Smart Home Assistant Telegram Bot using ESP8266

Acknowledgment

I would like to express my sincere gratitude to Dr. Dayan Rajapakse, Founder of Esoft Metro campus and SmartInternz for providing me a great opportunity in order to get project experience before joining the industry. And I acknowledge with thanks the support and timely guidance which I have received from my mentor Mr.Gnaneshwar Bandari. The completion of this project could not have been possible without the assistance and guidance of my mentor.

Thank you so much!

Introduction

☞ Overview

In the present trend and technologies smart homes has become a buzzword. Making the homes smart using the emerging technologies like the Internet of things and Artificial intelligence has become more popular.

☞ Purpose

The purpose of this project is to make Smart Home Assistant Telegram Bot Using ESP8266.
Note: The project is accomplished using IBM IoT Watson platform and Node red.

Literature survey

☞ Existing problem

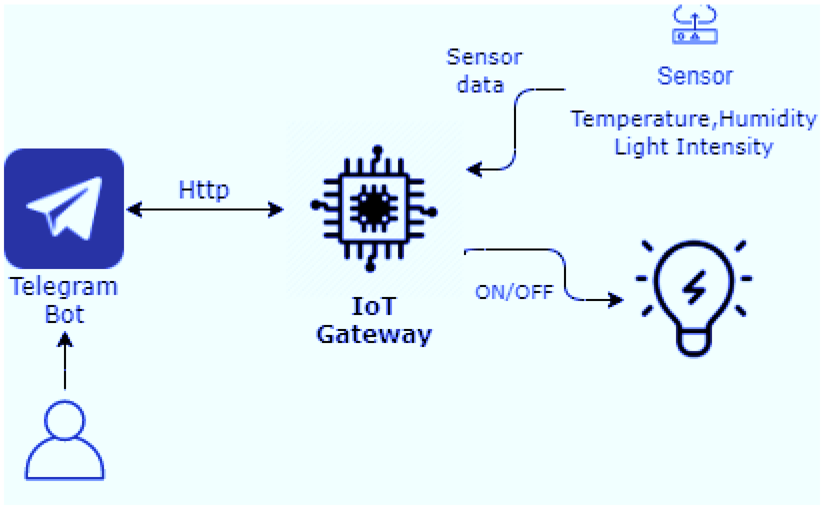
Mostly, in manual homes, the dweller must perform the workflow manually by themselves which gives discomfort and discommode for the dweller. Thus, this needs to be automated which termed as "Smart Homes".

☞ Proposed solution

To automate the homes with new technology such as Internet of Things (IoT) in order automate the workflow and to enhance the quality of life and convenience of the dweller.

Theoretical Analysis

☞ Block diagram



☞ Hardware/Software designing

Software used to complete the project is Telegram. And the programming language used for the completion of the project is Python (version 3.9).

Experimental Investigation

The development of this project has been achieved by using IBM Watson IoT platform and Node red which is very conductive to develop the the project with regard to need of the user. Screenshots of them are included in the appendix.

Result

Refer appendix for the result. The output of the the project has been included in the appendix with screenshots.

Advantages of IBM Watson IoT platform and Node-Red with regard to the project

Advantages of IBM Watson IoT platform.

- ✓ It has a substantial amount of potential with API.
- ✓ It integrates the capacity and functions of IBM IoT Connection Service in an end-to-end cloud service.
- ✓ It is secured, hence the users can safely develop the project without any trouble.

Advantages of Node-Red.

- ✓ It is simple and easy to program in Node-red.
- ✓ The flows created in Node-Red are saved using JSON, which share the flows with others.
- ✓ It provides wide range of nodes in the palette which makes the flow wire together.

Disadvantages of IBM Watson IoT platform and Node-Red

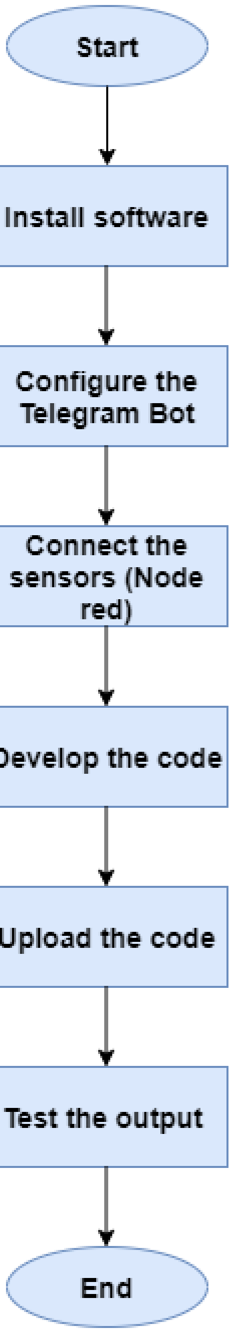
Disadvantages of IBM Watson IoT platform

- ✓ Slow integration.
- ✓ Takes time to setup with the devices.

Disadvantages of Node-Red.

- ✓ Node.js API has some consistence issues.
- ✓ It does not have a well facilitated library system compared to other programming languages.

Flowchart

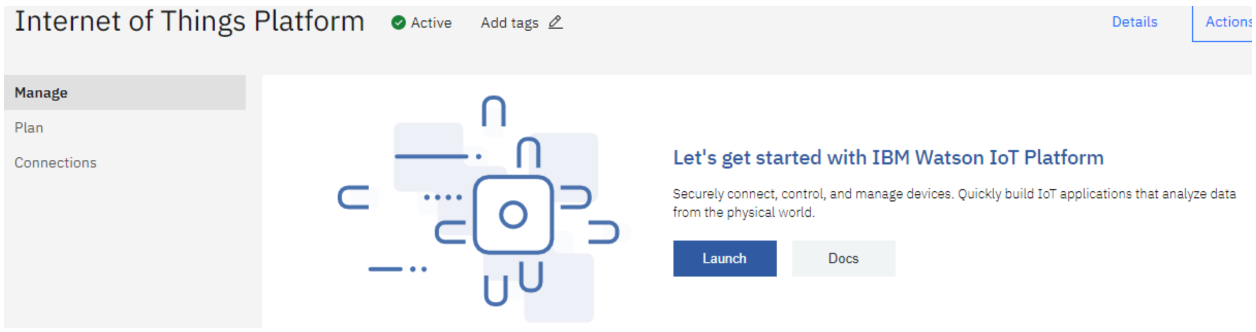

Start
Install software
Configure the Telegram Bot
Connect the sensors (Node red)
Develop the code
Upload the code
Test the output
End

Conclusion

The writer hereby concludes the project which states the existing problem and the proposed solution about smart homes with new technologies such as Artificial Intelligence and Internet of Things (IoT). Thus, by using smart home telegram bot, the dwellers may have a comfort zone as the workflow is automated.

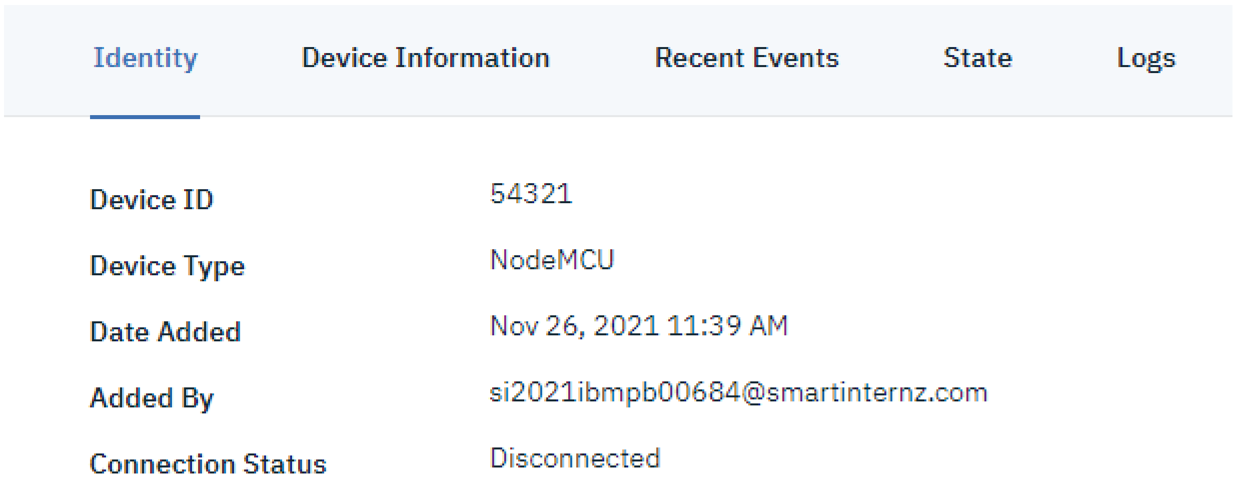
Appendix

IBM Watson IoT Platform

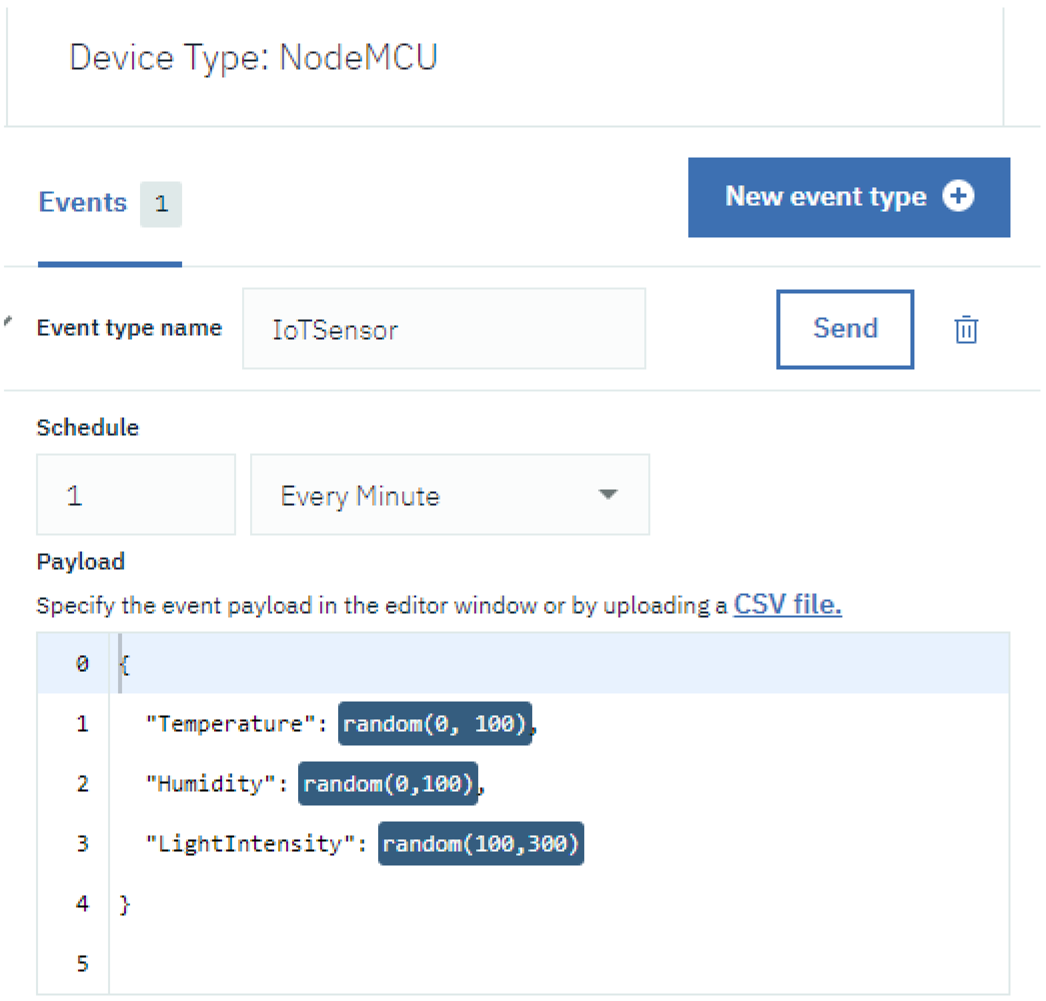


Let's get started with IBM Watson IoT Platform

Securely connect, control, and manage devices. Quickly build IoT applications that analyze data from the physical world.



Device ID	54321
Device Type	NodeMCU
Date Added	Nov 26, 2021 11:39 AM
Added By	si2021ibmpb00684@smartinternz.com
Connection Status	Disconnected



Event type name: IoTSensor

Schedule: 1 Every Minute

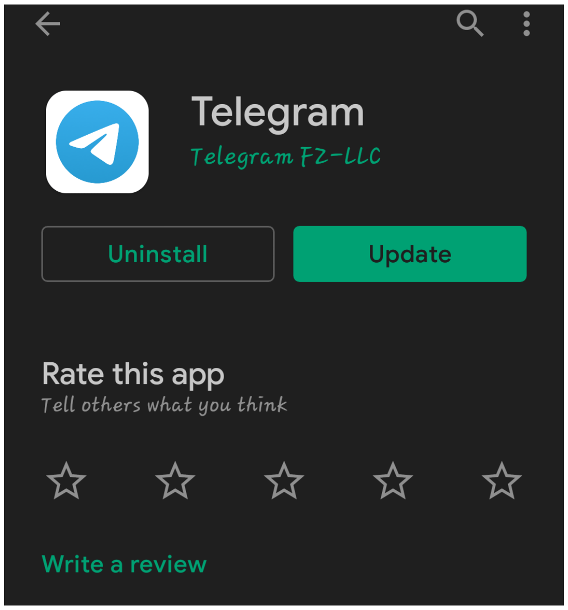
Payload

Specify the event payload in the editor window or by uploading a CSV file.

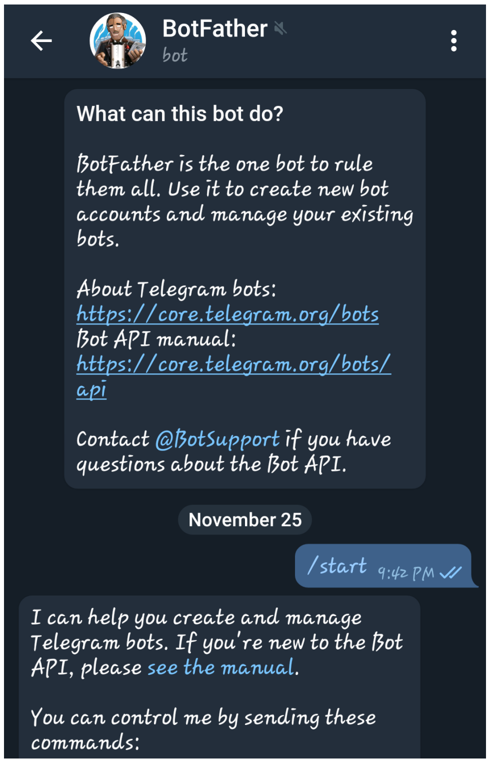
```
{
  "Temperature": random(0, 100),
  "Humidity": random(0,100),
  "LightIntensity": random(100,300)
}
```

Configuration of Telegram Bot

- **Telgram App**



- **Telegram Bot**



What can this bot do?

BotFather is the one bot to rule them all. Use it to create new bot accounts and manage your existing bots.

About Telegram bots: https://core.telegram.org/bots
Bot API manual: https://core.telegram.org/bots/api

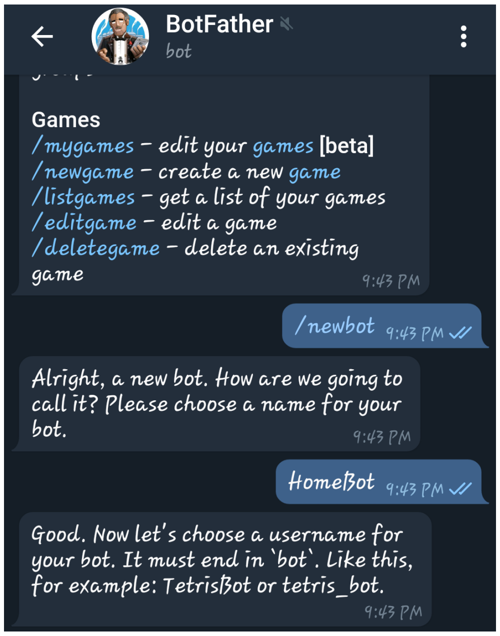
Contact @BotSupport if you have questions about the Bot API.

November 25

/start 9:42 PM

I can help you create and manage Telegram bots. If you're new to the Bot API, please see the manual.

You can control me by sending these commands:


BotFather
bot

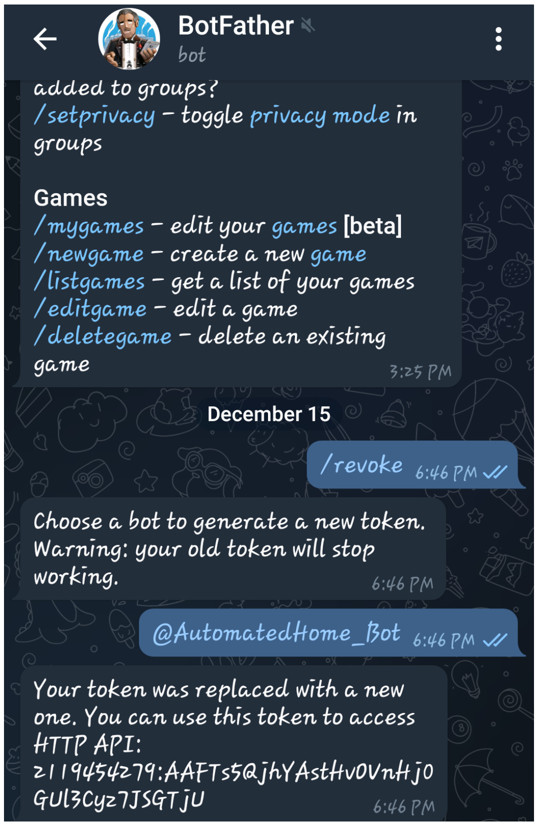
Games
/mygames - edit your games [beta]
/newgame - create a new game
/listgames - get a list of your games
/editgame - edit a game
/deletegame - delete an existing game
9:43 PM

/newbot 9:43 PM

Alright, a new bot. How are we going to call it? Please choose a name for your bot.
9:43 PM

HomeBot 9:43 PM

Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot.
9:43 PM


BotFather
bot

added to groups?
/setprivacy - toggle privacy mode in groups

Games
/mygames - edit your games [beta]
/newgame - create a new game
/listgames - get a list of your games
/editgame - edit a game
/deletegame - delete an existing game
3:25 PM

December 15

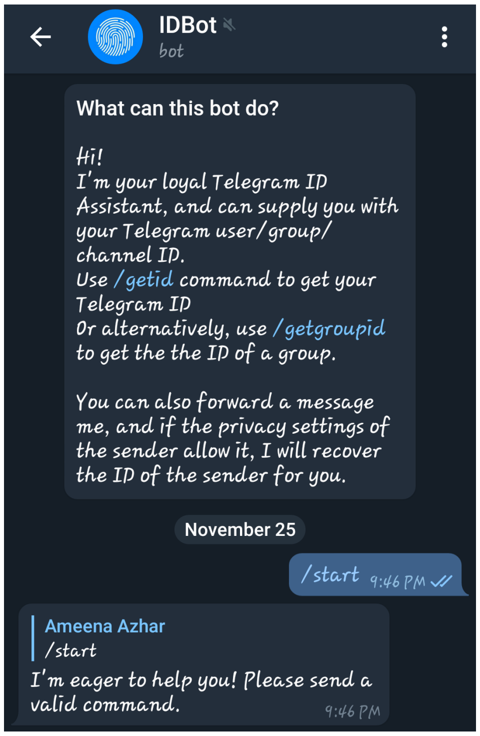
/revoke 6:46 PM

Choose a bot to generate a new token. Warning: your old token will stop working.
6:46 PM

@AutomatedHome_Bot 6:46 PM

Your token was replaced with a new one. You can use this token to access HTTP API:
2119454279:AAFTs5QjhYAstHv0VnHj0GUl3Cyz7JSGTjU
6:46 PM

Telegram UserId



IDBot
bot

What can this bot do?

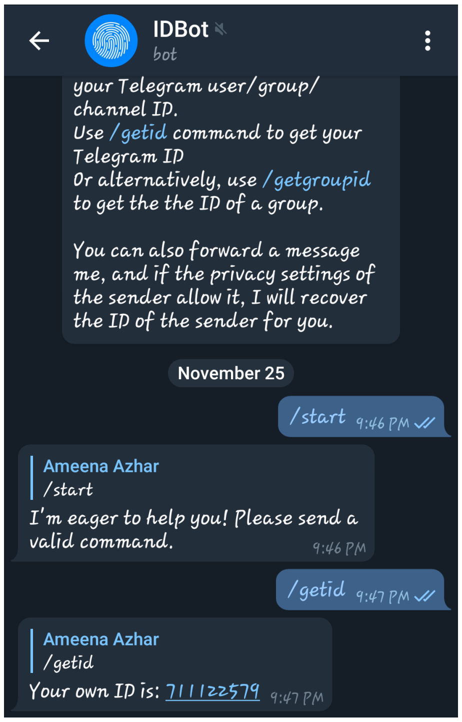
Hi!
I'm your loyal Telegram ID Assistant, and can supply you with your Telegram user/group/channel ID.
Use /getid command to get your Telegram ID
Or alternatively, use /getgroupid to get the the ID of a group.

You can also forward a message me, and if the privacy settings of the sender allow it, I will recover the ID of the sender for you.

November 25

/start 9:46 PM

Ameena Azhar
/start
I'm eager to help you! Please send a valid command. 9:46 PM



IDBot
bot

your Telegram user/group/channel ID.
Use /getid command to get your Telegram ID
Or alternatively, use /getgroupid to get the the ID of a group.

You can also forward a message me, and if the privacy settings of the sender allow it, I will recover the ID of the sender for you.

November 25

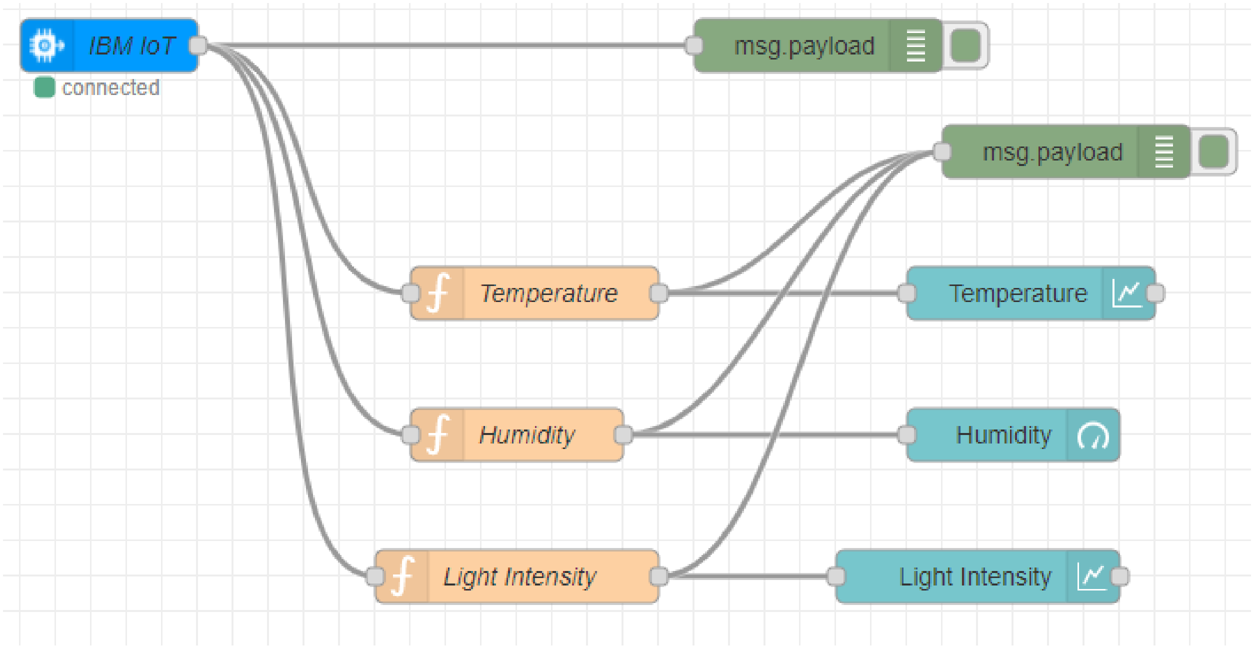
/start 9:46 PM

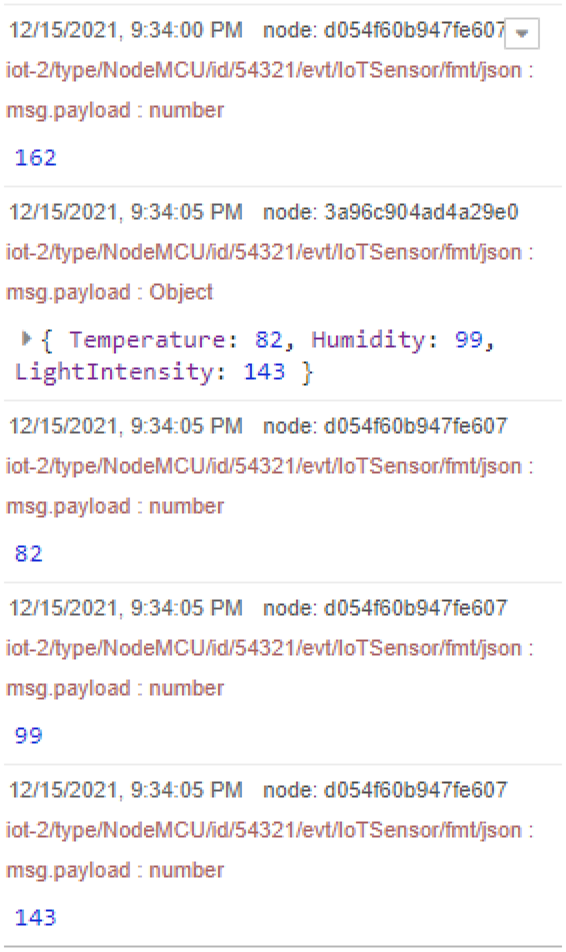
Ameena Azhar
/start
I'm eager to help you! Please send a valid command. 9:46 PM

/getid 9:47 PM

Ameena Azhar
/getid
Your own ID is: 711122579 9:47 PM

Node Red





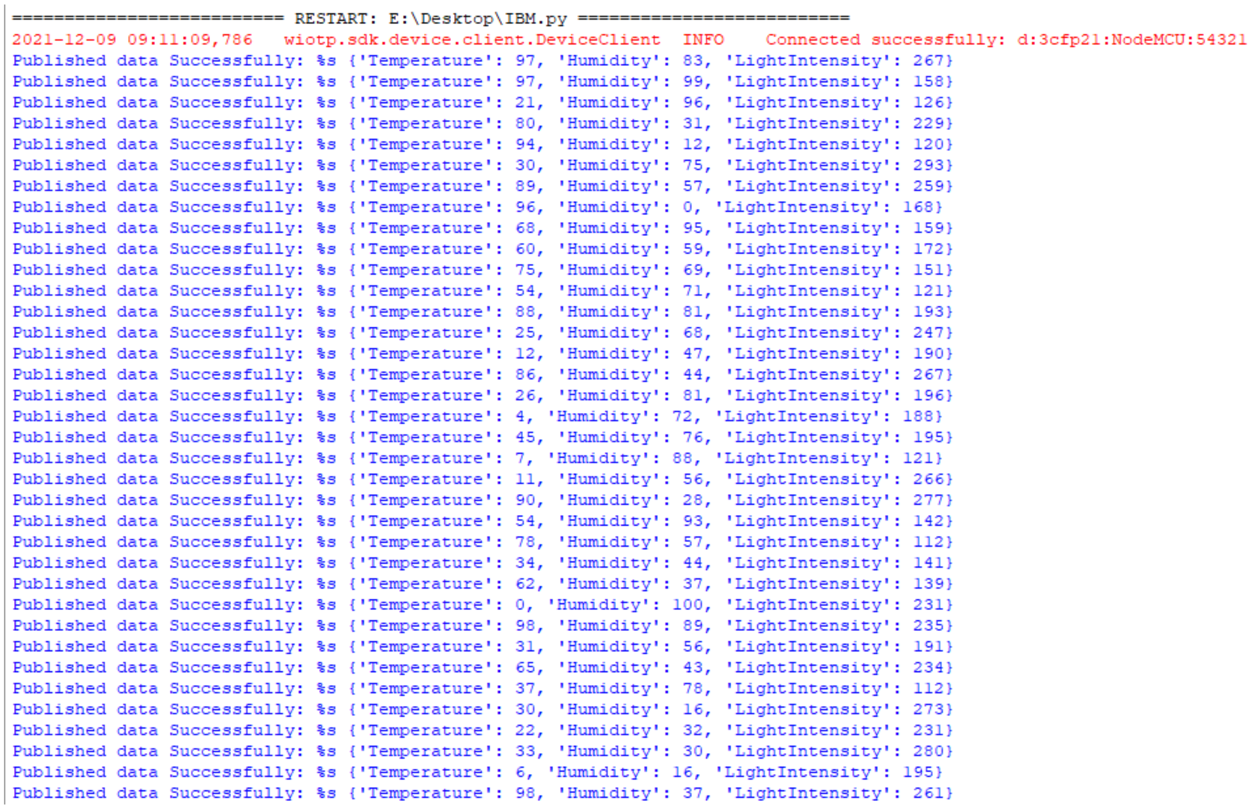
```
12/15/2021, 9:34:00 PM   node: d054f60b947fe607
iot-2/type/NodeMCU/id/54321/evt/IoTSensor/fmt/json :
msg.payload : number
 162
12/15/2021, 9:34:05 PM   node: 3a96c904ad4a29e0
iot-2/type/NodeMCU/id/54321/evt/IoTSensor/fmt/json :
msg.payload : Object
 ▸{ Temperature: 82, Humidity: 99,
 LightIntensity: 143 }
12/15/2021, 9:34:05 PM   node: d054f60b947fe607
iot-2/type/NodeMCU/id/54321/evt/IoTSensor/fmt/json :
msg.payload : number
 82
12/15/2021, 9:34:05 PM   node: d054f60b947fe607
iot-2/type/NodeMCU/id/54321/evt/IoTSensor/fmt/json :
msg.payload : number
 99
12/15/2021, 9:34:05 PM   node: d054f60b947fe607
iot-2/type/NodeMCU/id/54321/evt/IoTSensor/fmt/json :
msg.payload : number
 143
```

Code

IBM.py

```python
import wiotp.sdk.device
import time
import random
myConfig = {
    "identity": {
        "orgId": "3cfp21",
        "typeId": "NodeMCU",
        "deviceId":"54321"
    },
    "auth": {
        "token": "12345678"
    }
}

def myCommandCallback(cmd):
    print("Message received from IBM IoT Platform: %s" % cmd.data['command'])
    m=cmd.data['command']

client = wiotp.sdk.device.DeviceClient(config=myConfig, logHandlers=None)
client.connect()

while True:
    temp=random.randint(0,100)
    hum=random.randint(0,100)
    li=random.randint(100,300)
    myData={'Temperature':temp, 'Humidity':hum, 'LightIntensity':li}
    client.publishEvent(eventId="status", msgFormat="json", data=myData, qos=0, onPublish=None)
    print("Published data Successfully: %s", myData)
    client.commandCallback = myCommandCallback
    time.sleep(2)
client.disconnect()
```

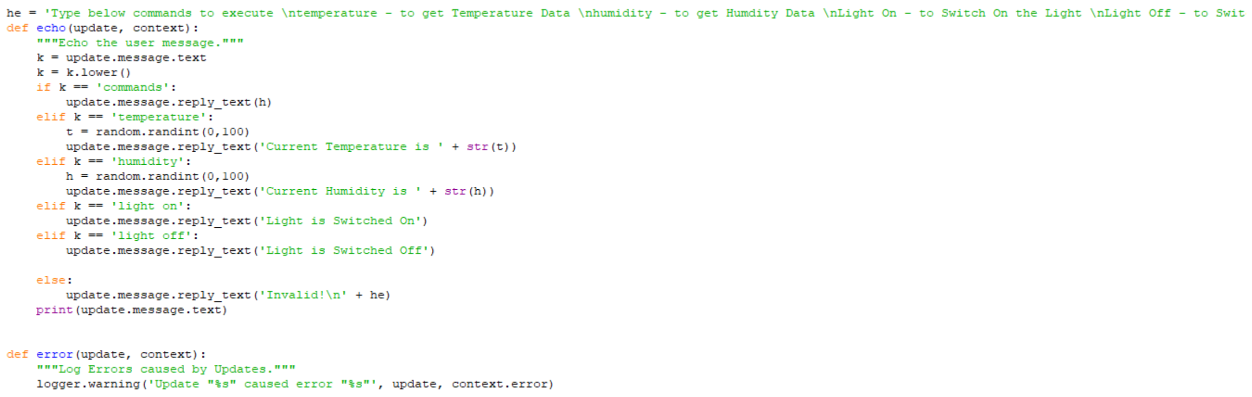
output



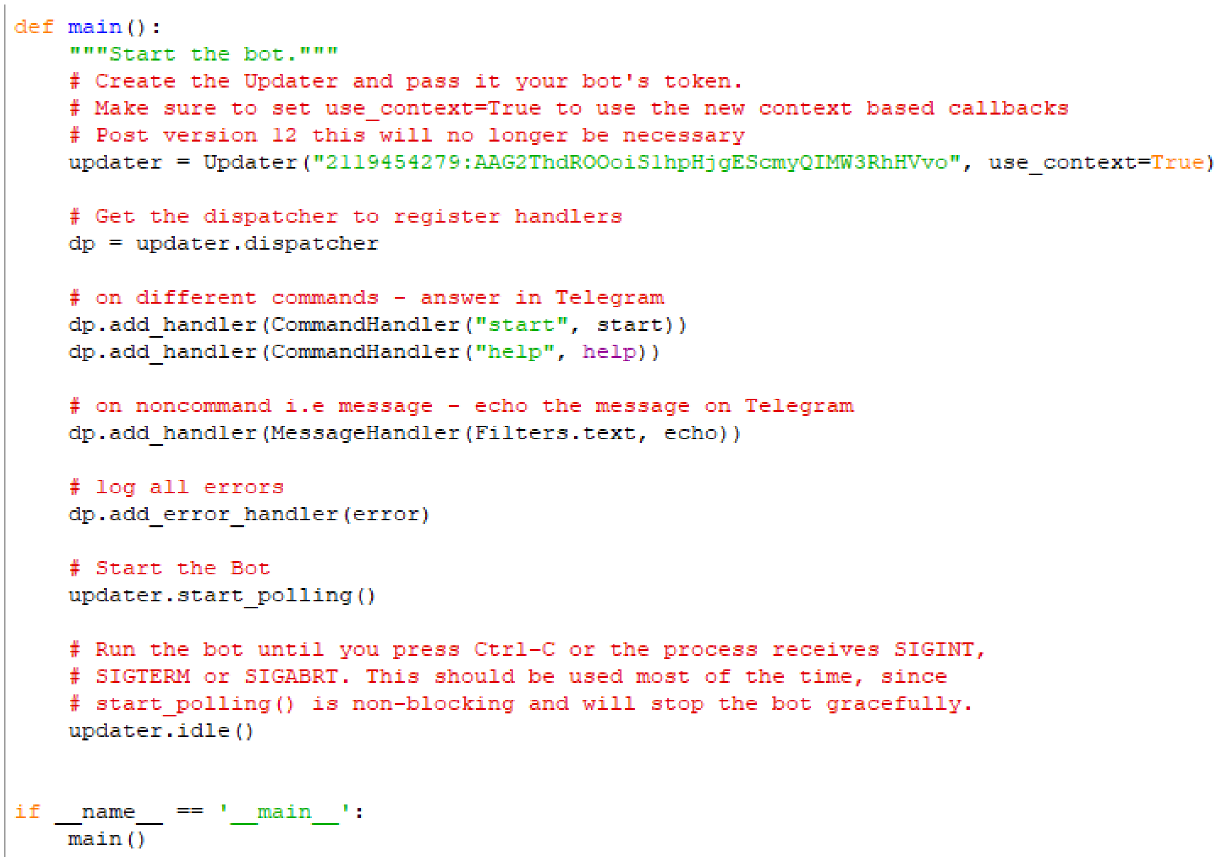
```
========================== RESTART: E:\Desktop\IBM.py ==========================
2021-12-09 09:11:09,786   wiotp.sdk.device.client.DeviceClient  INFO    Connected successfully: d:3cfp21:NodeMCU:54321
Published data Successfully: %s {'Temperature': 97, 'Humidity': 83, 'LightIntensity': 267}
Published data Successfully: %s {'Temperature': 97, 'Humidity': 99, 'LightIntensity': 158}
Published data Successfully: %s {'Temperature': 21, 'Humidity': 96, 'LightIntensity': 126}
Published data Successfully: %s {'Temperature': 80, 'Humidity': 31, 'LightIntensity': 229}
Published data Successfully: %s {'Temperature': 94, 'Humidity': 12, 'LightIntensity': 120}
Published data Successfully: %s {'Temperature': 30, 'Humidity': 75, 'LightIntensity': 293}
Published data Successfully: %s {'Temperature': 89, 'Humidity': 57, 'LightIntensity': 259}
Published data Successfully: %s {'Temperature': 96, 'Humidity': 0, 'LightIntensity': 168}
Published data Successfully: %s {'Temperature': 68, 'Humidity': 95, 'LightIntensity': 159}
Published data Successfully: %s {'Temperature': 60, 'Humidity': 59, 'LightIntensity': 172}
Published data Successfully: %s {'Temperature': 75, 'Humidity': 69, 'LightIntensity': 151}
Published data Successfully: %s {'Temperature': 54, 'Humidity': 71, 'LightIntensity': 121}
Published data Successfully: %s {'Temperature': 88, 'Humidity': 81, 'LightIntensity': 193}
Published data Successfully: %s {'Temperature': 25, 'Humidity': 68, 'LightIntensity': 247}
Published data Successfully: %s {'Temperature': 12, 'Humidity': 47, 'LightIntensity': 190}
Published data Successfully: %s {'Temperature': 86, 'Humidity': 44, 'LightIntensity': 267}
Published data Successfully: %s {'Temperature': 26, 'Humidity': 81, 'LightIntensity': 196}
Published data Successfully: %s {'Temperature': 4, 'Humidity': 72, 'LightIntensity': 188}
Published data Successfully: %s {'Temperature': 45, 'Humidity': 76, 'LightIntensity': 195}
Published data Successfully: %s {'Temperature': 7, 'Humidity': 88, 'LightIntensity': 121}
Published data Successfully: %s {'Temperature': 11, 'Humidity': 56, 'LightIntensity': 266}
Published data Successfully: %s {'Temperature': 90, 'Humidity': 28, 'LightIntensity': 277}
Published data Successfully: %s {'Temperature': 54, 'Humidity': 93, 'LightIntensity': 142}
Published data Successfully: %s {'Temperature': 78, 'Humidity': 57, 'LightIntensity': 112}
Published data Successfully: %s {'Temperature': 34, 'Humidity': 44, 'LightIntensity': 141}
Published data Successfully: %s {'Temperature': 62, 'Humidity': 37, 'LightIntensity': 139}
Published data Successfully: %s {'Temperature': 0, 'Humidity': 100, 'LightIntensity': 231}
Published data Successfully: %s {'Temperature': 98, 'Humidity': 89, 'LightIntensity': 235}
Published data Successfully: %s {'Temperature': 31, 'Humidity': 56, 'LightIntensity': 191}
Published data Successfully: %s {'Temperature': 65, 'Humidity': 43, 'LightIntensity': 234}
Published data Successfully: %s {'Temperature': 37, 'Humidity': 78, 'LightIntensity': 112}
Published data Successfully: %s {'Temperature': 30, 'Humidity': 16, 'LightIntensity': 273}
Published data Successfully: %s {'Temperature': 22, 'Humidity': 32, 'LightIntensity': 231}
Published data Successfully: %s {'Temperature': 33, 'Humidity': 30, 'LightIntensity': 280}
Published data Successfully: %s {'Temperature': 6, 'Humidity': 16, 'LightIntensity': 195}
Published data Successfully: %s {'Temperature': 98, 'Humidity': 37, 'LightIntensity': 261}
```

TelegramBot.py

```python
import logging
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters
import random

# Enable logging
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
                    level=logging.INFO)

logger = logging.getLogger(__name__)


# Define a few command handlers. These usually take the two arguments update and
# context. Error handlers also receive the raised TelegramError object in error.
def start(update, context):
    """Send a message when the command /start is issued."""
    update.message.reply_text('Hi!')

def help(update, context):
    """Send a message when the command /help is issued."""
    update.message.reply_text('Help!')
```

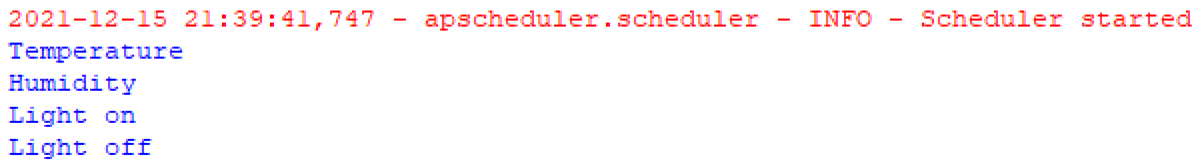


```
he = 'Type below commands to execute \ntemperature - to get Temperature Data \nhumidity - to get Humdity Data \nLight On - to Switch On the Light \nLight Off - to Swit
def echo(update, context):
    """Echo the user message."""
    k = update.message.text
    k = k.lower()
    if k == 'commands':
        update.message.reply_text(h)
    elif k == 'temperature':
        t = random.randint(0,100)
        update.message.reply_text('Current Temperature is ' + str(t))
    elif k == 'humidity':
        h = random.randint(0,100)
        update.message.reply_text('Current Humidity is ' + str(h))
    elif k == 'light on':
        update.message.reply_text('Light is Switched On')
    elif k == 'light off':
        update.message.reply_text('Light is Switched Off')

    else:
        update.message.reply_text('Invalid!\n' + he)
    print(update.message.text)


def error(update, context):
    """Log Errors caused by Updates."""
    logger.warning('Update "%s" caused error "%s"', update, context.error)
```

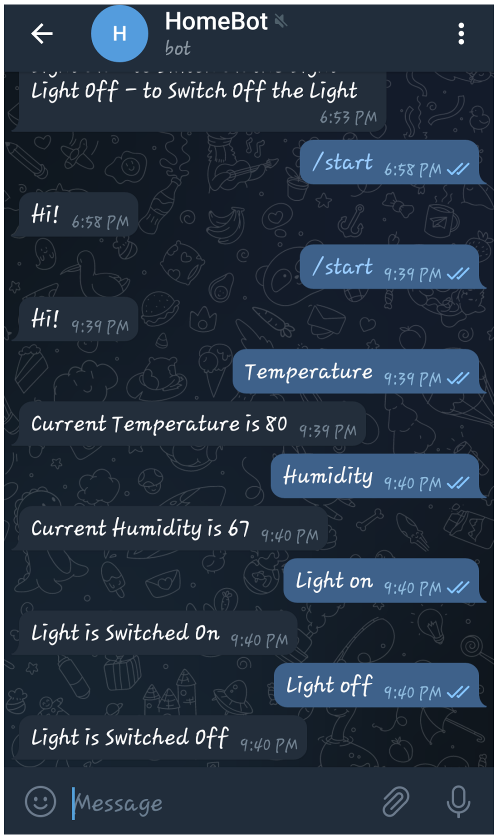


```
def main():
    """Start the bot."""
    # Create the Updater and pass it your bot's token.
    # Make sure to set use_context=True to use the new context based callbacks
    # Post version 12 this will no longer be necessary
    updater = Updater("2119454279:AAG2ThdROOoiSlhpHjgEScmyQIMW3RhHVvo", use_context=True)

    # Get the dispatcher to register handlers
    dp = updater.dispatcher

    # on different commands - answer in Telegram
    dp.add_handler(CommandHandler("start", start))
    dp.add_handler(CommandHandler("help", help))

    # on noncommand i.e message - echo the message on Telegram
    dp.add_handler(MessageHandler(Filters.text, echo))

    # log all errors
    dp.add_error_handler(error)

    # Start the Bot
    updater.start_polling()

    # Run the bot until you press Ctrl-C or the process receives SIGINT,
    # SIGTERM or SIGABRT. This should be used most of the time, since
    # start_polling() is non-blocking and will stop the bot gracefully.
    updater.idle()


if __name__ == '__main__':
    main()
```

Output



```
2021-12-15 21:39:41,747 - apscheduler.scheduler - INFO - Scheduler started
Temperature
Humidity
Light on
Light off
```



HomeBot
bot

Light Off – to Switch Off the Light 6:53 PM

/start 6:58 PM

Hi! 6:58 PM

/start 9:39 PM

Hi! 9:39 PM

Temperature 9:39 PM

Current Temperature is 80 9:39 PM

Humidity 9:40 PM

Current Humidity is 67 9:40 PM

Light on 9:40 PM

Light is Switched On 9:40 PM

Light off 9:40 PM

Light is Switched Off 9:40 PM

Integration of IBM code and Telegram code in Python

```python
import logging
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters
import wiotp.sdk.device
import time
import random


myConfig = {
    "identity": {
        "orgId": "3cfp21",
        "typeId": "NodeMCU",
        "deviceId":"54321"
    },
    "auth": {
        "token": "12345678"
    }
}

client = wiotp.sdk.device.DeviceClient(config=myConfig, logHandlers=None)
client.connect()

# Enable logging
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
                    level=logging.INFO)

logger = logging.getLogger(__name__)

def myCommandCallback(cmd):
    print("Message received from IBM IoT Platform: %s" % cmd.data['command'])
    m=cmd.data['command']

def start(update, context):
    """Send a message when the command /start is issued."""
    update.message.reply_text('Hi!')

def help(update, context):
    """Send a message when the command /help is issued."""
    update.message.reply_text('Help!')

he = 'Type below commands to execute \ntemperature - to get Temperature Data \nLight Intensity - to get Light Intensity Data \nhumidity - to get Humdity Data \nLight
def echo(update, context):
    """Echo the user message."""
    k = update.message.text
    k = k.lower()
    if k == 'commands':
        update.message.reply_text(h)
    elif k == 'temperature':
        t = random.randint(0,100)
        update.message.reply_text('Current Temperature is ' + str(t))
    elif k == 'humidity':
        h = random.randint(0,100)
        update.message.reply_text('Current Humidity is ' + str(h))
    elif k == 'Light Intensity':
        l = random.randint(100,300)
        update.message.reply_text('Current Light Intensity is ' + str(l))
    elif k == 'light on':
        update.message.reply_text('Light is Switched On')
    elif k == 'light off':
        update.message.reply_text('Light is Switched Off')

    else:
        update.message.reply_text('Invalid!\n' + he)
    print(update.message.text)


def error(update, context):
    """Log Errors caused by Updates."""
    logger.warning('Update "%s" caused error "%s"', update, context.error)
```

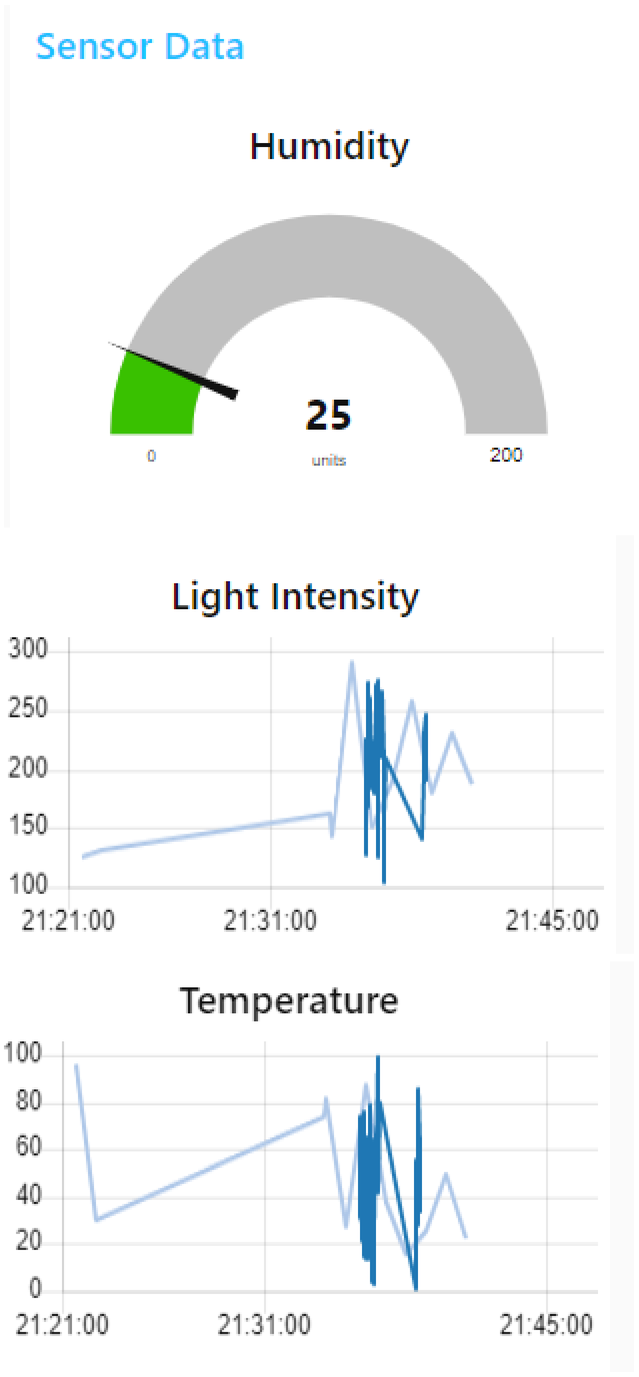
```
def main():

    """Start the bot."""
    # Create the Updater and pass it your bot's token.
    # Make sure to set use_context=True to use the new context based callbacks
    # Post version 12 this will no longer be necessary
    updater = Updater("2119454279:AAFTs5QjhYAstHvOVnHj0GUl3Cyz7JSGTjU", use_context=True)

    # Get the dispatcher to register handlers
    dp = updater.dispatcher


    # on different commands - answer in Telegram
    dp.add_handler(CommandHandler("start", start))
    dp.add_handler(CommandHandler("help", help))

    # on noncommand i.e message - echo the message on Telegram
    dp.add_handler(MessageHandler(Filters.text, echo))

    # log all errors
    dp.add_error_handler(error)

    # Start the Bot
    while True:
        temp=random.randint(0,100)
        hum=random.randint(0,100)
        li=random.randint(100,300)
        myData={'Temperature':temp, 'Humidity':hum, 'LightIntensity':li}
        client.publishEvent(eventId="status", msgFormat="json", data=myData, qos=0, onPublish=None)
        print("Published data Successfully: %s", myData)
        client.commandCallback = myCommandCallback
        time.sleep(2)
        updater.start_polling()
    #print(updater.start_polling())
```

```
    # Run the bot until you press Ctrl-C or the process receives SIGINT,
    # SIGTERM or SIGABRT. This should be used most of the time, since
    # start_polling() is non-blocking and will stop the bot gracefully.
    updater.idle()
    print("main")

if __name__ == '__main__':
    print("start")
    main()

client.disconnect()
```

output

```
2021-12-15 21:38:33,657   wiotp.sdk.device.client.DeviceClient  INFO    Connected successfully: d:3cfp21:NodeMCU:54321
start2021-12-15 21:38:33,657 - wiotp.sdk.device.client.DeviceClient - INFO - Connected successfully: d:3cfp21:NodeMCU:54321

Published data Successfully: %s {'Temperature': 1, 'Humidity': 21, 'LightIntensity': 141}
2021-12-15 21:38:35,699 - apscheduler.scheduler - INFO - Scheduler started
Published data Successfully: %s {'Temperature': 55, 'Humidity': 62, 'LightIntensity': 142}
Published data Successfully: %s {'Temperature': 28, 'Humidity': 12, 'LightIntensity': 196}
Published data Successfully: %s {'Temperature': 51, 'Humidity': 70, 'LightIntensity': 223}
Published data Successfully: %s {'Temperature': 85, 'Humidity': 56, 'LightIntensity': 231}
Published data Successfully: %s {'Temperature': 62, 'Humidity': 89, 'LightIntensity': 245}
Published data Successfully: %s {'Temperature': 33, 'Humidity': 64, 'LightIntensity': 190}
```

UI


Sensor Data
Humidity
25
units
0
200
Light Intensity
300
250
200
150
100
21:21:00
21:31:00
21:45:00
Temperature
100
80
60
40
20
0
21:21:00
21:31:00
21:45:00