ThinkSpeak Platform

ThingSpeak is an Internet of Things (IoT) platform and analytics service that allows users to collect, analyze, visualize, and act on data from sensors or other devices in the cloud. It provides instant visualizations of live data streams, and with the ability to execute MATLAB code, users can perform online analysis and processing of the data as it comes in. ThingSpeak is widely used for projects involving data logging, real-time data visualization, analysis, and is often utilized for prototyping and proof of concept IoT systems requiring analytics.

ThingSpeak Key Features

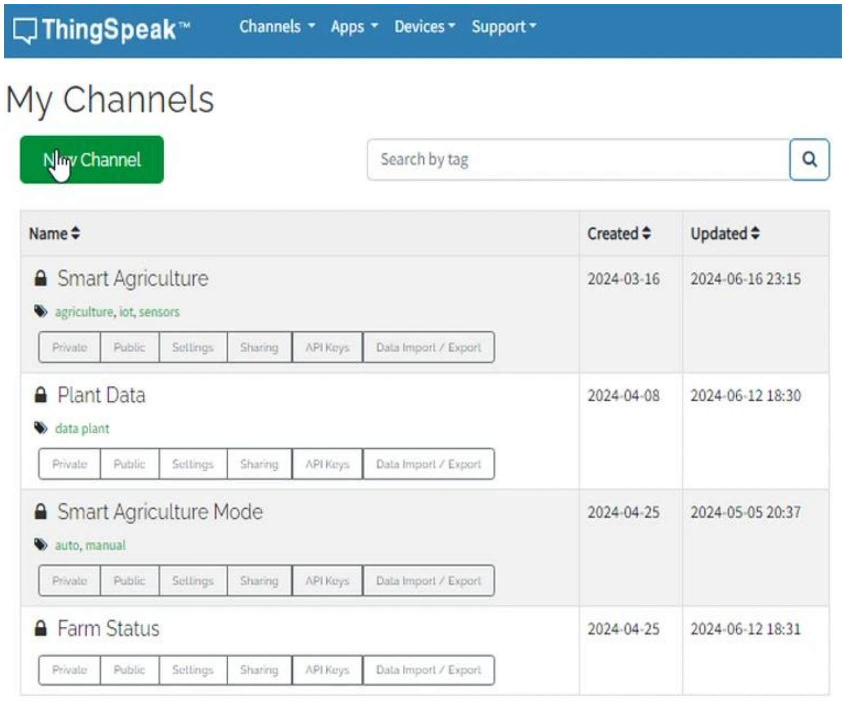
ThingSpeak allows you to aggregate, visualize and analyze live data streams in the cloud. Some of the key capabilities of ThingSpeak include the ability to:

- Easily configure devices to send data to ThingSpeak using popular IoT protocols.
- Send sensor data privately to the cloud.
- Visualize your sensor data in real-time.
- Aggregate data on-demand from third-party sources.
- Use the power of MATLAB to make sense of your IoT data.
- Run your IoT analytics automatically based on schedules or events.
- Prototype and build IoT systems without setting up servers or developing web software.
- Automatically act on your data and communicate using third-party services like Twitter.

How we can used Thinkspeak platform:

How we can used Thinkspeak platform:

1. **First thing in Thinkspeak platform : Creating Channals**
   to send and retrieve data sensor to and from the channel



My Channels

Name	Created	Updated
Smart Agriculture agriculture, iot, sensors Private · Public · Settings · Sharing · API Keys · Data Import / Export	2024-03-16	2024-06-16 23:15
Plant Data data plant Private · Public · Settings · Sharing · API Keys · Data Import / Export	2024-04-08	2024-06-12 18:30
Smart Agriculture Mode auto, manual Private · Public · Settings · Sharing · API Keys · Data Import / Export	2024-04-25	2024-05-05 20:37
Farm Status Private · Public · Settings · Sharing · API Keys · Data Import / Export	2024-04-25	2024-06-12 18:31

There is our channal that we created it and will use it in our project

Every channal from there channal have same settings like

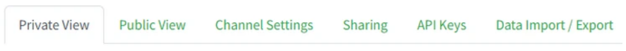
Channel Name: Enter a unique name for the ThingSpeak channel

Description: Enter a description of the ThingSpeak channel.

Field#: Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.

Tags: Enter keywords that identify the channel. Separate tags with commas.

Link to GitHub: If you store your ThingSpeak code on GitHub®, specify the GitHub repository URL.You now see these tabs:



Private View: This tab displays information about your channel that only you can see.

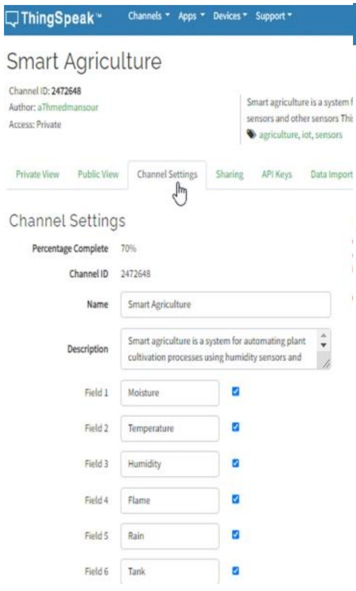
Public View: If you choose to make your channel publicly available, use this tab to display selected fields and channel visualizations.

Channel Settings: This tab shows all the channel options you set at creation. You can edit, clear, or delete the channel from this tab.

Sharing: This tab shows channel sharing options. You can set a channel as private, shared with everyone (public), or shared with specific users.

API Keys: This tab displays your channel API keys. Use the keys to read from and write to your channel.

Data Import/Export: This tab enables you to import and export channel data.



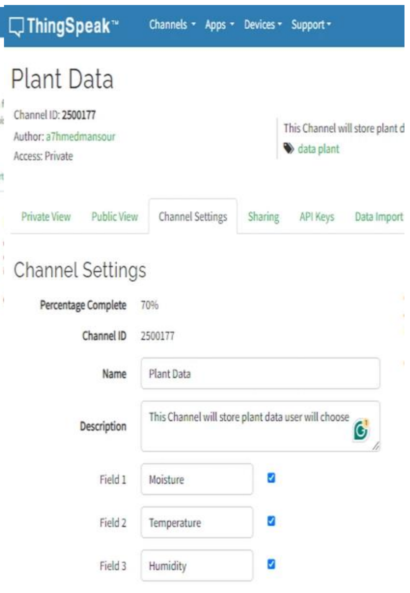
Smart Agriculture

Channel ID: 2472648
Author: a7hmedmansour
Access: Private

Smart agriculture is a system f
sensors and other sensors Thi
agriculture, iot, sensors

Channel Settings

Percentage Complete 70%
Channel ID 2472648
Name: Smart Agriculture
Description: Smart agriculture is a system for automating plant cultivation processes using humidity sensors and
Field 1: Moisture ✓
Field 2: Temperature ✓
Field 3: Humidity ✓
Field 4: Flame ✓
Field 5: Rain ✓
Field 6: Tank ✓



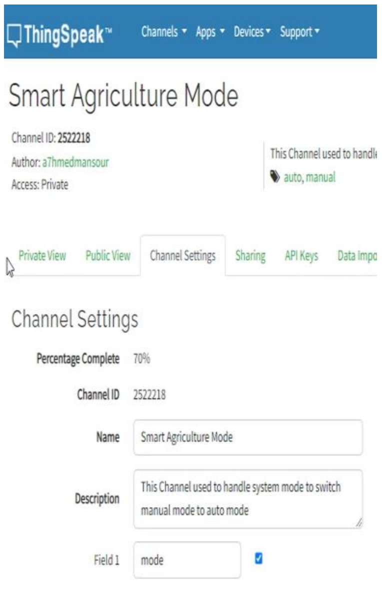
Plant Data

Channel ID: 2500177
Author: a7hmedmansour
Access: Private

This Channel will store plant d
data plant

Channel Settings

Percentage Complete 70%
Channel ID 2500177
Name: Plant Data
Description: This Channel will store plant data user will choose
Field 1: Moisture ✓
Field 2: Temperature ✓
Field 3: Humidity ✓



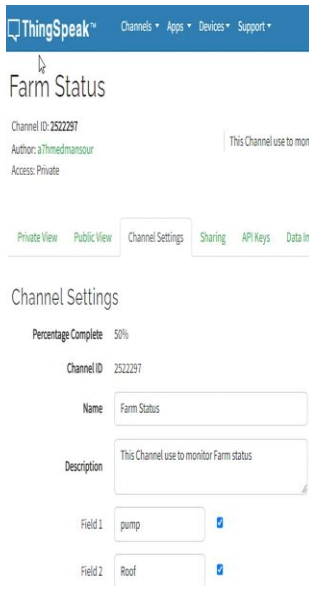
Smart Agriculture Mode

Channel ID: 2522218
Author: a7hmedmansour
Access: Private

This Channel used to handl
auto, manual

Channel Settings

Percentage Complete 70%
Channel ID 2522218
Name: Smart Agriculture Mode
Description: This Channel used to handle system mode to switch manual mode to auto mode
Field 1: mode ✓



Farm Status

Channel ID: 2522297
Author: a7hmedmansour
Access: Private

This Channel use to mon

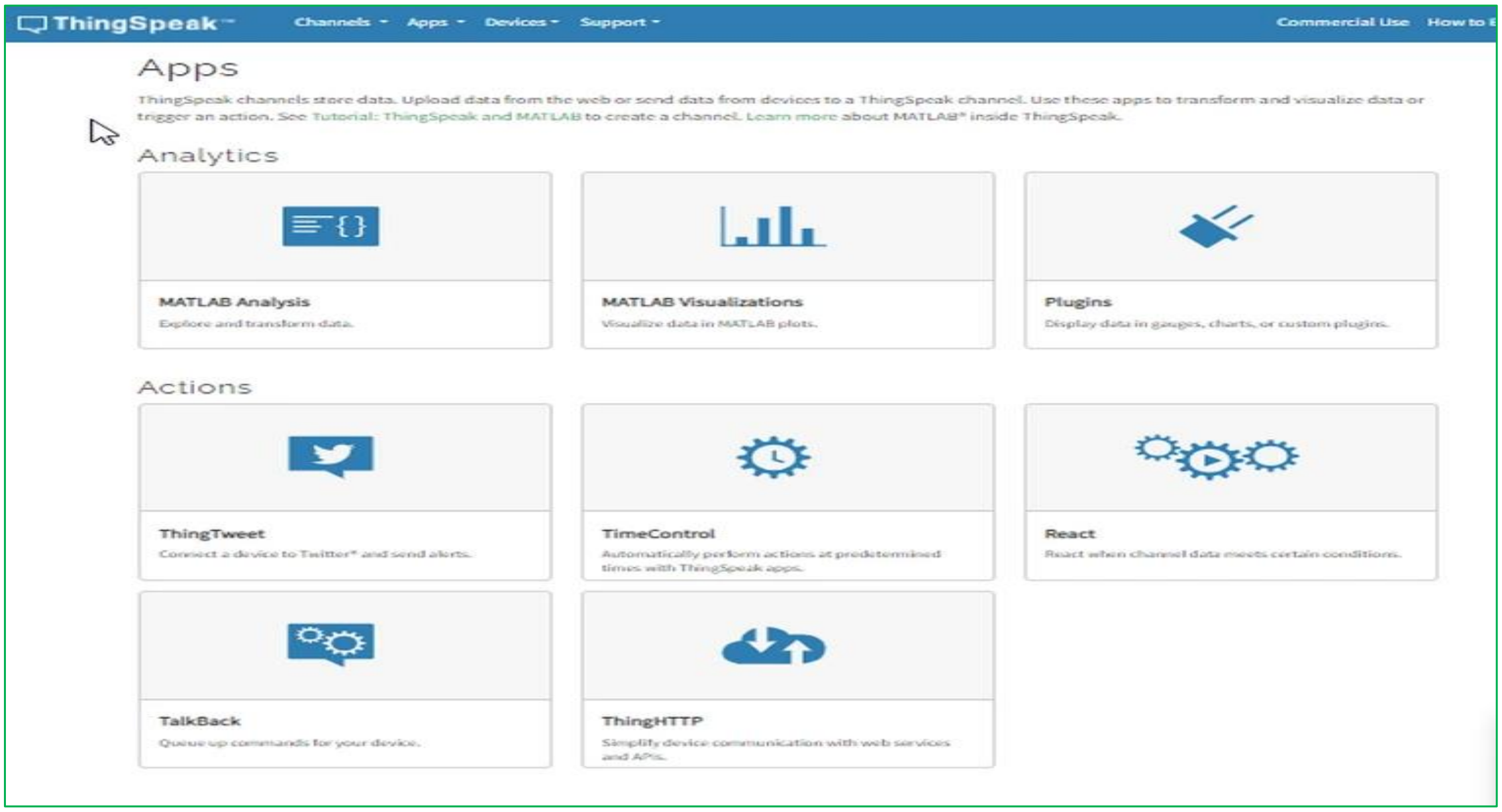
Channel Settings

Percentage Complete 50%
Channel ID 2522297
Name: Farm Status
Description: This Channel use to monitor Farm status
Field 1: pump ✓
Field 2: Roof ✓

Channals

1. **Smart Agriculture** : This Channel will store sensors data like Moisture, Temperature, Humidity, Flame, Rain, Tank
2. **Plant Data : This Channel will store plant data user will choose like** Moisture, Temperature, Humidity
3. **Smart Agriculture Mode : This Channel used to handle system mode to switch manual mode to auto mode and have one Field is** Mode
4. **Farm Status : This Channel use to monitor Farm status like** pump **,** Roof

2. **Second thing in Thinkspeak platform : ThinkSpeak Apps**



Apps

ThingSpeak channels store data. Upload data from the web or send data from devices to a ThingSpeak channel. Use these apps to transform and visualize data or trigger an action. See Tutorial: ThingSpeak and MATLAB to create a channel. Learn more about MATLAB® inside ThingSpeak.

Analytics

MATLAB Analysis
Explore and transform data.

MATLAB Visualizations
Visualize data in MATLAB plots.

Plugins
Display data in gauges, charts, or custom plugins.

Actions

ThingTweet
Connect a device to Twitter® and send alerts.

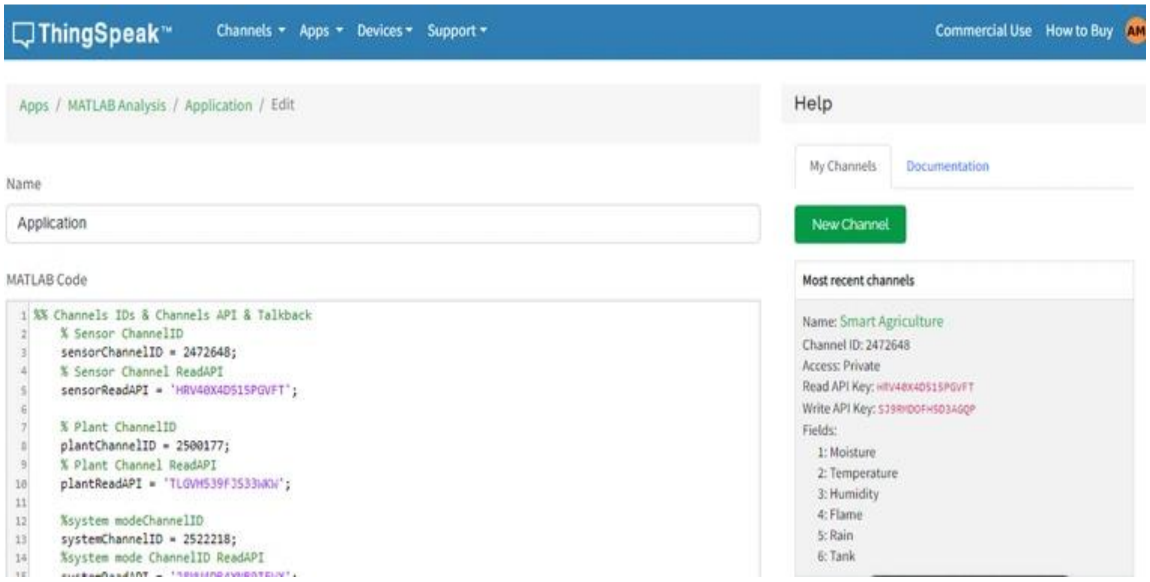
TimeControl
Automatically perform actions at predetermined times with ThingSpeak apps.

React
React when channel data meets certain conditions.

TalkBack
Queue up commands for your device.

ThingHTTP
Simplify device communication with web services and APIs.

1- MATLAB Analysis : Explore data collected in a channel or scraped from a website ,Find and remove bad data ,Convert data to different units , Calculate new data ,Build data models



Apps / MATLAB Analysis / Application / Edit

Name

Application

MATLAB Code

```
%% Channels IDs & Channels API & Talkback
    % Sensor ChannelID
    sensorChannelID = 2472648;
    % Sensor Channel ReadAPI
    sensorReadAPI = 'HRV40X4D515PGVFT';

    % Plant ChannelID
    plantChannelID = 2500177;
    % Plant Channel ReadAPI
    plantReadAPI = 'TLGVH539FJ533WKW';

    %system modeChannelID
    systemChannelID = 2522218;
    %system mode ChannelID ReadAPI
```

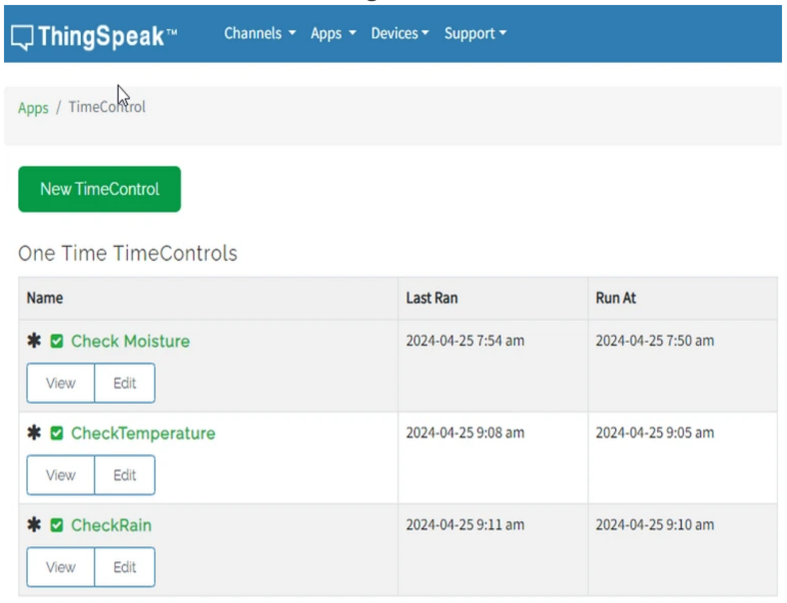
Help

My Channels | Documentation

Most recent channels

Name: Smart Agriculture
Channel ID: 2472648
Access: Private
Read API Key: HRV40X4D515PGVFT
Write API Key: S39RHDOFHSD3AGQP
Fields:
1: Moisture
2: Temperature
3: Humidity
4: Flame
5: Rain
6: Tank

2- TimeControl : Use the TimeControl app to trigger an action at a specified time. You can set up TimeControl to run MATLAB® code, send ThingTweet statuses, add new TalkBack commands, or send ThingHTTP requests.

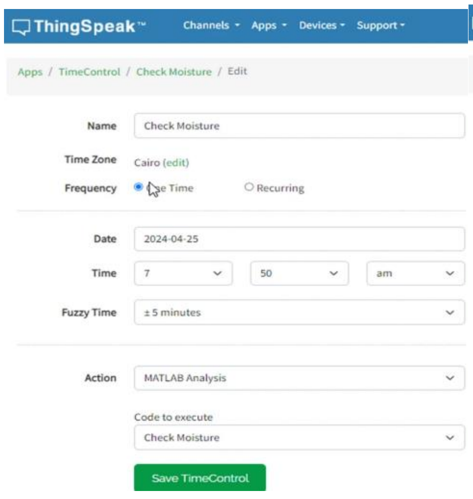


One Time TimeControls

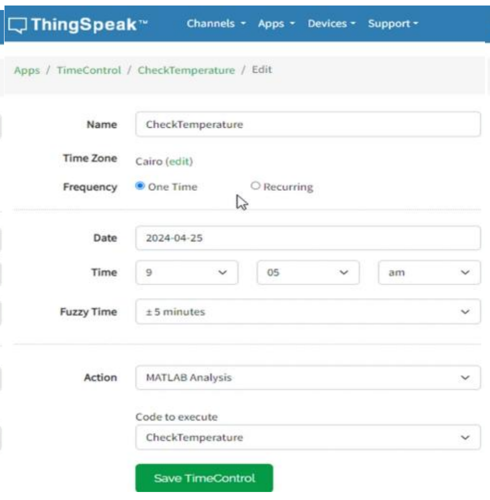
Name	Last Ran	Run At
✓ Check Moisture	2024-04-25 7:54 am	2024-04-25 7:50 am
✓ CheckTemperature	2024-04-25 9:08 am	2024-04-25 9:05 am
✓ CheckRain	2024-04-25 9:11 am	2024-04-25 9:10 am

TimeControl Settings

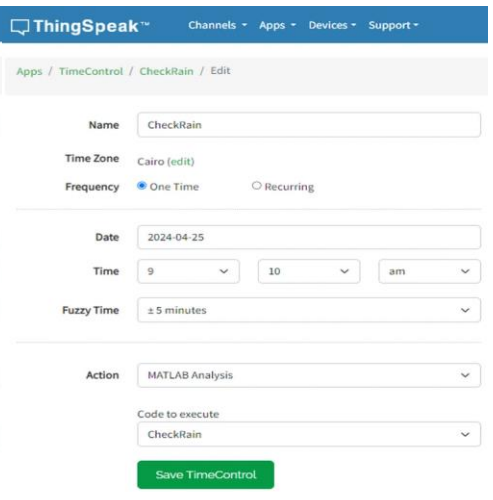
- **Name**: Enter a unique name for this TimeControl.
- **Time Zone**: The time zone is based on your account settings. To change this, click **edit**.
- **Frequency**: Choose whether the TimeControl runs once or at recurring intervals.
- **Recurrence**: Choose a recurrence pattern:
  - Week: Every week day specified
  - Day: At a specified time daily
  - Hour: At a specified hourly interval
  - Minute: At a specified minute interval
- **Time**: Enter the time at which the TimeControl runs.
- **Fuzzy Time**: Enter a random number of minutes around a scheduled time for the TimeControl to run.
- **Action**: Select the action to trigger when the specified parameters are met.
  - **MATLAB Analysis**: Run scheduled code.
  - **ThingHTTP**: Connect to a web server and read or write data.
  - **ThingTweet**: Tweet to a specified Twitter account.
  - **TalkBack**: Queue commands to send to your device.



Name: Check Moisture
Time Zone: Cairo (edit)
Frequency: [✓ One Time] Recurring
Date: 2024-04-25
Time: 7 50 am
Fuzzy Time: ± 5 minutes
Action: MATLAB Analysis
Code to execute: Check Moisture

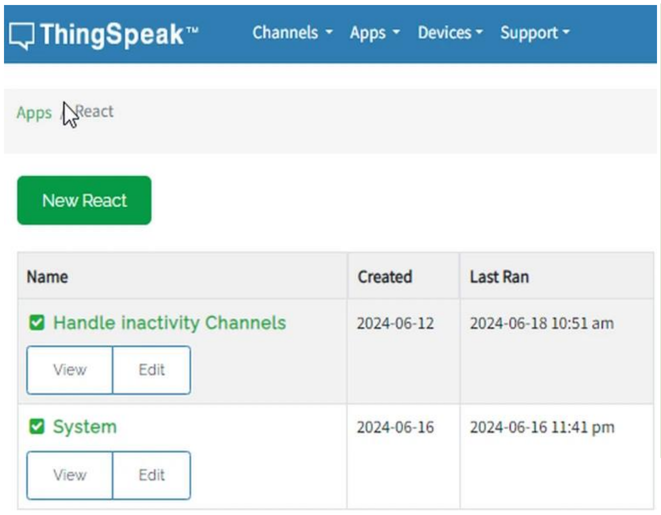


Name: CheckTemperature
Time Zone: Cairo (edit)
Frequency: [✓ One Time] Recurring
Date: 2024-04-25
Time: 9 05 am
Fuzzy Time: ± 5 minutes
Action: MATLAB Analysis
Code to execute: CheckTemperature



Name: CheckRain
Time Zone: Cairo (edit)
Frequency: [✓ One Time] Recurring
Date: 2024-04-25
Time: 9 10 am
Fuzzy Time: ± 5 minutes
Action: MATLAB Analysis
Code to execute: CheckRain

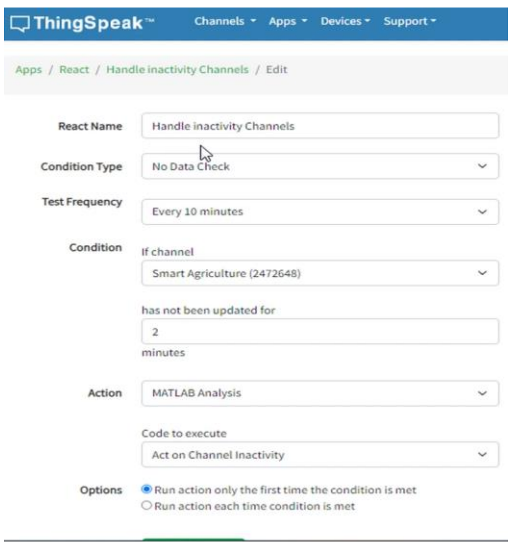
3- React : React works with ThingHTTP, ThingTweet, and MATLAB Analysis apps to perform actions when channel data meets a certain condition. For example, you can have a mobile app report your latitude and longitude to a ThingSpeak channel. When your position is within a certain distance of your house, have ThingHTTP turn on your living room lights.



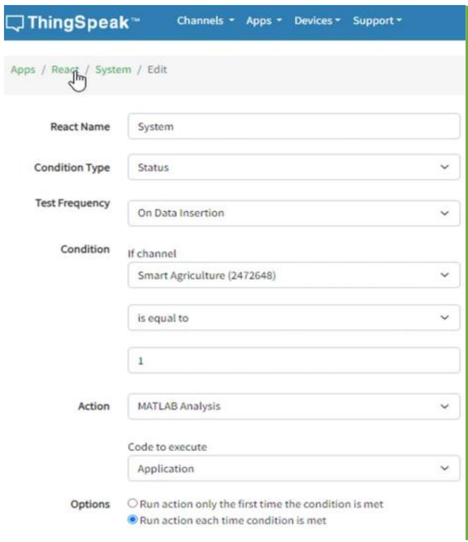
Name	Created	Last Ran
✓ Handle inactivity Channels	2024-06-12	2024-06-18 10:51 am
✓ System	2024-06-16	2024-06-16 11:41 pm

React Settings

- **React Name**: Enter a unique name for your React.
- **Condition Type:** Select a condition type corresponding with your data. A channel can hold numeric sensor data, text, strings, status updates, or geographic location information.
- **Test Frequency:** Choose whether to test your condition every time data enters the channel or on a periodic basis.
- **Condition**: Select a channel, a field and the condition for your React.
- **Action:** Select ThingTweet, ThingHTTP, or MATLAB Analysis to run when the condition is met.
- **Options**: Select when the React runs.



React Name	Handle inactivity Channels
Condition Type	No Data Check
Test Frequency	Every 10 minutes
Condition	If channel Smart Agriculture (2472648) has not been updated for 2 minutes
Action	MATLAB Analysis Code to execute Act on Channel Inactivity
Options	● Run action only the first time the condition is met ○ Run action each time condition is met



React Name	System
Condition Type	Status
Test Frequency	On Data Insertion
Condition	If channel Smart Agriculture (2472648) is equal to 1
Action	MATLAB Analysis Code to execute Application
Options	○ Run action only the first time the condition is met ● Run action each time condition is met

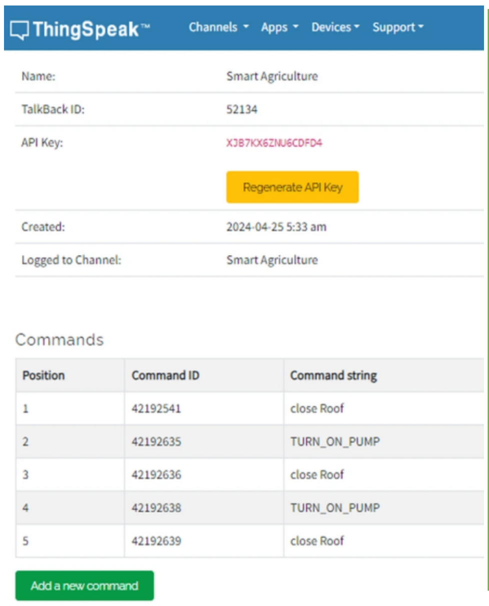
Handle inactivity Channels : this React use when No Data Check Every 10 Minutes in Smart Agriculture channal and has not been updated for 2 minutes

Action will be use Matlab Analysis and choose code to execute

System : this React use when No Data insertion in Smart Agriculture channal

Action will be use Matlab Analysis and choose code to execute

4- TalkBack :



Name:	Smart Agriculture
TalkBack ID:	52134
API Key:	XJB7KX6ZNU6CDFD4
Created:	2024-04-25 5:33 am
Logged to Channel:	Smart Agriculture

Commands

Position	Command ID	Command string
1	42192541	close Roof
2	42192635	TURN_ON_PUMP
3	42192636	close Roof
4	42192638	TURN_ON_PUMP
5	42192639	close Roof

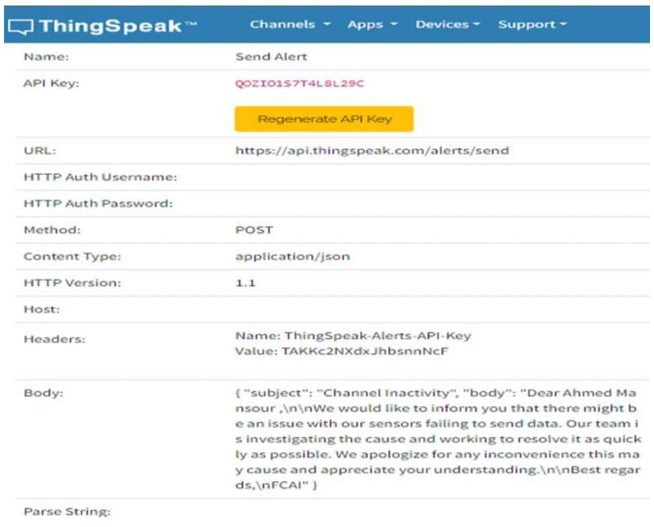
TalkBack enables any device to act on queued commands

in this example my **queued** is message for every processes in Roof in case of **OPEN** or **CLOSE**

and this in our code will dependent on **soilMoisture**

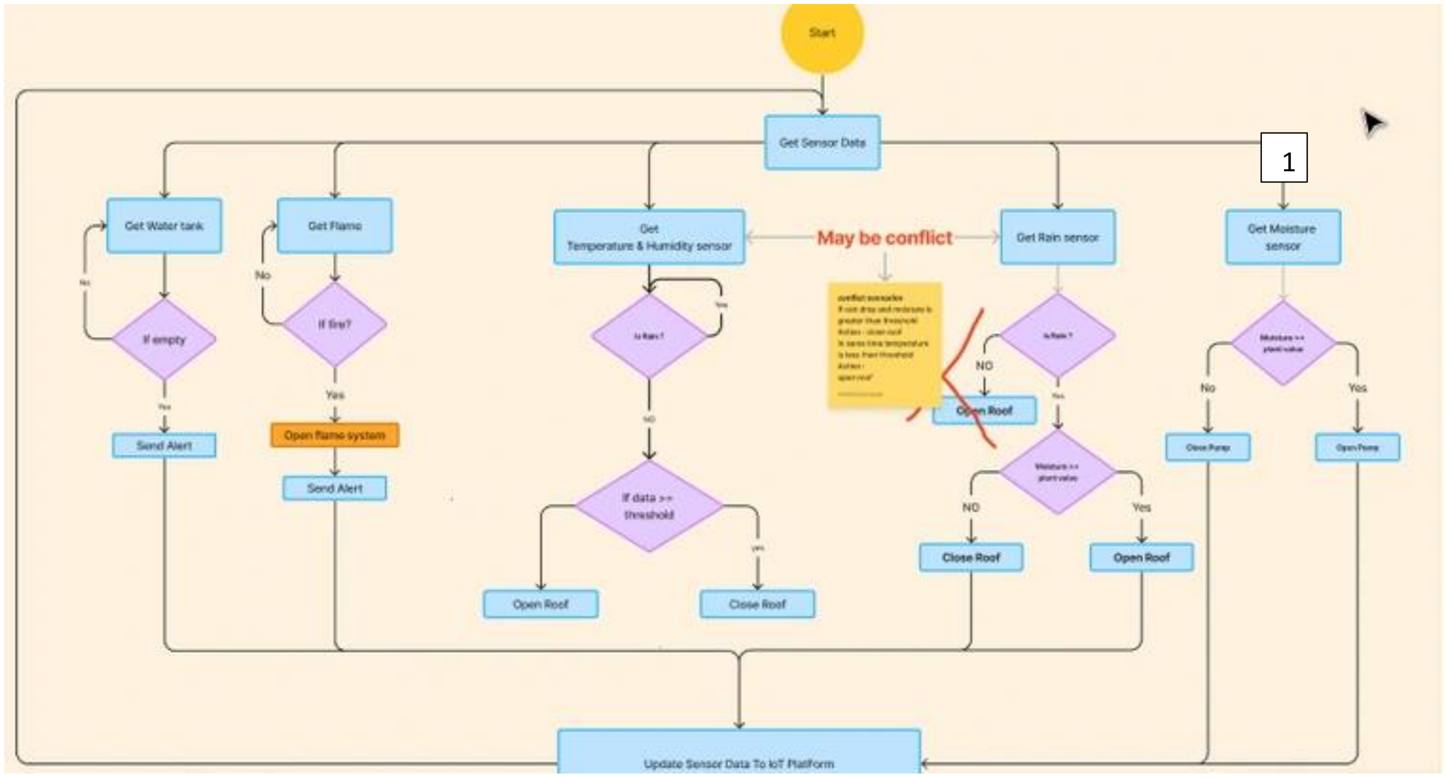
```
% TalkBack app ID
TalkBack_ID = '52134';
% TalkBack app API key
TalkBack_apikey = 'XJB7KX6ZNU6CDFD4';
%url to send commands to TalkbackAPP
url =
strcat('https://api.thingspeak.com/talkbacks/',TalkB
ack_ID,'/commands.json');
```

5- ThingHTTP : ThingHTTP enables communication among devices, websites, and web services without having to implement the protocol on the device level. You specify actions in ThingHTTP, which you trigger using other ThingSpeak apps such as TweetControl, TimeControl, and React .



Name:	Send Alert
API Key:	QOZIO1S7T4L8L29C
URL:	https://api.thingspeak.com/alerts/send
HTTP Auth Username:	
HTTP Auth Password:	
Method:	POST
Content Type:	application/json
HTTP Version:	1.1
Host:	
Headers:	Name: ThingSpeak-Alerts-API-Key Value: TAKKc2NXdxJhbsnnNcF
Body:	{ "subject": "Channel Inactivity", "body": "Dear Ahmed Mansour ,\n\nWe would like to inform you that there might be an issue with our sensors failing to send data. Our team is investigating the cause and working to resolve it as quickly as possible. We apologize for any inconvenience this may cause and appreciate your understanding.\n\nBest regards,\nFCAI" }
Parse String:	

Explain Our Code



General Data (Channels IDs & Channels API & Talkback)

```
% Sensor ChannelID
    sensorChannelID = 2472648;
    % Sensor Channel ReadAPI
    sensorReadAPI = 'HRV40X4D515PGVFT';
```

2472648 is a channel ID of **Smart Agriculture**

And HRV40X4D515PGVFT is Read API Keys Use this key to allow other people to view your private channel feeds and charts.

Read API Keys

Key HRV40X4D515PGVFT

```
% Plant ChannelID
    plantChannelID = 2500177;
    % Plant Channel ReadAPI
    plantReadAPI = 'TLGVH539FJS33WKW';
```

2500177is a channel ID of **Plant Data**

And 'TLGVH539FJS33WKW' is Read API Keys Use this key to allow other people to view your private channel feeds and charts.

```
    %system modeChannelID
    systemChannelID = 2522218;
    %system mode ChannelID ReadAPI
    systemReadAPI = '28MU4DB4XNB9IEWX';
```

2522218 is a channel ID of **Smart Agriculture Mode**

And '28MU4DB4XNB9IEWX'is Read API Keys Use this key to allow other people to view your private channel feeds and charts.

```
    % TalkBack app ID
    TalkBack_ID = '52134';
    % TalkBack app API key
    TalkBack_apikey = 'XJB7KX6ZNU6CDFD4';
    %url to send commands to TalkbackAPP
    url =  strcat('https://api.thingspeak.com/talkbacks/',TalkBack_ID,'/commands.json');
```

```
'52134' is a TalkBack_ID

And 'XJB7KX6ZNU6CDFD4' is TalkBack_apikey used to
connect to this talkBack

url use to send commands to TalkbackAPP
```

```
'TAKKc2NXdxJhbsnnNcF' is a alertApiKey use to
send alert to mail if any condition has been
```

The Alerts REST API schedules emails to be sent when you send a request to `api.thingspeak.com/alerts/send`

```
webwrite uses weboptions to add required
headers. Alerts need a ThingSpeak-Alerts-API-Key
```

Alerts API Key TAKKc2NXdxJhbsnnNcF

```
% ThingSpeak Alerts API Key
    alertApiKey = 'TAKKc2NXdxJhbsnnNcF';%
    % Set the address for the HTTP call
    alertURL = "https://api.thingspeak.com/alerts/send";
   % webwrite uses weboptions to add required headers. Alerts need a ThingSpeak-Alerts-API-Key header.
   options = weboptions("HeaderFields", ["ThingSpeak-Alerts-API-Key", alertApiKey]);
```

Function To Check Moisture

```
function CheckMoisture
(soilMoisture,plantMoisture,TalkBack_apikey,url)
% Check Conditions for Moisture and handle potential
webwrite errors
if isnan(soilMoisture)
    disp(["Moisture sensor is ",soilMoisture]);
elseif isnan(plantMoisture)
    disp(["plant Moisture is ",plantMoisture]);
else
    try
        if soilMoisture >= plantMoisture
        response = webwrite(url, 'api_key',
TalkBack_apikey, 'command_string', 'TURN_ON_PUMP');
        disp("True ON PUMP");
        else
        response = webwrite(url, 'api_key',
TalkBack_apikey, 'command_string', 'TURN_OFF_PUMP');
        disp("True OFF PUMP");
        end
        % Display the response from the web service
        disp("Response:");
        disp(response);
    catch ME
        % Handle errors in the webwrite function
        disp('An error occurred:');
        disp(ME.message);
    end
end
```

This function use to check if a Soil Misture is greater than or equal to PlantMoisture and if this condition is True the PUMP will OPEN A command to turn on the water pump (`TURN_ON_PUMP`) is sent using `webwrite`

Else the PUMP will CLOSE The response from the web service is displayed.

This function take `(soilMoisture,plantMoisture,TalkBack_apikey,url)`

As prameters and display soilMoisture,plantMoisture

TalkBack_apikey: The API key for authorization with the web service.
url: The URL of the web service to send commands to

`isnan(soilMoisture)`: Checks if the `soilMoisture` value is `NaN` (Not a Number), indicating a sensor error or missing data.

catch ME: Catches any errors that occur during the execution of the `try` block.

Function To Check Raining

```
function
CheckRain(RainSensor,Moisture,plantMoisture,TalkBack_ap
ikey,url)
if isnan(RainSensor)
    disp(["Rain Sensor is ",soilMoisture]);
elseif isnan(plantMoisture)
    disp(["plant Moisture is ",plantMoisture]);
else
    % Check if it is raining
    if RainSensor == 0 % RainSensor = 0 --> Raining
        % Check if the soil needs irrigation
        try
            if Moisture >= plantMoisture
                response = webwrite(url, 'api_key',
TalkBack_apikey, 'command_string', 'TURN_ON_ROOF');
                disp("open Roof");
            else
                response = webwrite(url, 'api_key',
TalkBack_apikey, 'command_string', 'TURN_OFF_ROOF');
                disp("close Roof");
            end
            % Display the response from the web service
            disp("Response:");
            disp(response);
        catch ME
            % Handle errors in the webwrite function
            disp('An error occurred:');
            disp(ME.message);
        end
    else
        disp("It is not raining, no action taken.");
    end
end
end
```

This function use to check **Raining**

RainSensor == 0: Checks if the rain sensor indicates that it is raining (assuming 0 means rain).
The second condition if it raining
Moisture >= plantMoisture: Checks if the soil moisture is above the threshold needed by the plant.

If it is raining and the soil needs irrigation (moisture level is above or equal to the threshold), it sends a command to open the roof (TURN_ON_ROOF) using webwrite.
If the soil does not need irrigation, it sends a command to close the roof (TURN_OFF_ROOF) using webwrite.
If it is not raining (`RainSensor` is not 0), it simply displays a message indicating no action is taken.

This function take
(RainSensor,Moisture,plantMoisture,TalkBack_apike,url)
As prameters

and display soilMoisture if

isnan(RainSensor): Checks if the RainSensor value is NaN (Not a Number)

and display plantMoisture if

isnan(plantMoisture): Checks if the plantMoisture value is NaN (Not a Number)

TalkBack_apikey: The API key for authorization with the web service.
url: The URL of the web service to send commands to

catch ME: Catches any errors that occur during the execution of the try block.

Function To Check Temperature

```
function
CheckTemperature(Temperature,Humidity,plantTemperature
,plantHumidity,TalkBack_apikey,url)
    % Check temperature and humidity conditions
if isnan(Temperature) || isnan(Humidity)
disp(["Temperature & Humidity sensor is
",Temperature]);
elseif isnan(plantTemperature) || isnan(plantHumidity)
    disp(["plant Temperature is ",plantTemperature]);
    disp(["plant Humidity is ",plantHumidity]);
else
    try
        if Temperature >= plantTemperature
            response = webwrite(url, 'api_key',
TalkBack_apikey, 'command_string', 'close Roof');
            disp("close Roof");
        elseif Temperature <= plantTemperature
            if RainSensor == 1  % Not Raining
                response = webwrite(url, 'api_key',
TalkBack_apikey, 'command_string', 'open Roof');
                disp("open Roof");
            else
                response = 'No action taken because it
is raining.';
            end
        else
            response = 'No action taken.';
        end
        % Display the response from the web service or
the custom message
        disp("Response:");
        disp(response);
    catch ME
        % Handle errors in the webwrite function
        disp('An error occurred:');
        disp(ME.message);
    end
end
end
```

This function use to check **Temperature**

If the temperature is greater than or equal to the plant's required temperature (Temperature >= plantTemperature), it sends a command to close the roof (close Roof) using webwrite.

If the temperature is less than or equal to the plant's required temperature (Temperature <= plantTemperature), it checks the rain condition.

If RainSensor == 1 (assuming 1 means not raining), it sends a command to open the roof (open Roof) using webwrite.

If it is raining (RainSensor != 1), it sets the response to "No action taken because it is raining."

If neither condition is met, it sets the response to "No action taken."

This function take
(Temperature,Humidity,plantTemperature,plantHumidity,TalkBack_apikey,url)
As prameters
and display Temperature if

isnan(Temperature) || isnan(Humidity):Checks if the Temperature and Humidity value is NaN (Not a Number)

and display plantTemperature and plantHumidity if

isnan(plantTemperature) || isnan(plantHumidity): Checks if the plantTemperature and plantHumidity value is NaN (Not a Number)

TalkBack_apikey: The API key for authorization with the web service.
url: The URL of the web service to send commands to

catch ME: Catches any errors that occur during the execution of the try block.

Function To Check Tank Level

```
function MonitorTankLevel(TankData,alertURL,options)
 % Tank full --> 0
 if isnan(TankData)
     disp(["Tank Sensors is ",TankData]);
 else
    if TankData == 1 % empty --> 1
        alertBody = 'Tank Is Empty';
        alertSubject = 'Tank Status Alert: Empty';
        disp('Tank is empty');

        % Send alert
        try
            response = webwrite(alertURL, "body",
alertBody, "subject", alertSubject, options);
            disp('Alert sent successfully.');
            disp("Response:");
            disp(response);
        catch ME
            % Handle errors in the webwrite function
            disp('Failed to send alert:');
            disp(ME.message);
        end
    elseif TankData == 0 % Full --> 0
        disp('Tank is not Full.');
    end
 end
end
```

This function use to check **Tank Level**

This function take (TankData,alertURL,options)
As prameters

If TankData == 1, it indicates the tank is empty

The function sets the alert body and subject messages to indicate that the tank is empty, It then displays a message "Tank is empty".

If TankData == 0, it indicates the tank is full

It displays a message "Tank is Full ".

The function attempts to send an alert using webwrite with the alertURL, alertBody, alertSubject, and options.

If the alert is sent successfully, it displays the response from the web service.

If an error occurs during the webwrite function, it catches the exception and displays an error message.

and display TankData if

isnan(TankData):Checks if the TankData value is NaN (Not a Number)

catch ME: Catches any errors that occur during the

Function To CheckFlameSensor

```
function MonitorFlameSensor(FlameData,alertURL,options)
if isnan(FlameData)
    disp(["Flame Sensors is ",FlameData])
else
   % Check if fire is detected
    if FlameData == 1 % fire detected
        % Define the alert message
        alert_body = 'There is a fire';
        alert_subject = 'ThingSpeak Alert email';


        disp('Fire detected! Sending alert...');
        try
            % Send the alert
            response = webwrite(alertURL, "body",
alert_body, "subject", alert_subject, options);
            disp('Alert sent successfully.');
            disp("Response:");
            disp(response);
        catch ME
            % Handle errors in the webwrite function
            disp('An error occurred while sending the
alert:');
            disp(ME.message);
        end
    else
        disp('No fire detected.');
    end

end
end
```

This function use to check **FlameSensor**

This function take (FlameData,alertURL,options)
As prameters

If `FlameData == 1`, it indicates that a fire has been detected.

The function sets the alert body and subject messages to indicate the presence of fire.

It then displays a message "Fire detected! Sending alert...".

If `FlameData == 0`, it indicates that no fire has been detected.

It displays a message "No fire detected.".

The function attempts to send an alert using webwrite with the alertURL, alertBody, alertSubject, and options.

If the alert is sent successfully, it displays the response from the web service.

If an error occurs during the webwrite function, it catches the exception and displays an error message.

and display `FlameData` if

`isnan(FlameData)`:Checks if the `FlameData` value is NaN (Not a Number)

catch ME: Catches any errors that occur during the execution of the try block.

Check System Auto or Manual

```
systemMode = thingSpeakRead(systemChannelID,
'Fields', 1, ReadKey=systemReadAPI);
% Auto = 1 , manual = 0
if systemMode == 1
```

The thingSpeakRead function returns the value of field 1 from the specified channel, storing it in the variable systemMode.

if systemMode == 1: Checks if the systemMode value is 1, indicating that the system is in automatic mode. If true, the code within the if block will execute.