, guid, carbondioxide, location

INTO

StreamAnalyticsOutput

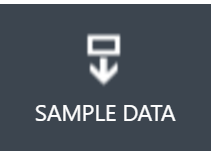
from

iotsensorhub;

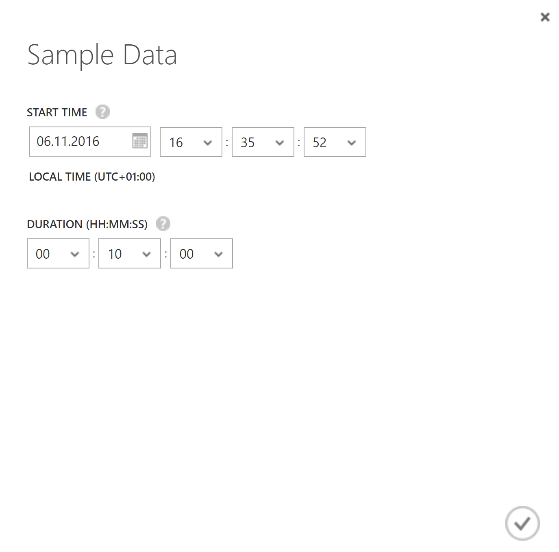
## Test runs

Før du spinner opp jobben kan det lønne seg å kjøre test på dataene. Dette kan spare deg for masse tid da oppstart av jobben tar lang tid og aggregeringen av dataene tar lang tid. Jeg brant meg veldig på dette under prosjektet da webhooken min hadde feil i JSON, men skjønte ikke hva som var problemet til jeg testet dataene.

Gå først inn på input for å laste ned sample data.



Her kan du velge hva slags intervall dataene dine skal komme fra. Hvis det var en tidsperiode der din particle ikke fungerte, kan du lett filtrere bort dette.

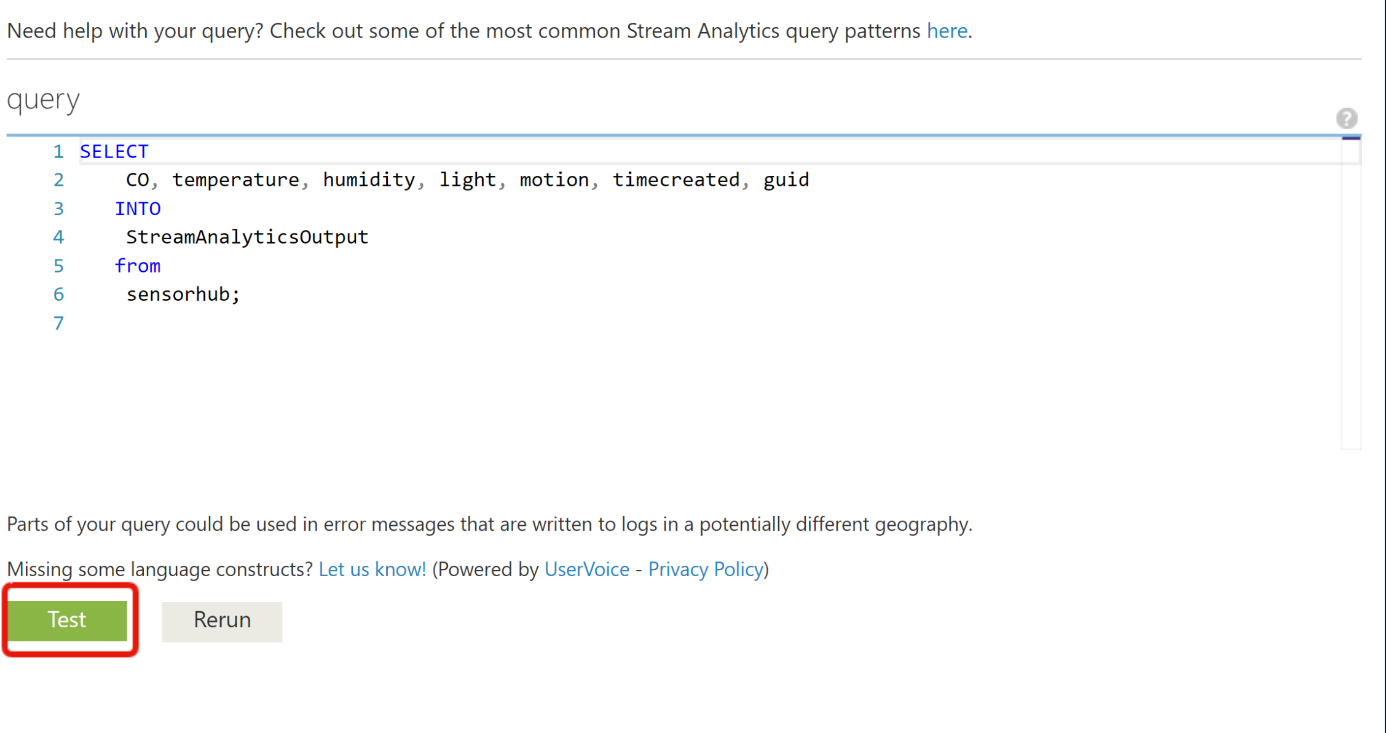


Når den er ferdig lastet trykk på details for å laste ned dataene.

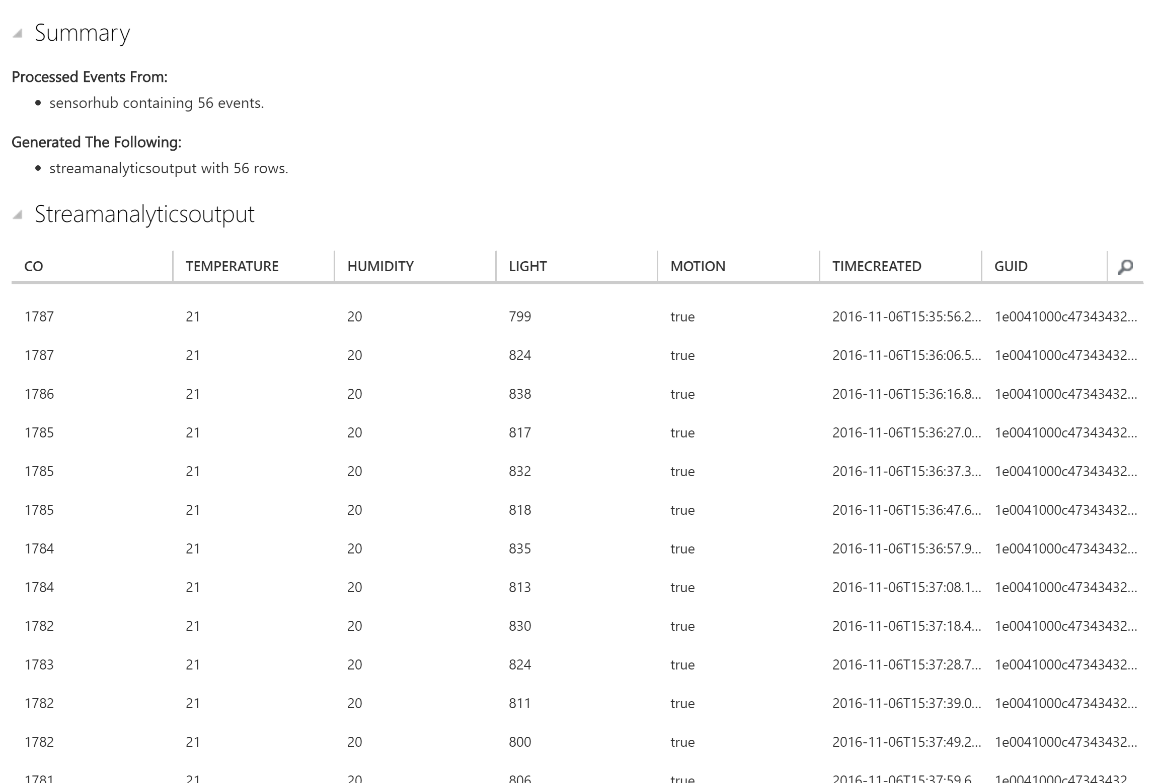


Bruk: <https://jsonformatter.curiousconcept.com> til å formatere JSON slik at det blir lettere å se dataene. Dette gjør du ved å åpne JSON filen, klipper ut innholdet og limer den inn i vinduet. Hvis alt her viser seg å være fint, betyr det at alle dataene som blir sendt fra din particle photon er safe og det er ikke noe feil der som må rettes opp.

Naviger deg videre inn til query og trykk på Test knappen.



Last opp JSON filen du fikk fra input. Da vil du også kunne få ut hvordan det vil se ut i databasen.

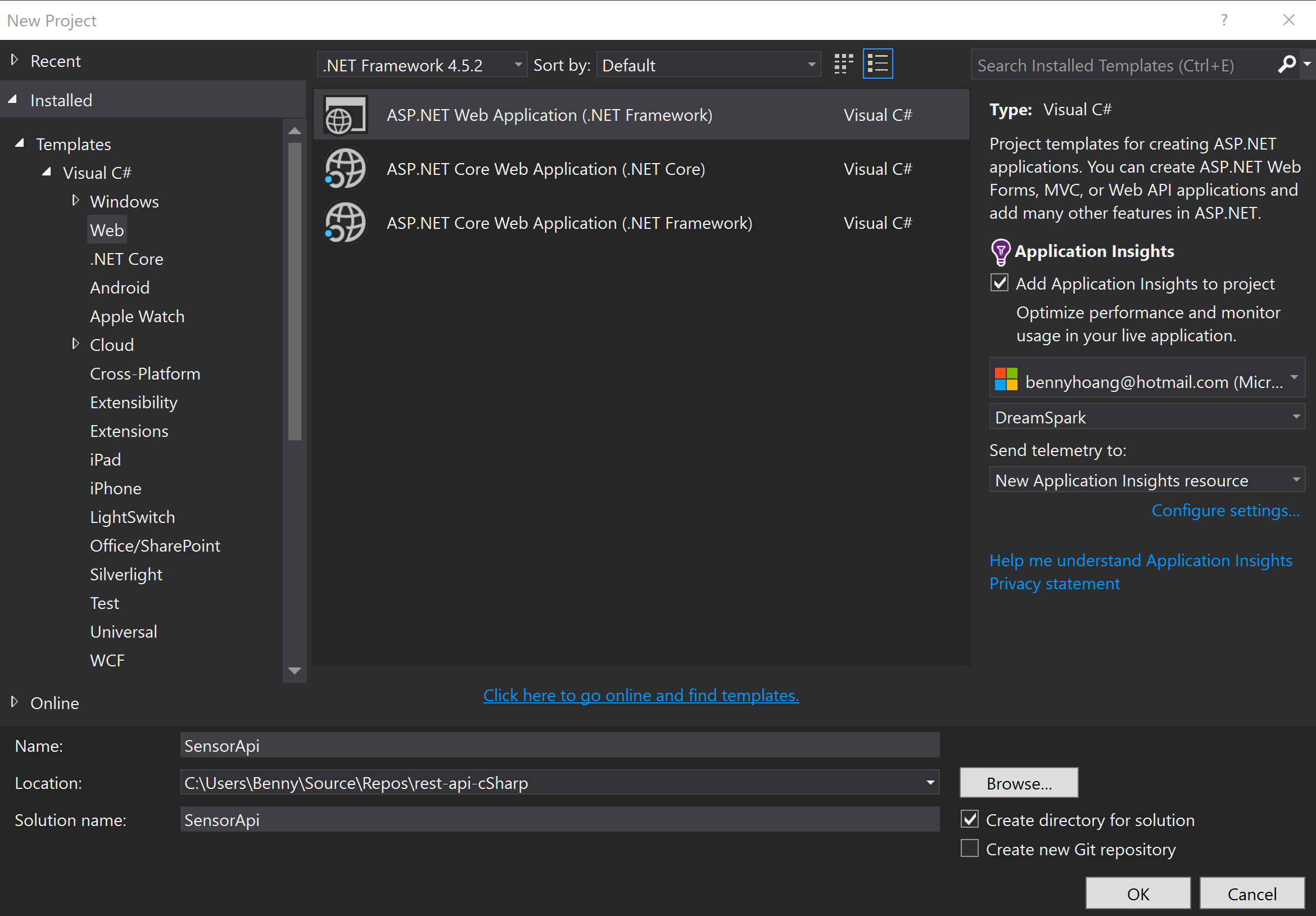


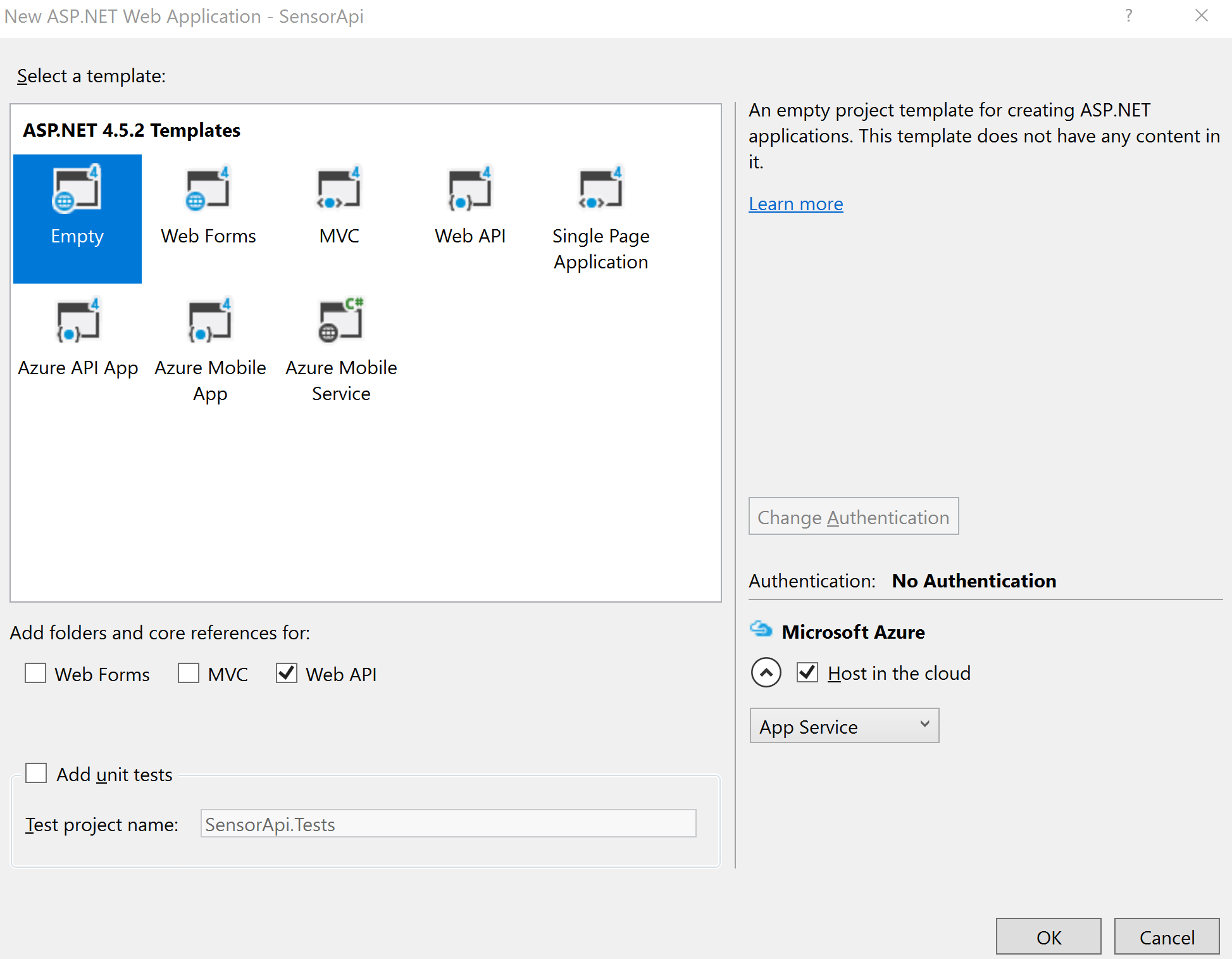
Hvis alt ser ut som det skal fungere, kan du bare starte jobben og samle inn sensor data.

# Web Api i C#

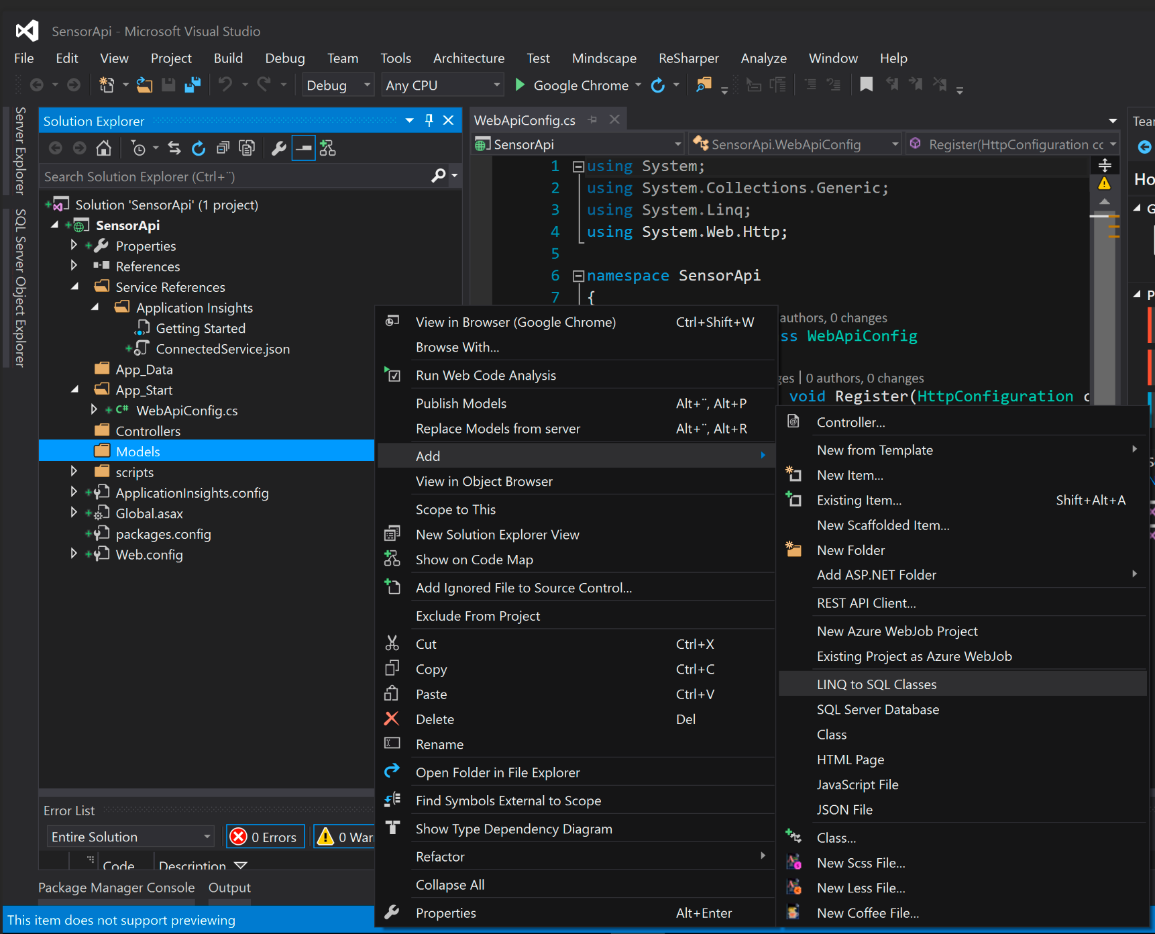
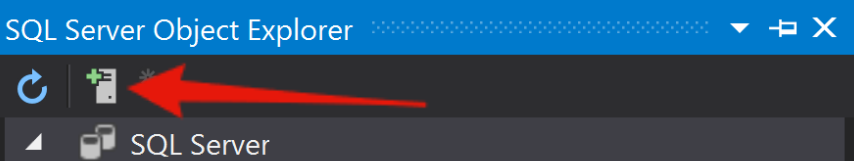
## Oppsett av prosjektet

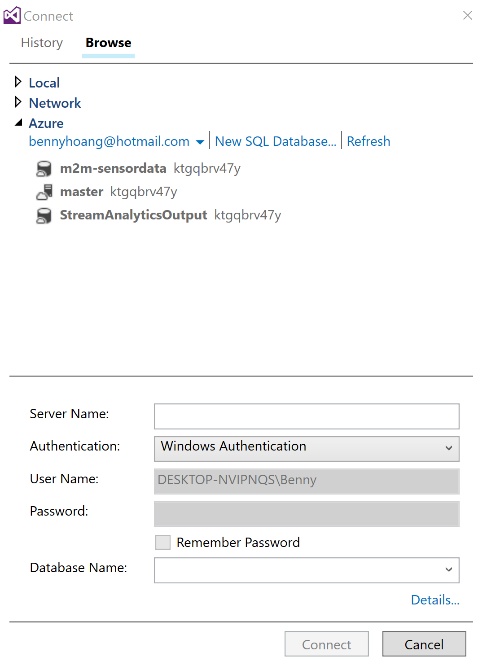
* Start med å lage ett nytt prosjekt ved å gå inn på **file** > **new** > **Project ...**
* Velg **ASP.Net Web application (.NET framework)**



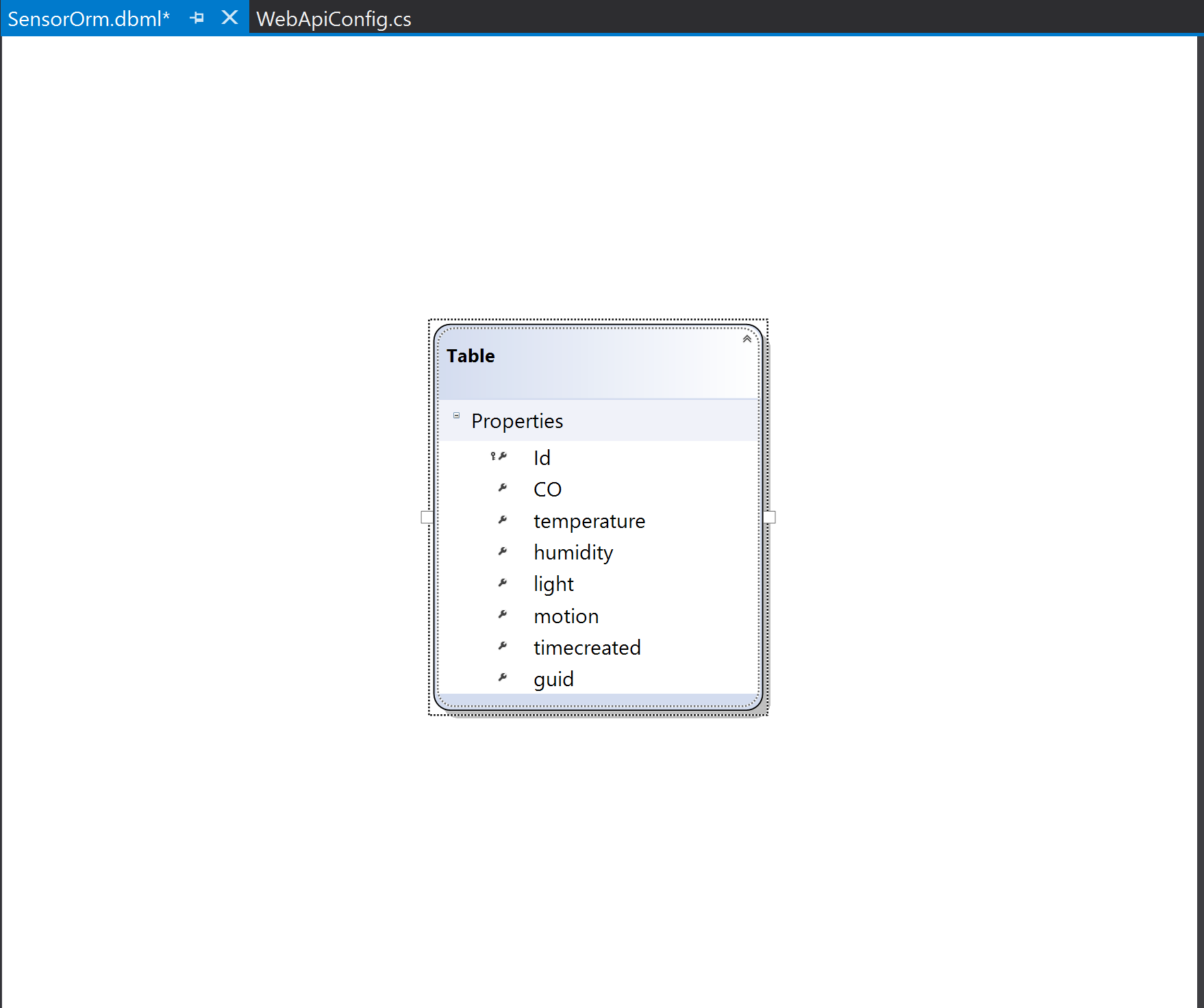
* Velg **Empty** og huk av på **Web API**. Det er også mulig å publisere api-et til azure hvis det er ønsket. 

## Lag Modell

* Legg til LINQ to SQL Classer ved å høyreklikke på Models. 
* Gå inn på **SQL Server Object Explorer** i venstre tab og velg din SQL server fra Azure. Hvis det ikke er allerede opprettet sql server kan du klikke på add **sql server** ikonet. 
* Du vil deretter få dialogboks om hvilken server du skal koble deg opp mot. Velg riktig database og deretter fyll inn feltene med riktig info.

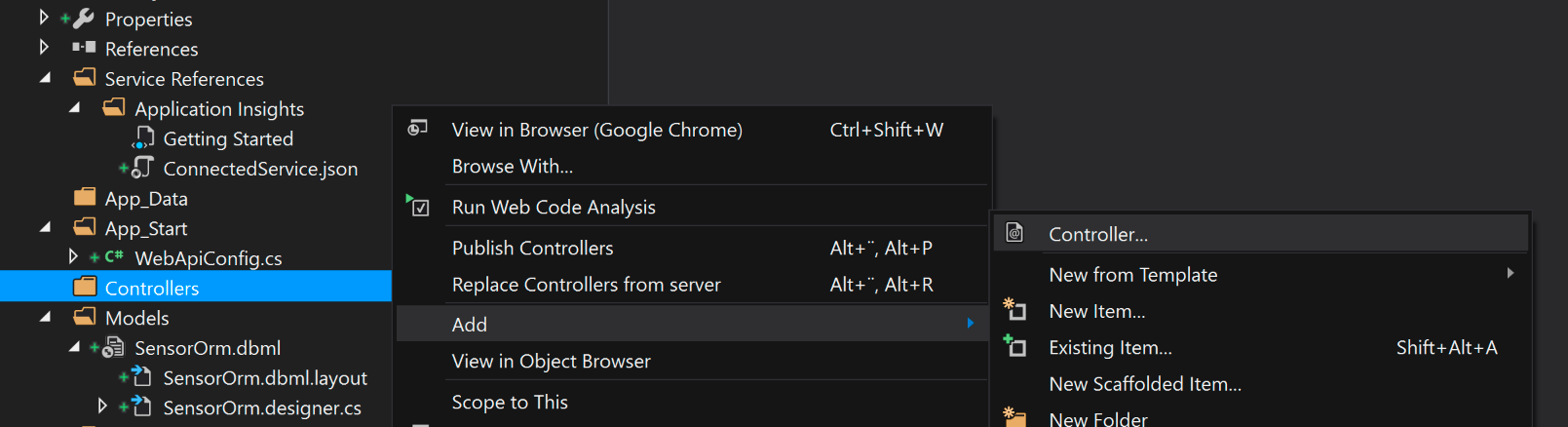
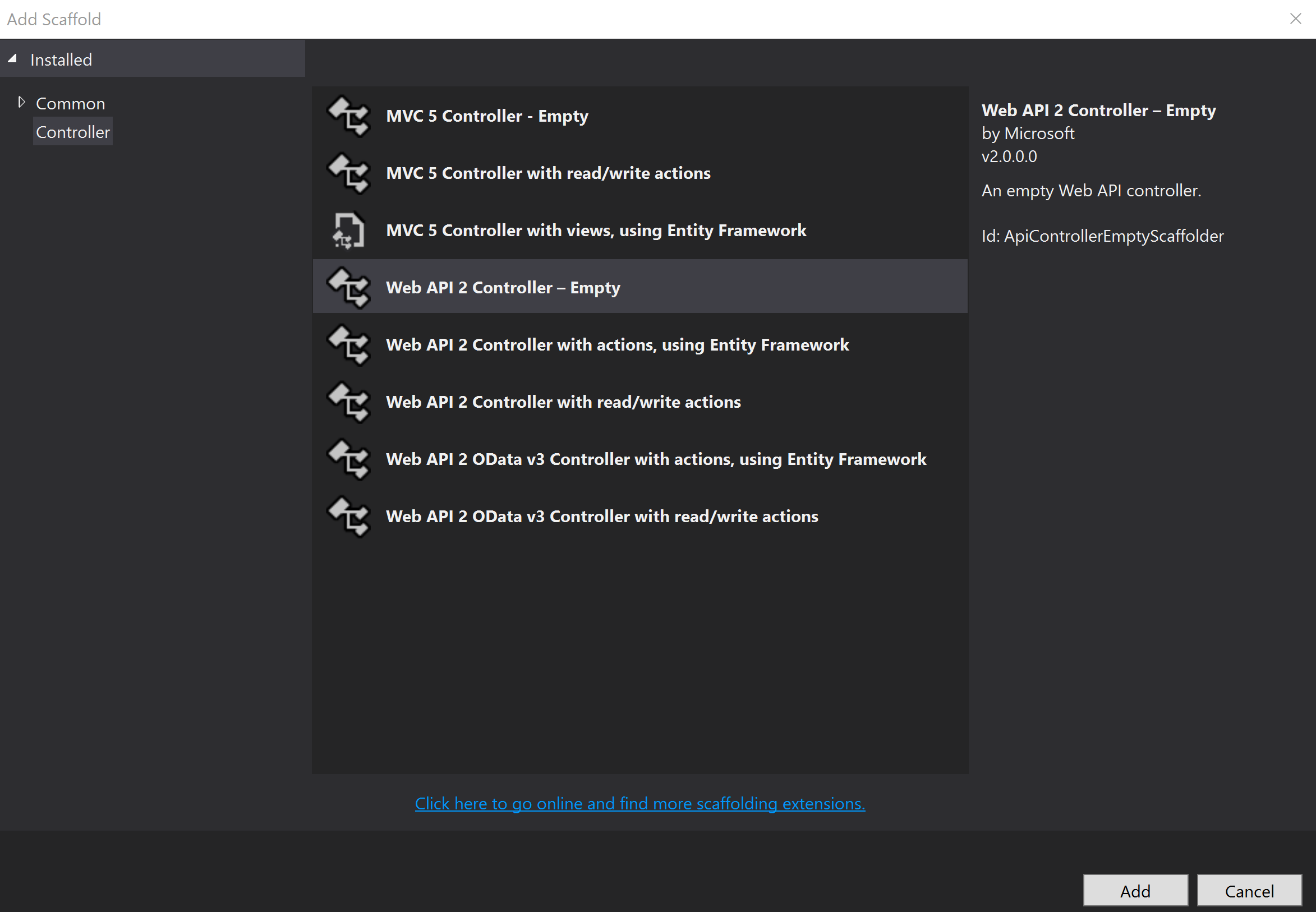


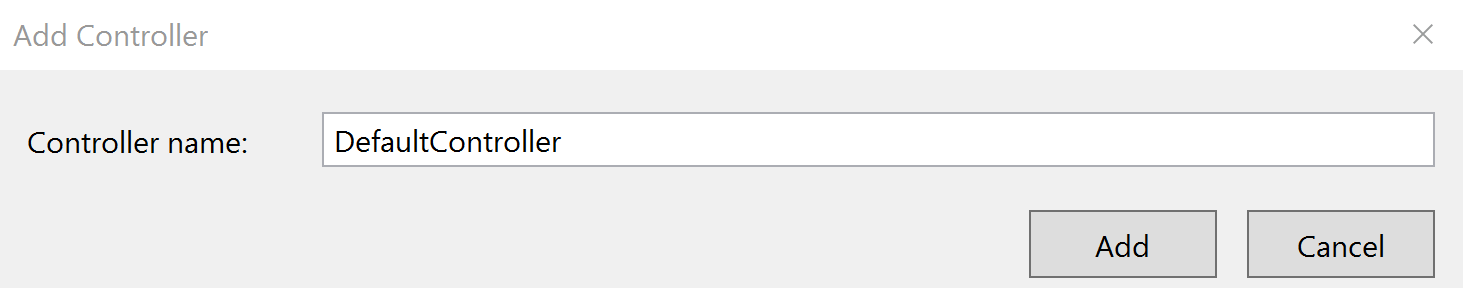
* Etter å ha åpnet sql serveren din kan du finne tabellen hvor sensordataene ligger. Deretter dra hele tabellen inn i det hvite vinduet i dbml filen. I mitt tilfelle heter tabellen min **dbo.Table**



For å gjøre det lettere i kode gir jeg tabellen et mer logisk og spesefikt navn slik at det blir lettere å holde tungen rett i munnen. Dette gjør du ved å dobbelklikke på tittelen, deretter lagre filen. På denne måten lager vi et objekt som representerer data i web-apiet. ASP.NET Web API kan automatisk serialisere modellen din til formater som **JSON** eller **XML.**

## Lag Controller

* Høyreklikk på **Controller** > **Add** > **Controller...** 
* I **Scaffold** dialogboksen velg **Web API 2 Controller – Empty**
* Gi deretter logisk navn til kontrolleren. Det er viktig å ha med prefixen «**Controller**»på slutten av kontrolleren din slik at applikasjonen kan vite at dette er en kontroller. Navnet må også følge PascalCase for at det skal fungere.



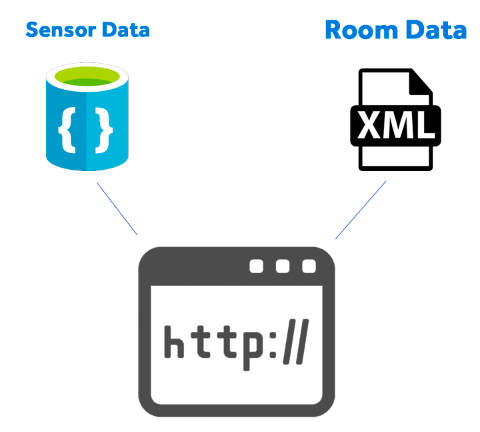
* Vi lager en metode som henter alt fra databasen og populerer den til en liste. For at vi skal kunne få tilgang til modellene som er opprettet må du inkludere modellene. Dette gjør du ved å skrive:

using <*NAMESPACE*>.Models;

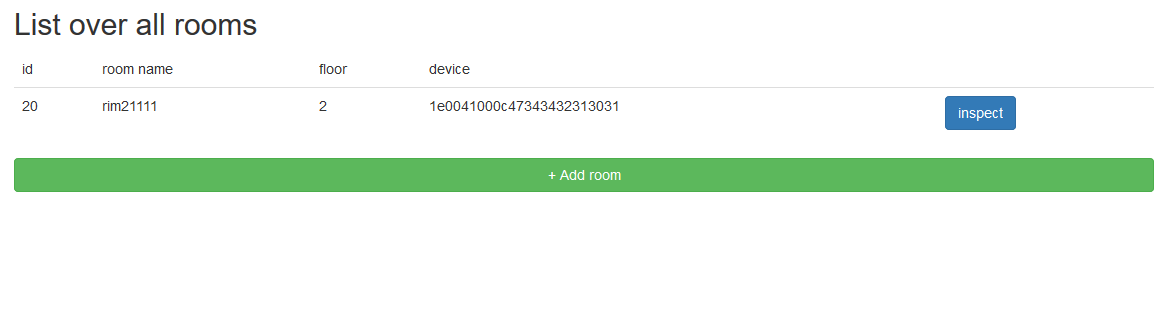
Namespace er det prosjektet ditt heter. I mitt tilfelle skriver jeg **using SensorApi.Models;**Apiet ligger publisert her: <http://sensorapi2030394.azurewebsites.net/api/sensor/getallsensordata>

# Frontend

Jeg skal ikke gå i dybden om hvordan man legger opp koden her da dette er mer fritt fram hvordan man har tenkt til å visualisere data osv.



Løsningen er laget for driftansvarlige. Her er det mulighet til å lage nye rom og refere sensor til respektive rom. Valg av XML som database vil ikke være skalerbar, men med tanke på prototyping har jeg valgt å bare gå for xml for å demonstrere hvordan brukerflyten er.

List:  


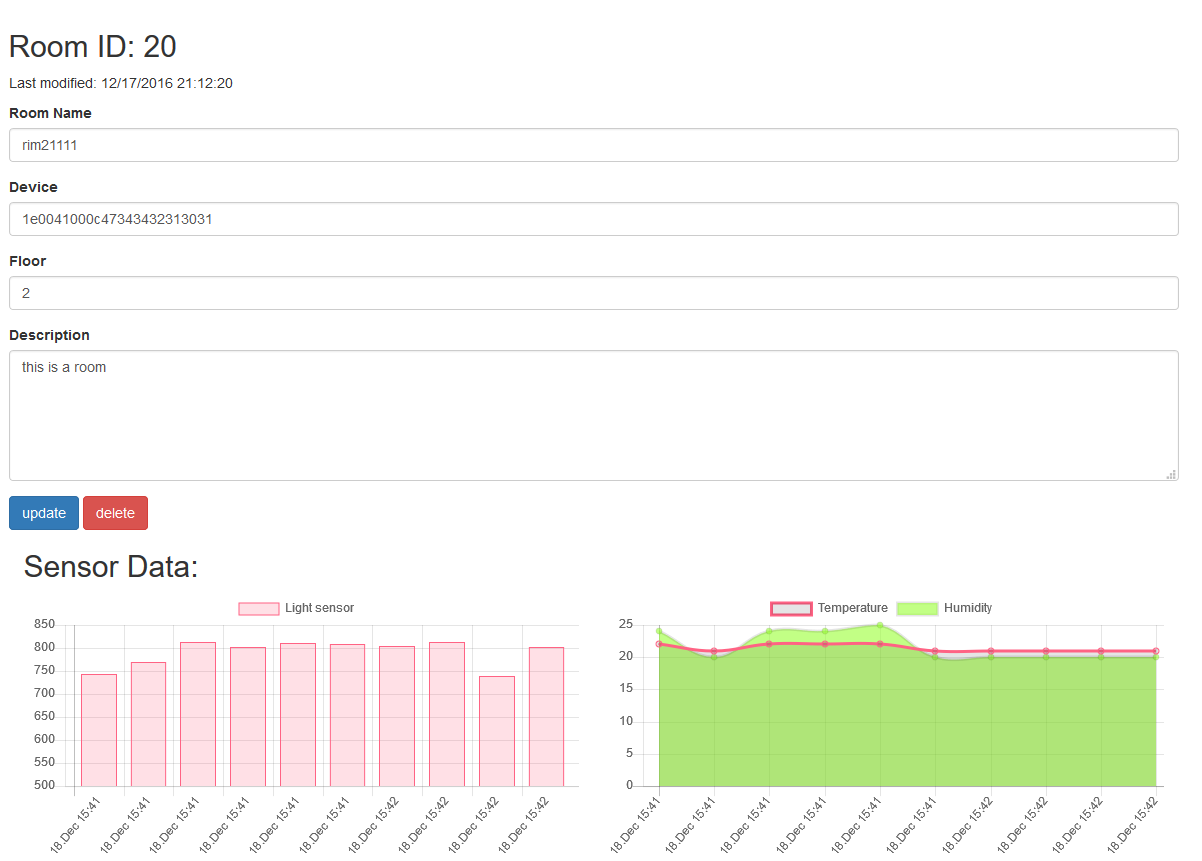
Her er tanken at vi kan ha oversikt over alle rom, for å så gå inn og lese av status på rommet.

Add room:

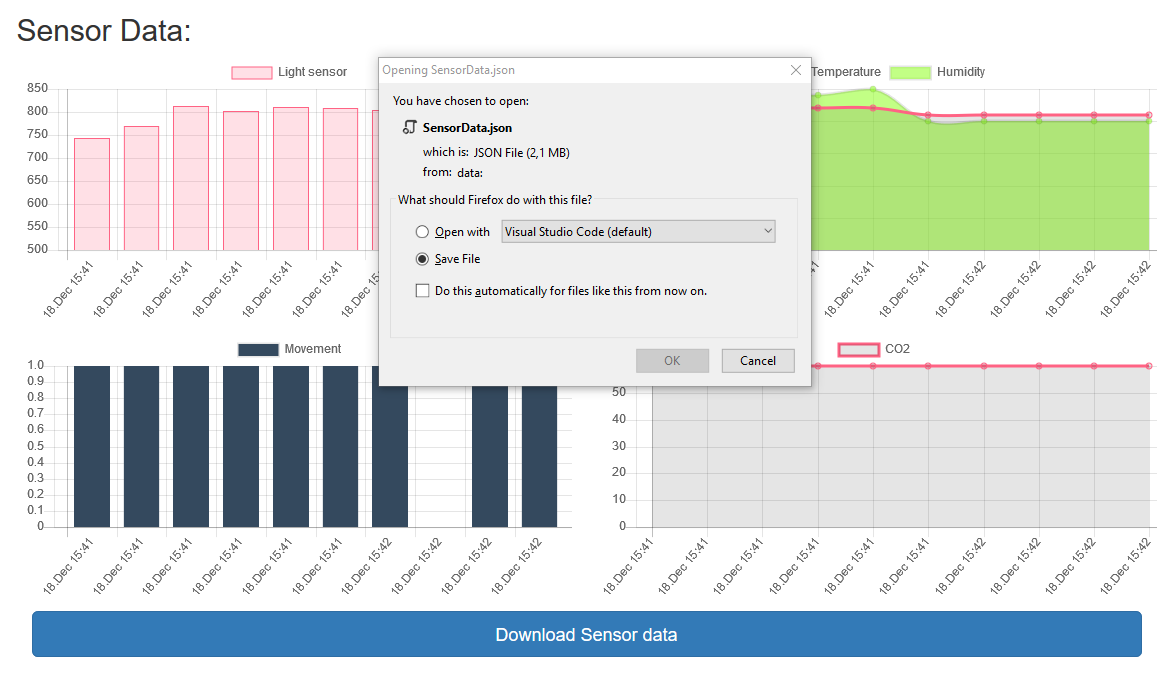


Når man legger til rom er det tenkt at man også kan velge hvilken enhet som skal bli lagt til i rommet.

Room:



Når rommet er opprettet vil man kunne gå og se på status på rommet. På grunn av at jeg bare har en sensor til rådighet kan jeg bare legge til ett rom og få en type sett med data. Jeg har filtrert ut dataene her på Device ID, så hensikten her er å kunne se på data fra spesefikk device. Når man legger til ny rom med annen sensor vil grafene se annerledes ut.



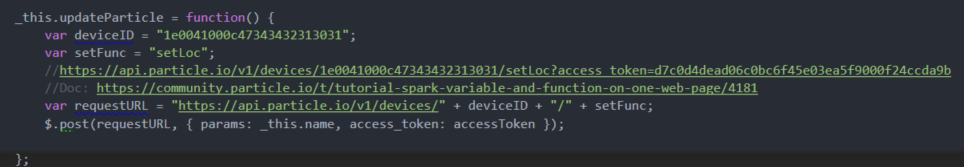
Det er også mulighet til å laste ned hele datasettet. Her er det tiltenkt at den personen som adminstrerer kan eksportere dataene og behandle dem gjennom ønsket program i tillegg til dette. Optimalt sett ville det kanskje vært best å kunne filtrere dataene etter dato osv (slik som man gjør i stream analytics for testing av data).

## Room Name = Location @Particle



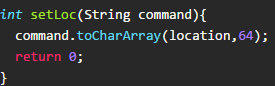
En ting som er verdt å merke her er når man oppdaterer romnavnet, så vil det også bli oppdatert i particle sin location variabel.

Javascript:



Particle Photon:

https://i.gyazo.com/499dca06dd1e8be9a3e9b889d403fe4d.png



Dette vil bli satt til bake til hva location var satt til hvis du skrur av og på enheten. Løsningen her kunne vært å titte litt på EEPROM for å manipulere firmware. <https://docs.particle.io/reference/firmware/photon/#eeprom>

# Avslutning

Front end er ikke en komplett løsning, men er en fin måte til å demonstrere hva mulighetene etter at man har satt opp Azure apps og Particle Photon.

Jeg er heller ikke sikker på om alle sensor data er 100% korrekt eller kan gjøres mer treffsikker. Jeg har i denne oppgaven fokusert mest på å kunne få data til å bli eksponert til en web-api slik at de dataene kan bli brukt av forskjellige plattformer. Dette kan være Web, mobil eller desktop applikasjoner. Det kan til og med være en raspberry pi med LCD-skjerm som henter dataene over http ?

## Known bugs

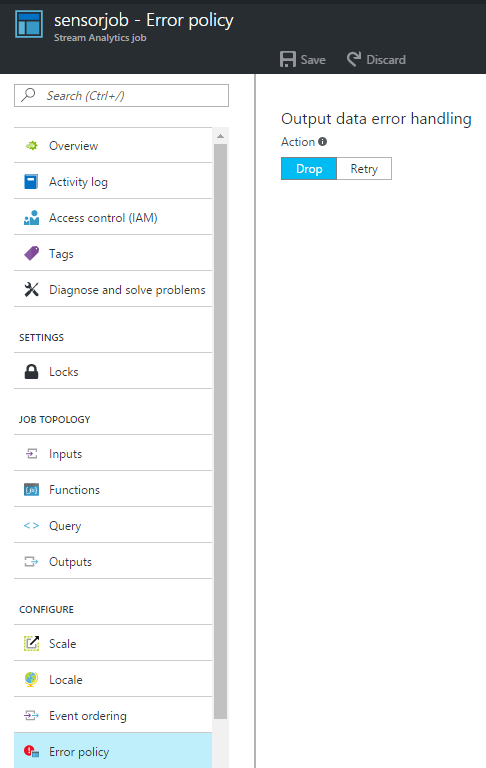
**Stream Analytics**

Jeg har støtt på en bug i stream analytics. Det hender noen ganger at particle sender data når DHT skriver ut NaN. Da vil den slite med å skrive til databasen pga typesjekking. Du kommer helt sikkert til å støte på:

Encountered error trying to write 1 event(s): Cannot convert from property 'temperature' of type 'System.String' to column 'temperature' of type 'System.Double'

Dette vil resultere med at stream analytics ikke får til å skrive data til databasen. Løsningen på dette vil være å endre på Error policy i azure portalen (Her kan du bruke den nye portalen). Bytt fra retry til drop. Dette vil sørge for at når particle sender data som ikke kan bli parset av stream analytics, er det kanskje bedre å bare ignorere dem, og vente til neste lesning. Hvis du har lengre intervall per lesning kan det lønne seg å heller CAST dataene i «Query» istedenfor. F.eks

temperature = CASE WHEN temperature = 'NaN' THEN 0.0 ELSE CAST(temperature AS Float),



# Kilder

<https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf>

<https://www.hackster.io/falafel-software/pir-sensor-with-a-photon-particle-cloud-and-a-uwa-97e79b>

<https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf>

<https://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api>

<https://www.linkedin.com/pulse/getting-started-azure-iot-services-stream-analytics-rob-tiffany>

<https://github.com/BennyHoang/webutvikling_innlevering_2>

<https://docs.particle.io/tutorials/integrations/azure-iot-hub/#example-use-cases>