



Smart Rural

Agricultura Inteligente



Recapitulamos...

Los objetivos de un sistema de Smart Rural son:

- Lanzar datos relacionados con el mundo rural a un sistema de tiempo real.
- Recoger dichos datos y lanzar eventos complejos.
- Capturar dichos eventos y realizar un procesamiento de dichos datos en tiempo real.



Generación de datos

Hemos realizado un generador con Java 8 y usando la api de ThingSpeak para generar los datos aleatorios

```
*Generating random data from channel SmartRural_2  
  
Random value of Field 1, sensorId : 4.0  
Random value of Field 1, isCeilingGreenhouseOpen : 0.91  
Random value of Field 1, isWallGreenhouseOpen : 0.49  
Random value of Field 1, countIllumination : 43.72  
Random value of Field 1, isAtDaytime : 0.75  
Random value of Field 1, canPhotosynthesisImprove : 0.5  
  
*Remember, system will generate data every 20 seconds  
  
*Generating random data from channel SmartRural_1  
  
Random value of Field 1, sensorId : 2.0  
Random value of Field 1, roomTemperature : 96.08  
Random value of Field 1, airHumidity : 59.09  
Random value of Field 1, groundHumidity : 96.13  
Random value of Field 1, litrePerMeterWater : 90.78  
Random value of Field 1, windForce : 18.99  
Random value of Field 1, windDirection : 12.44  
Random value of Field 1, isRaining : 0.28  
  
*Remember, system will generate data every 20 seconds
```

Almacén de datos

Para guardar los datos, hemos usado canales de ThingSpeak

Name	Created	Updated
 SmartRural_1 <div>PrivatePublicSettingsSharingAPI KeysData Import / Export</div>	2020-12-08	2020-12-08 23:33
 <u>SmartRural_2</u> <div>PrivatePublicSettingsSharingAPI KeysData Import / Export</div>	2020-12-08	2020-12-09 10:10

Almacén de datos

SmartRural_1

Channel ID: 1252445
Author: msa00001E904022
Access: Public

Channel for Project SmartRural

Private View Public View Channel Settings Sharing API Keys Data Import / Export

Add Visualizations

Add Widgets

Export recent data

MATLAB Analysis

MATLAB Visualization

More Information

GitHub

Channel 1 of 2 < >

Channel Stats

Created: about a month ago
Last entry: 5 minutes ago
Entries: 317



Almacén de datos

SmartRural_2

Channel ID: 1252453
Author: mwa000016964022
Access: Public

Channel for project Smart Rural

Private View

Public View

Channel Settings

Sharing

API Keys

Data Import / Export

Add Visualizations

Add Widgets

Export recent data

More information

Go to Hub

MATLAB Analysis

MATLAB Visualization

Channel 2 of 2

Channel Stats

Created: about a month ago
Last entry: 7 minutes ago
Entries: 317

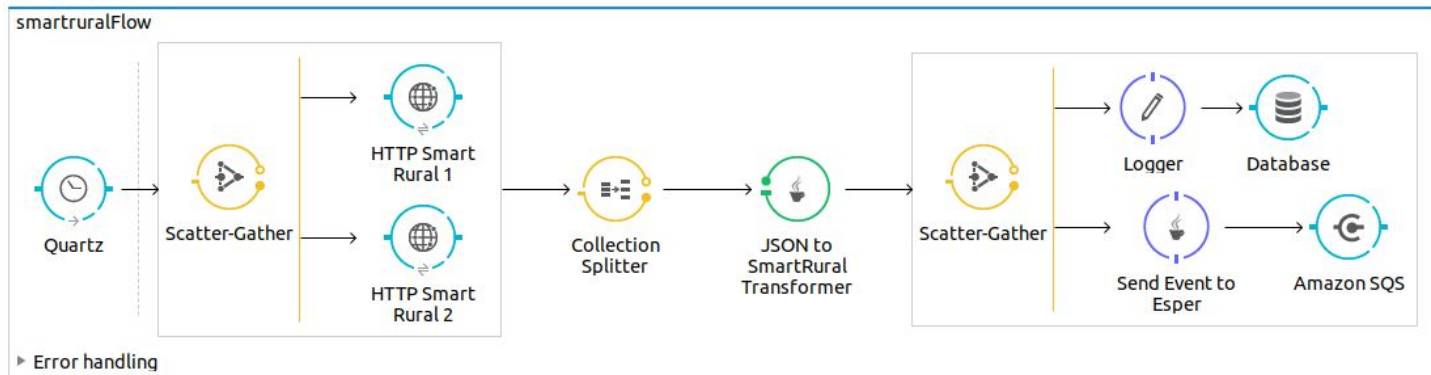


Procesamiento

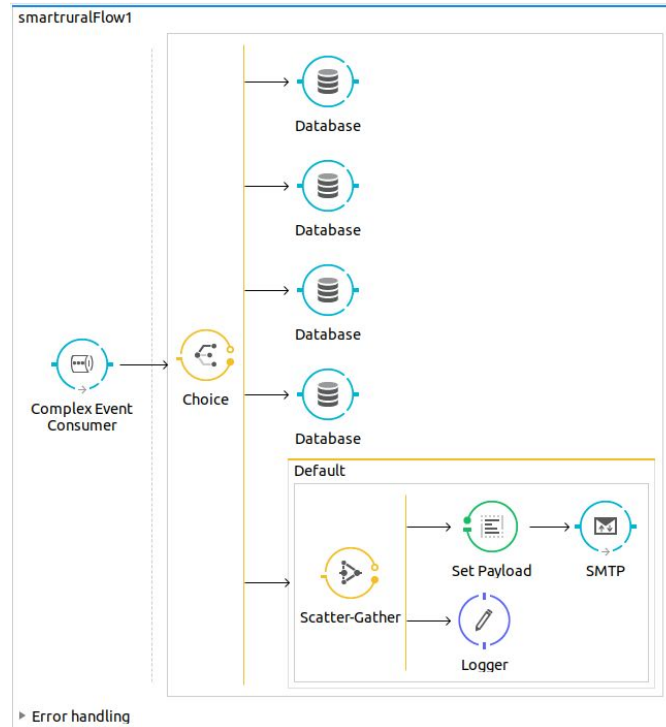
Para hacer el procesamiento de los datos, hemos realizado una aplicación mule con el AnypointStudio, y dividiendo la ejecución en 3 flujos para mayor división del trabajo.

Así pues:

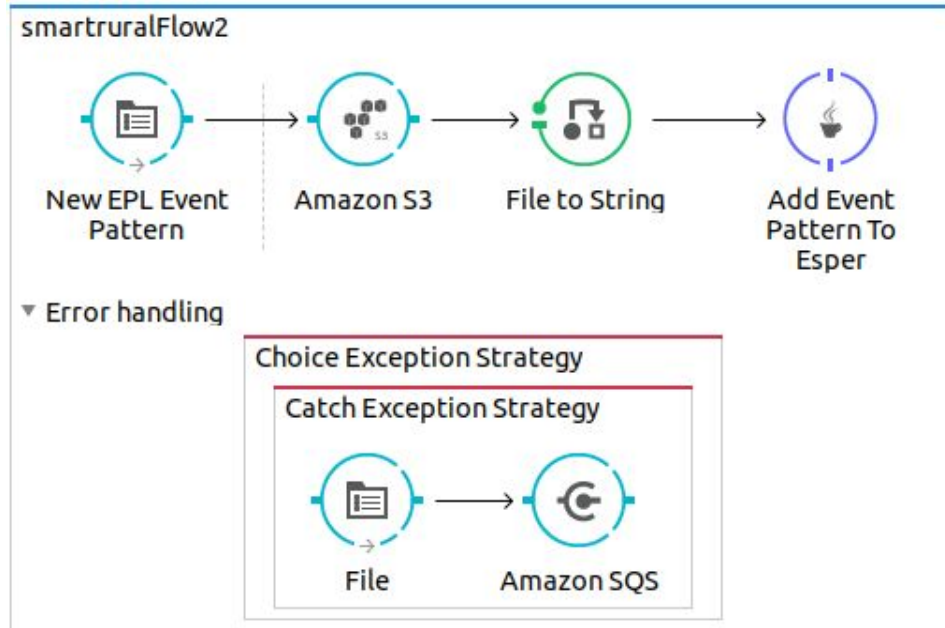
Flujo 1



Flujo 2



Flujo 3



Eventos

Como buen sistema de procesamiento continuo de eventos, debe tener eventos.

Así pues, la definición de los mismos son:

Eventos - OpenCeilingGreenHouse

```
@Name("OpenCeilingGreenHouse")
insert into OpenCeilingGreenHouse
select sr.sensorId as sensorId, sr.isAtDaytime as isAtDaytime, sr.isRaining as isRaining
from pattern [
    every sr = SmartRural(
        | isAtDaytime >= 0.5,
        | isRaining >= 0.5
    )
];
```

Eventos - Irrigate

```
@Name("Irrigate")
insert into Irrigate
select sr.sensorId as sensorId, sr.isAtDaytime as isAtDaytime, sr.isRaining as isRaining,
      sr.airHumidity as airHumidity, sr.roomTemperature as roomTemperature
from pattern [
  every
    sr = SmartRural(
      isAtDaytime >= 0.5,
      isRaining < 0.5,
      airHumidity >= 2.5,
      roomTemperature < 25 // grade celsius
    )
];
```

Eventos - CanOpenWallGreenhouse

```
@Name("OpenWallGreenhouse")
insert into OpenWallGreenhouse
select sr.sensorId as sensorId, sr.windForce as windForce, sr.isRaining as isRaining,
      sr.roomTemperature as roomTemperature
from pattern [
  every sr = SmartRural(
    windForce < 0.5,
    isRaining < 0.5,
    roomTemperature > 25 // grade celcius
  )
].win:time_batch(1 minutes);

@Name("CanOpenWallGreenhouse")
insert into CanOpenWallGreenhouse
select count(sensorId) as numEvents, (count(sensorId) > 1) as canOpen
from OpenWallGreenhouse.win:time_batch(1 min)
group by sensorId
having count(sensorId) > 1;
```

Eventos - CanFertilizer

```
@Name("Fertilizer")
insert into Fertilizer
select sr.sensorId as sensorId, sr.isAtDaytime as isAtDaytime, sr.isRaining as isRaining,
      sr.airHumidity as airHumidity, sr.canPhotosynthesisImprove as canPhotosynthesisImprove
from pattern [
  every sr = SmartRural(
    isAtDaytime >= 0.5,
    isRaining >= 0.5,
    airHumidity < 1,
    canPhotosynthesisImprove >= 0.5
  ) -> timer:at[1, *, *, *, *]
];

@Name("CanFertilizer")
insert into CanFertilizer
select sensorId
from Fertilizer.win:time_batch(1 min)
group by sensorId;
```

Base de Datos

Por otra parte, para guardar la información proveniente de los eventos para el posterior tratado por un backend, necesitamos una base de datos.

Para ello, necesitamos una definición de tablas, las cuales son:

Base de Datos

```
drop table if exists SmartRural;
create table SmartRural(
  id          bigint auto_increment primary key,
  sensorId    bigint,
  roomTemperature    float,
  airHumidity    float,
  groundHumidity  float,
  litrePerMeterWater    float,
  windForce      float,
  windDirection  float,
  countIllumination    float,
  isRaining       tinyint(1),
  isCeilingGreenhouseOpen    tinyint(1),
  isWallGreenhouseOpen    tinyint(1),
  isAtDaytime      tinyint(1),
  canPhotosynthesisImprove    tinyint(1),
  `date`           timestamp not null default current_timestamp
);

drop table if exists OpenCeilingGreenHouse;
create table OpenCeilingGreenHouse(
  id          bigint auto_increment primary key,
  `date`       timestamp not null default current_timestamp,
  message      varchar(500) not null default 'Mensaje generico para la apertura del techo del invernadero'
);

drop table if exists Irrigate;
create table Irrigate(
  id          bigint auto_increment primary key,
  `date`       timestamp not null default current_timestamp,
  message      varchar(500) not null default 'Mensaje generico para poder regar el invernadero'
);

drop table if exists CanFertilizer;
create table CanFertilizer(
  id          bigint auto_increment primary key,
  `date`       timestamp not null default current_timestamp,
  message      varchar(500) not null default 'Mensaje generico para poder fertilizar el invernadero'
);

drop table if exists CanOpenWallGreenhouse;
create table CanOpenWallGreenhouse(
  id          bigint auto_increment primary key,
  `date`       timestamp not null default current_timestamp,
  message      varchar(500) not null default 'Mensaje generico para la apertura de la pared del invernadero'
);
```

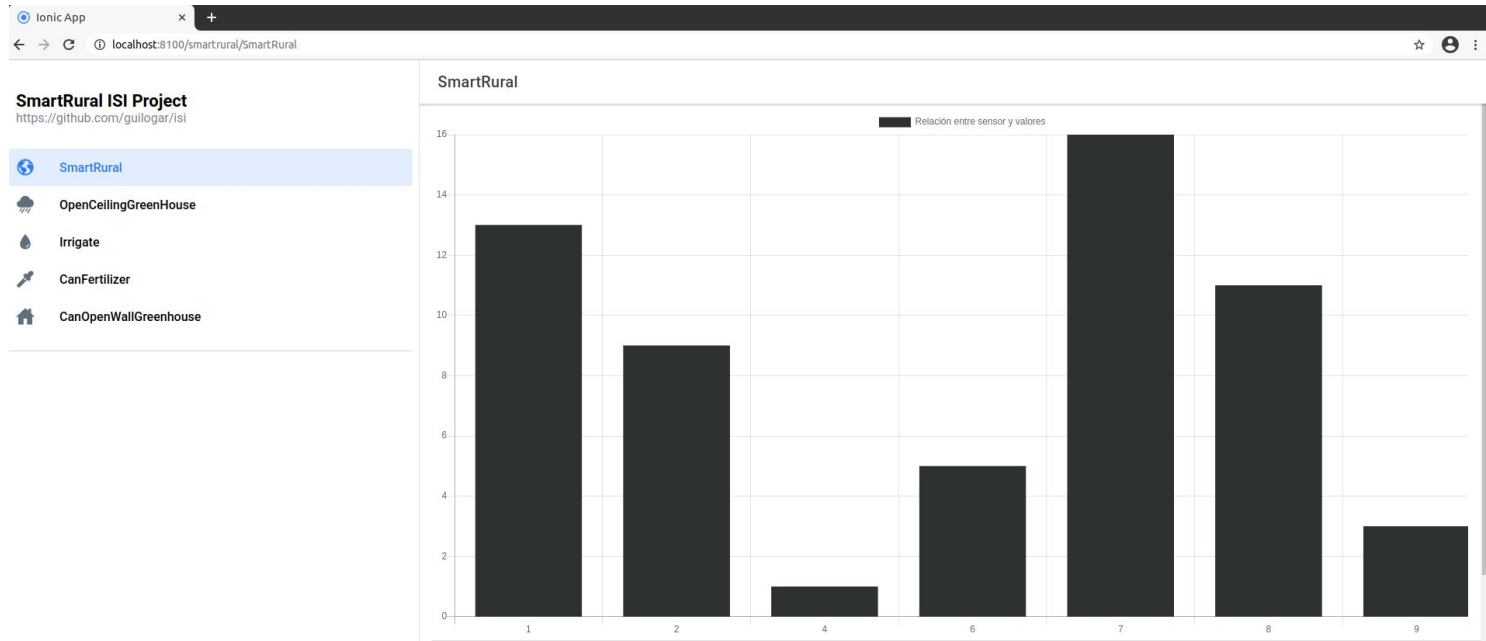
Backend + Frontend

Por último, para mostrar información de forma amigable para el usuario, necesitamos un backend que gestione la información incrustada en la base de datos y un frontend encargado de visualizarla.

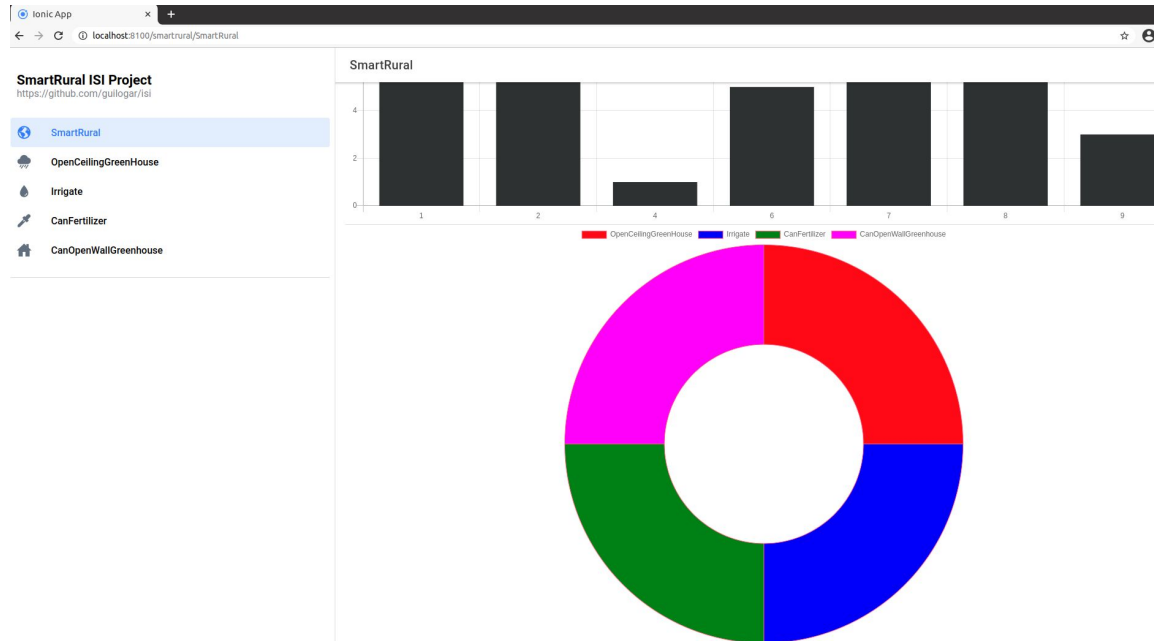
Por ende, las tecnologías usadas son:

- Backend: NodeJS con el ORM Sequelize
- Frontend: Ionic con React.JS

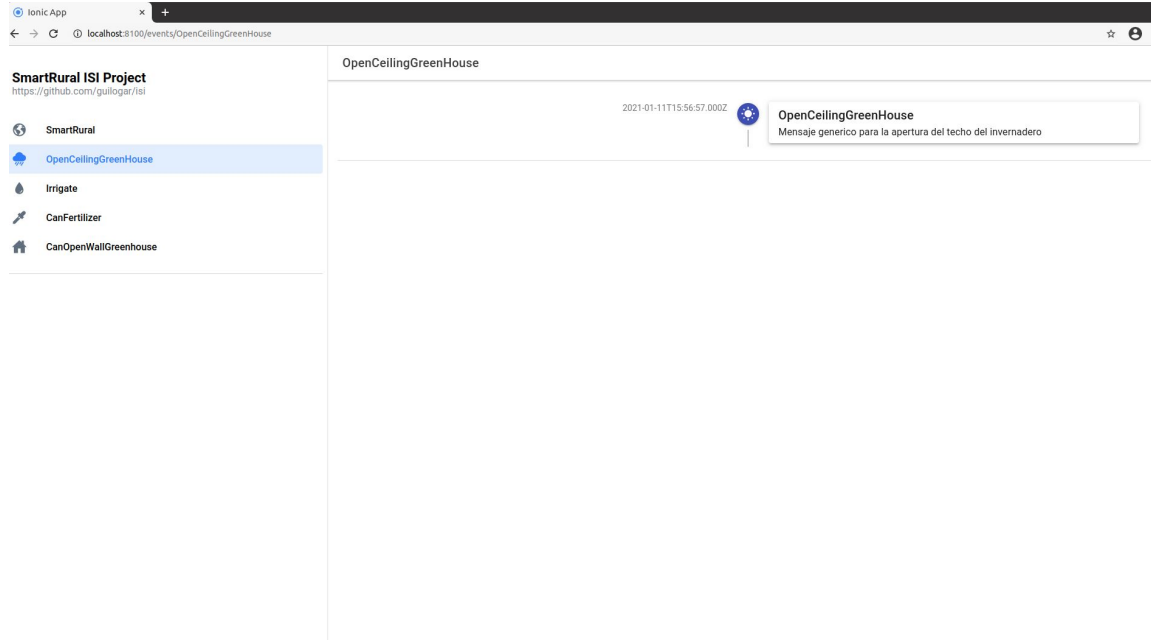
Frontend - SmartRural



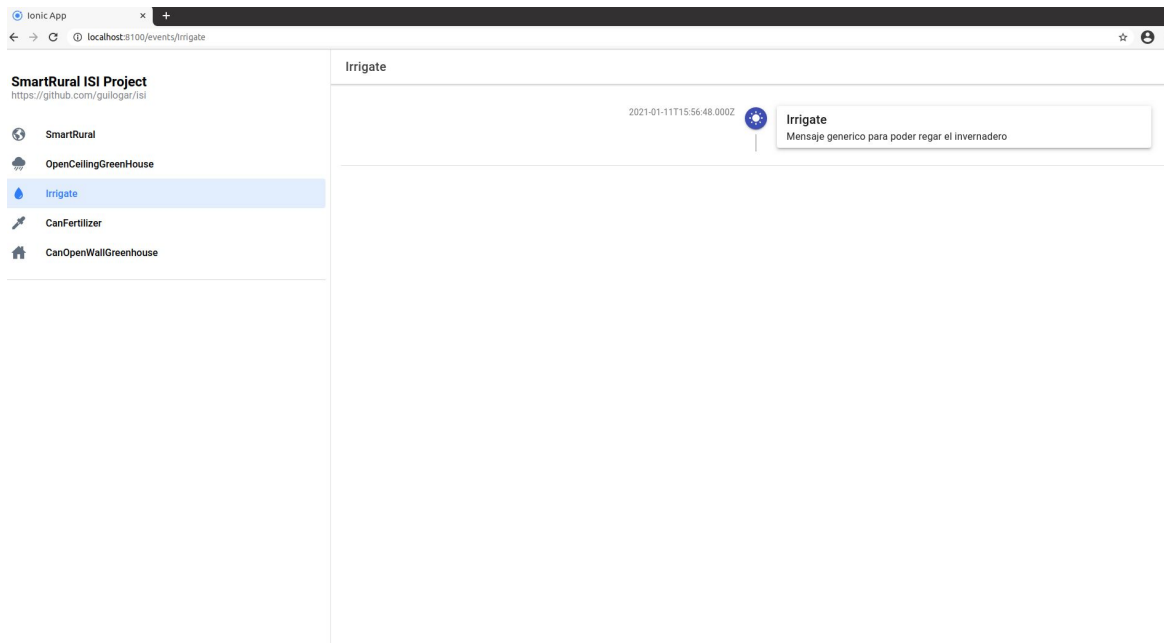
Frontend - SmartRural



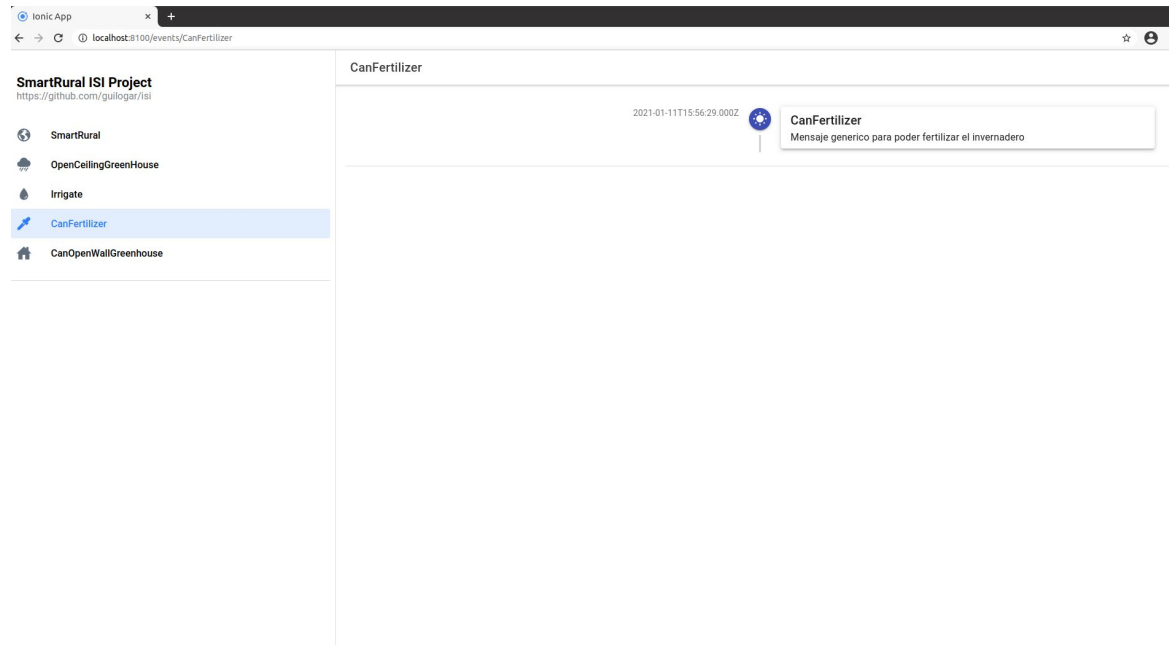
Frontend - OpenCeilingGreenHouse



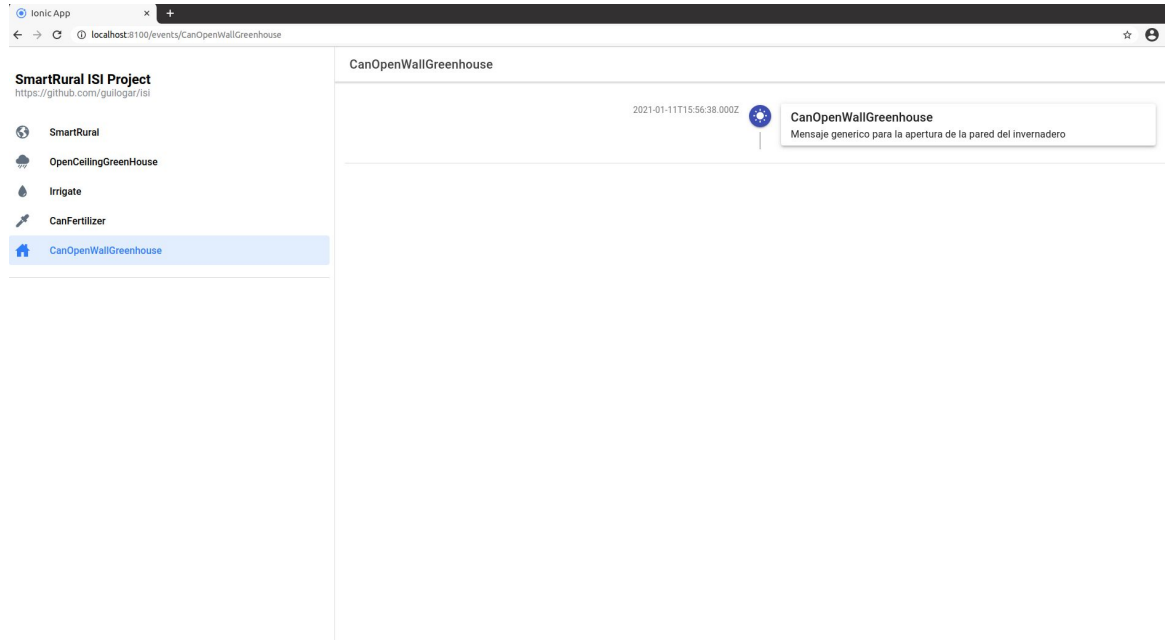
Frontend - Irrigate



Frontend - CanFertilizer



Frontend - CanOpenWallGreenhouse



Trabajo Futuro

- TFG
- TFM
- Producto real que se pueda comercializar
- Industrializar la agricultura
- Hacer feliz a la gente del campo



Gracias por su atención