6. REQUIREMENT ANALYSIS

Requirement Analysis is the first and important phase of the software developing activity in developing any kind of project effectively. I started to list out all the functionalities that my application should provide. There have been some minor changes with respect to the functionalities over the course of development.

6.1 SOFTWARE SPECIFICATIONS

• Operating System	: Windows 7 or higher
• Platform	: ThingSpeak IoT analytics cloud platform
• IDE	: Arduino Uno 1.8.4
• Programming language used	: C

6.2 HARDWARE SPECIFICATIONS

• Microcontroller	: Arduino Uno Board
• Sensors	: Temperature(LM35), Pulse Rate Sensor
• Processor	: Pentium IV or higher
• Processor speed	: 1.6GHz
• RAM	: 512 MB
• Disk Space	: 250 MB or higher

7. IMPLEMENTATION

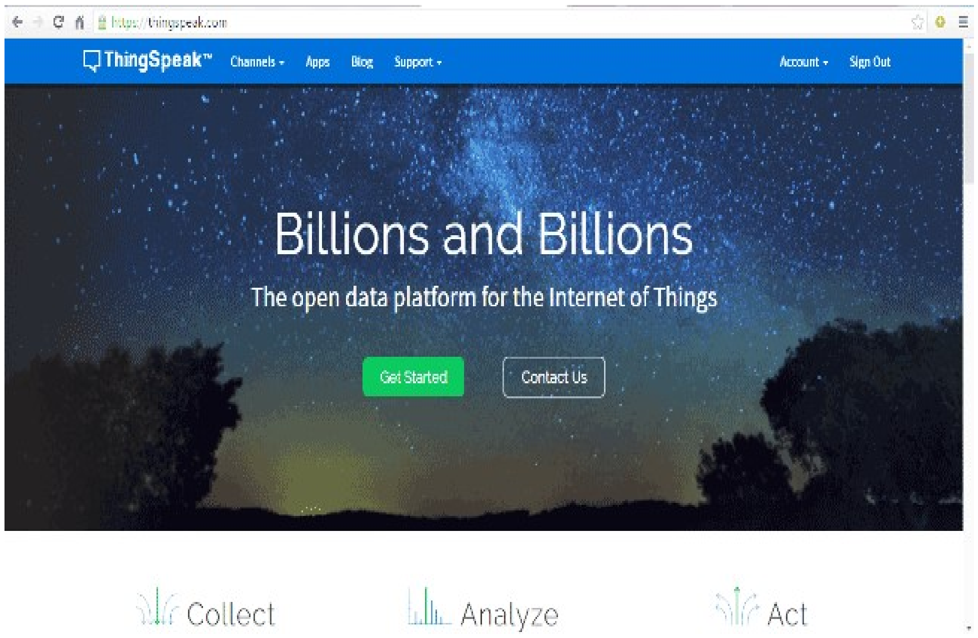
This project has been developed with Arduino microcontroller connected with sensors which are attached to the patient. All the sensors and location data sent from microcontroller to thingspeak IoT analytics platform. A doctor or guardian can log in to web portal to monitor patient's data at any point in time. In case of emergencies, like temperature spike or heartbeat spike, etc. an SMS and email alert sent to doctor and guardian's mobile and email respectively. And at any point of time either a doctor or guardian can log into web portal with patient unique credentials and can track patient's data which would help medical services to send appropriate help in case of emergencies.

7.1 CONFIGURING THINGSPEAK TO RECORD PATIENT DATA ONLINE

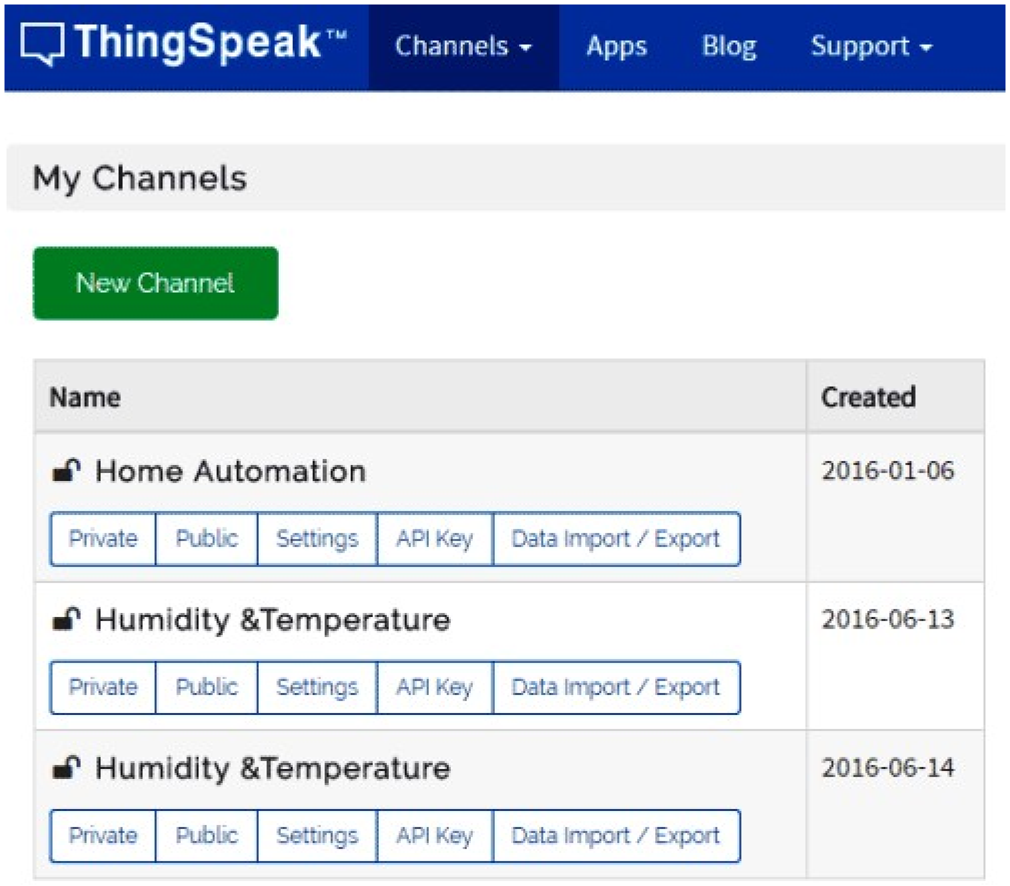
ThingSpeak provides very good tool for IoT based projects. By using ThingSpeak site, we can monitor our data and control our system over the Internet, using the Channels and web pages provided by ThingSpeak. ThingSpeak **'Collects'** the data from the sensors, **'Analyze and Visualize'** the data and **'Acts'** by triggering a reaction. We have previously used ThingSpeak in Weather station project using Raspberry Pi and using Arduino, check them to learn more about ThingSpeak. Here we are briefly explaining to use ThingSpeak for this **IoT Patient Monitoring Project.**

We will use **ThingSpeak** to monitor patient heartbeat and temperature online using internet. We will also use **IFTTT**platform to connect ThingSpeak to email/message service so that alert message can be sent whenever the patient is in critical state.

Step 1: First of all, user needs to Create a Account on ThingSpeak.com, then **Sign In and click On click**

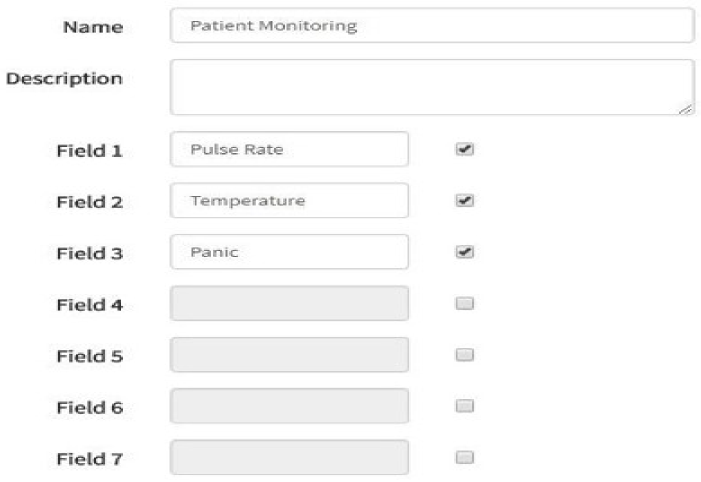


Step 2: Now go to the 'Channels' menu and click on **New Channel** option on the same page for further process.



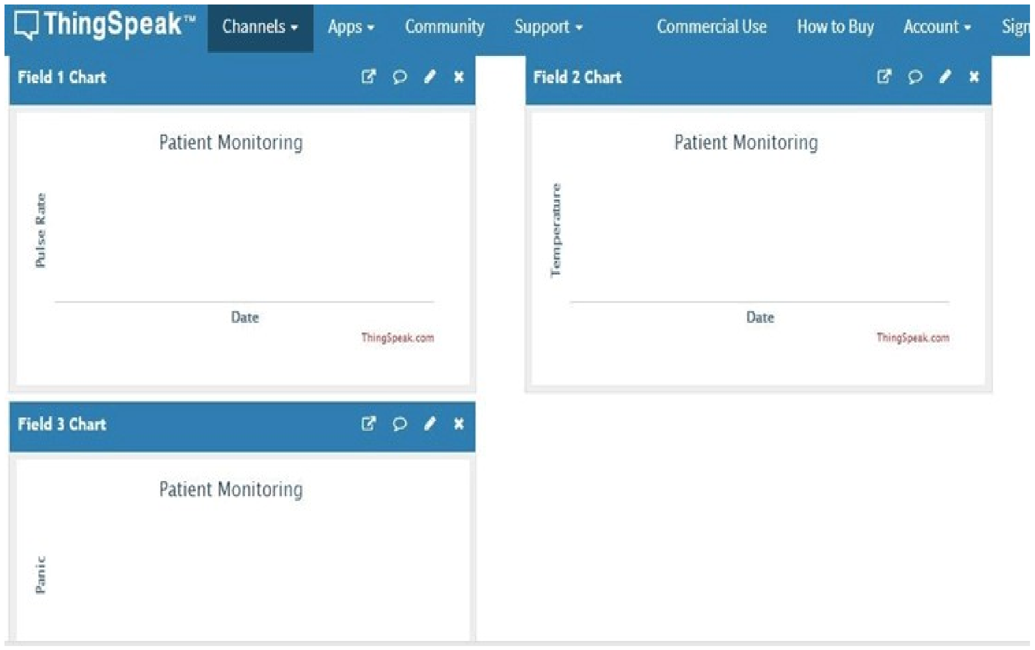
Name	Created
Home Automation Private | Public | Settings | API Key | Data Import / Export	2016-01-06
Humidity &Temperature Private | Public | Settings | API Key | Data Import / Export	2016-06-13
Humidity &Temperature Private | Public | Settings | API Key | Data Import / Export	2016-06-14

Step 3: Now you will see a form for **creating the channel**, fill in the Name and Description per your choice. Then fill 'Pulse Rate', 'Temperature' and 'Panic' in Field 1, Field 2 and Field 3 labels, tick the checkboxes for the Fields. Also tick the check box for 'Make Public' option below in the form and finally Save the Channel. Now your new channel has been created.

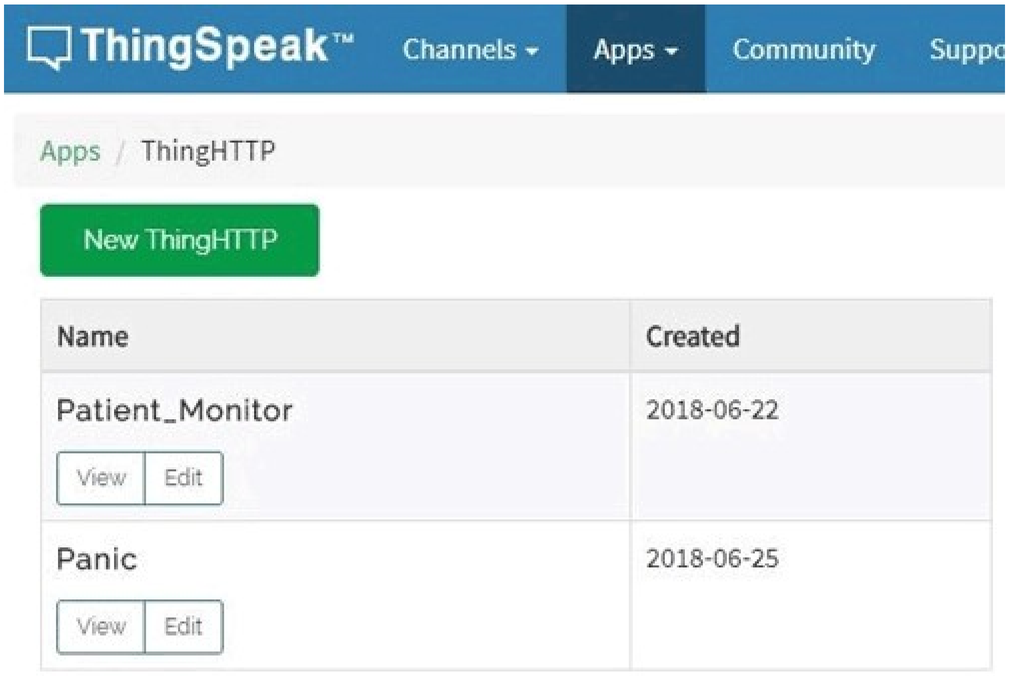


Name	Patient Monitoring	
Description		
Field 1	Pulse Rate	✓
Field 2	Temperature	✓
Field 3	Panic	✓
Field 4		☐
Field 5		☐
Field 6		☐
Field 7		☐

Step 4: You will see three charts as shown below. Note the **Write API key**, we will use this key In our code.



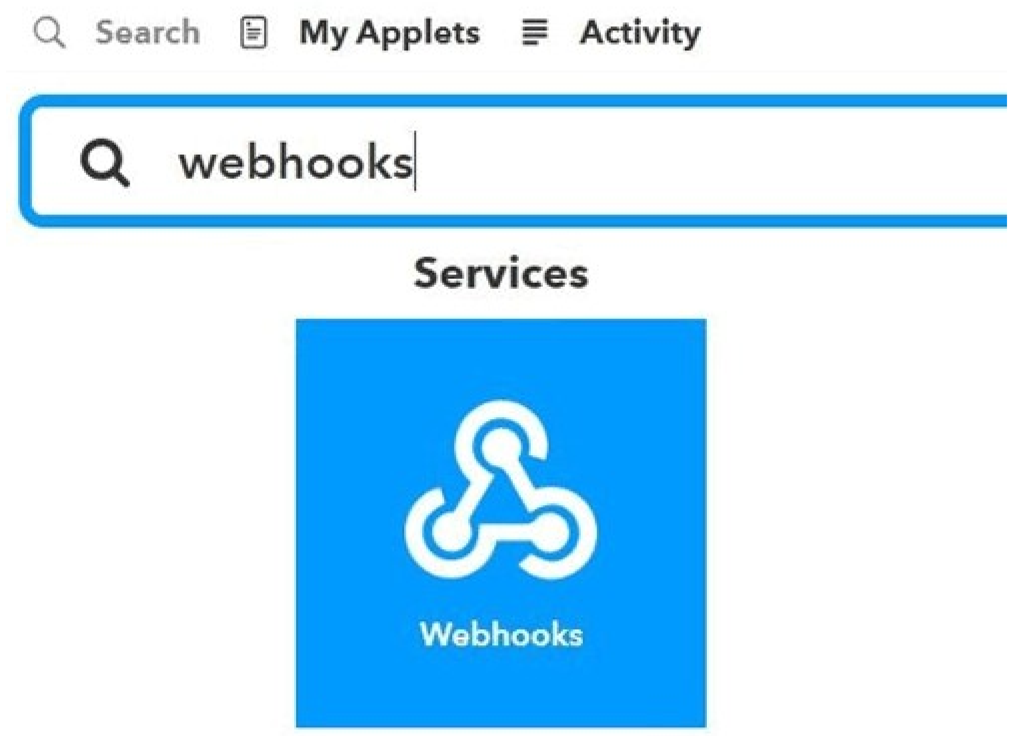
Step 5: Now, we will use ***ThingHTTP*** app of the server to trigger the IFTTT applet for data entry to Google sheets and send email/sms.



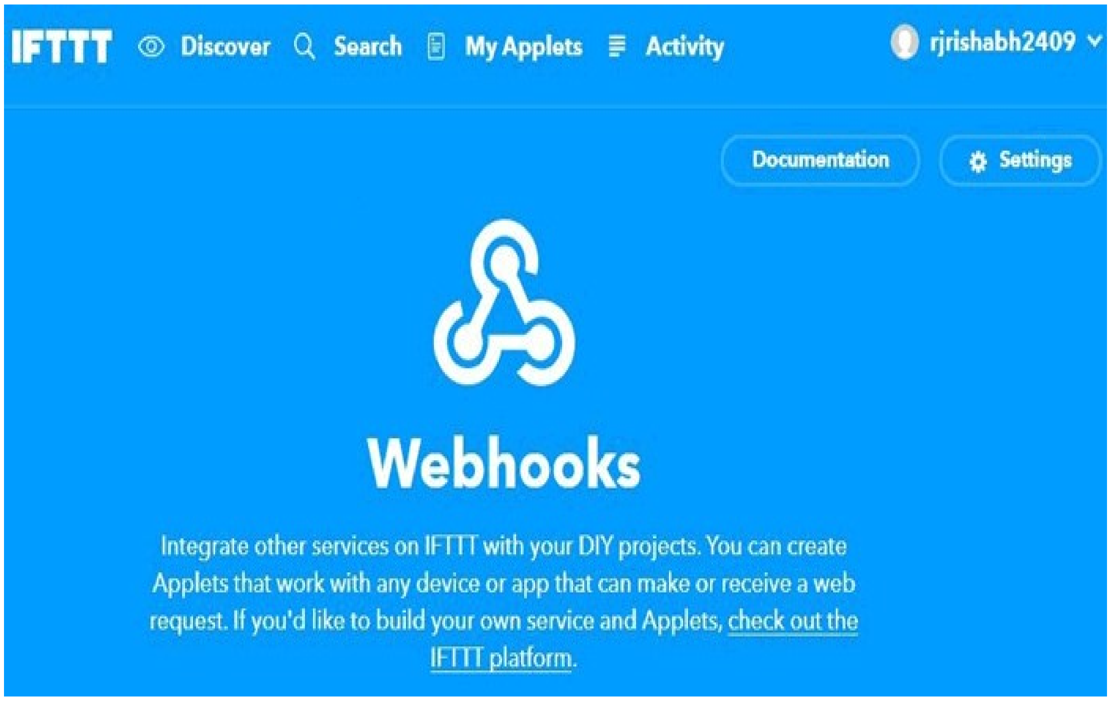
Name	Created
Patient_Monitor	2018-06-22
Panic	2018-06-25

7.2 CONFIGURING IFTTT FOR TRIGGERING MAIL/SMS BASED ON THINGSPEAK VALUES

Step 1: Login to IFTTT and search for **Webhooks** and click on it.

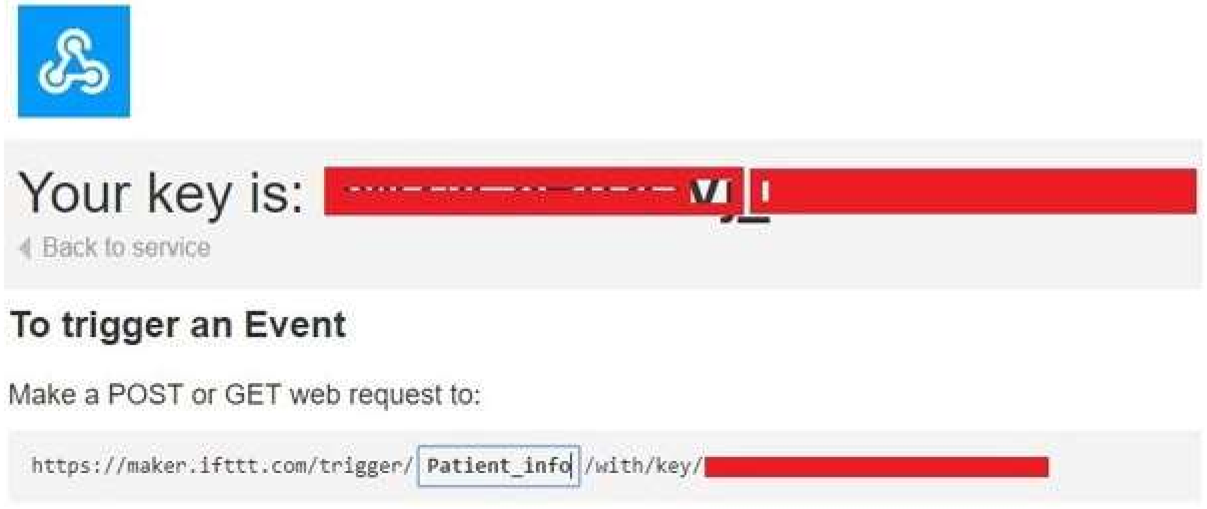


Step 2: Click on **Documentation**.



Integrate other services on IFTTT with your DIY projects. You can create Applets that work with any device or app that can make or receive a web request. If you'd like to build your own service and Applets, check out the IFTTT platform.

Step 3: Type "Patient Info" in the event box and copy the URL. We will use this URL in ThingHTTP.



Your key is: [redacted]

Back to service

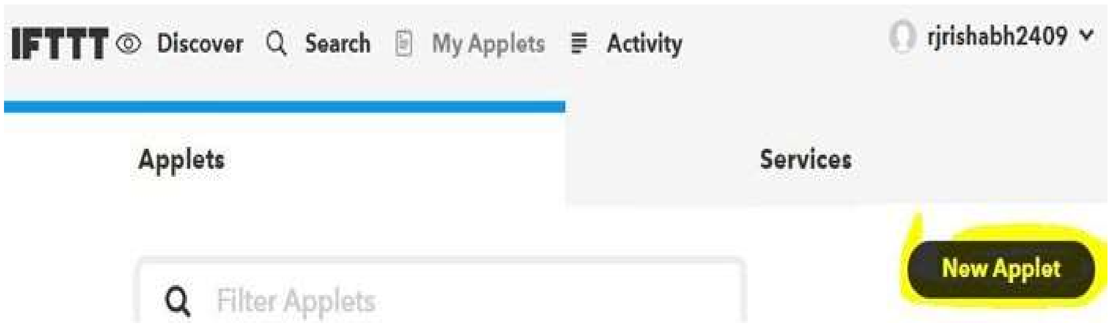
To trigger an Event

Make a POST or GET web request to:

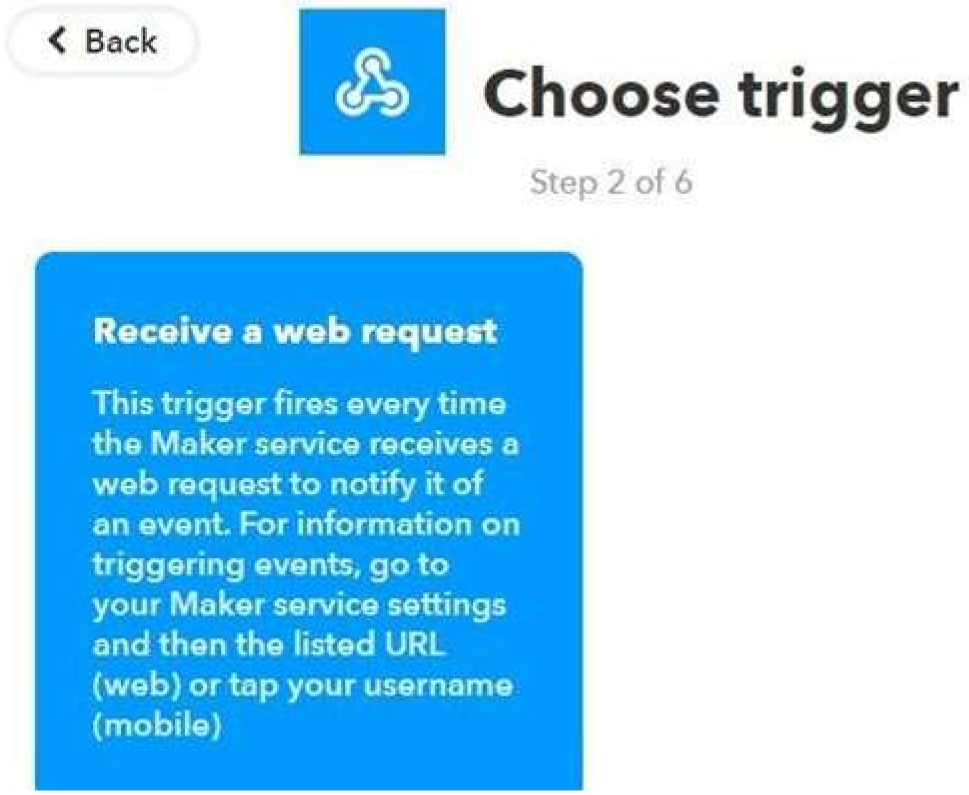
```
https://maker.ifttt.com/trigger/ Patient_info /with/key/[redacted]
```

Now let's make Applet to link ThingHTTP to Google sheet and to send email/sms. After that we will jump to complete our ThingHTTP.

Step 4: Click on *New* Applete in *My* Applete option.

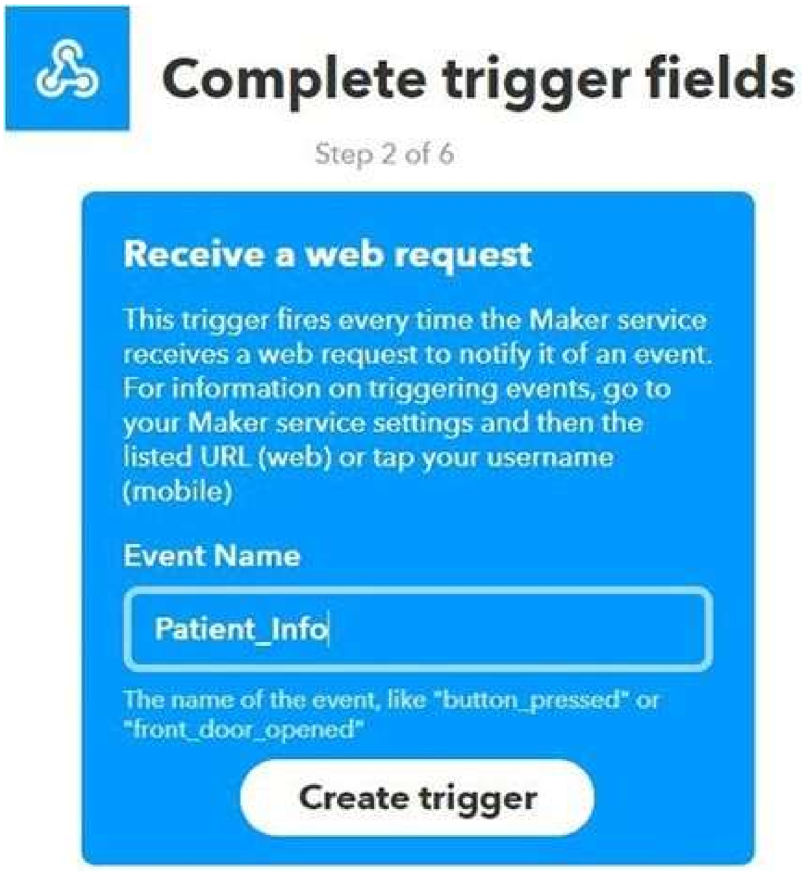


Step 5: Click on "+this" and search for Webhooks and click on it. Choose trigger as "Receive".



This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)

Step 6: Type the Event Name which is same as you write in the event box in webhooks URL.



Complete trigger fields

Step 2 of 6

Receive a web request

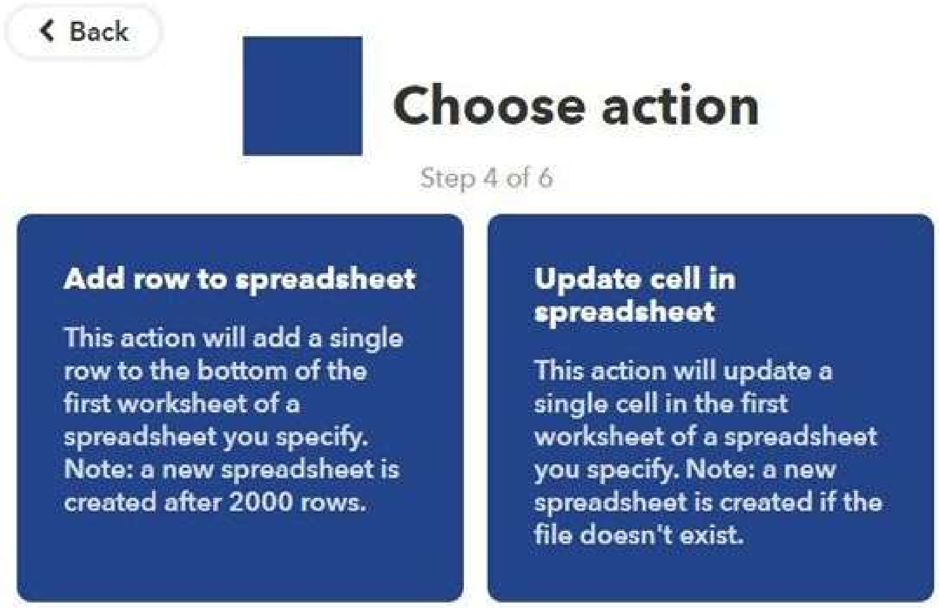
This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)

Event Name

Patient_Info

The name of the event, like "button_pressed" or "front_door_opened"

Step 7: Click on "+that" and search for Google Sheets and click on it.
Click on **Add row to spreadsheet**.



Choose action

Step 4 of 6

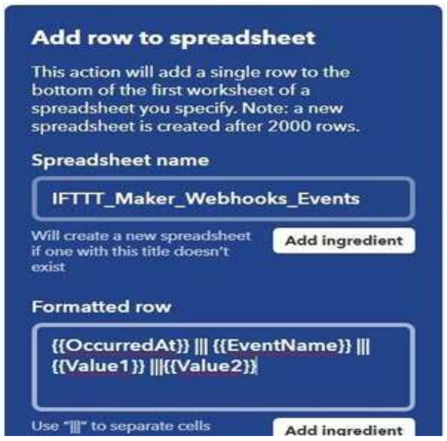
Add row to spreadsheet

This action will add a single row to the bottom of the first worksheet of a spreadsheet you specify. Note: a new spreadsheet is created after 2000 rows.

Update cell in spreadsheet

This action will update a single cell in the first worksheet of a spreadsheet you specify. Note: a new spreadsheet is created if the file doesn't exist.

Step 8: Give any name to your sheet. In formatted row box, you have date and time, event name, BPM value and body temperature which will be written as shown.


Add row to spreadsheet

This action will add a single row to the bottom of the first worksheet of a spreadsheet you specify. Note: a new spreadsheet is created after 2000 rows.

Spreadsheet name

IFTTT_Maker_Webhooks_Events

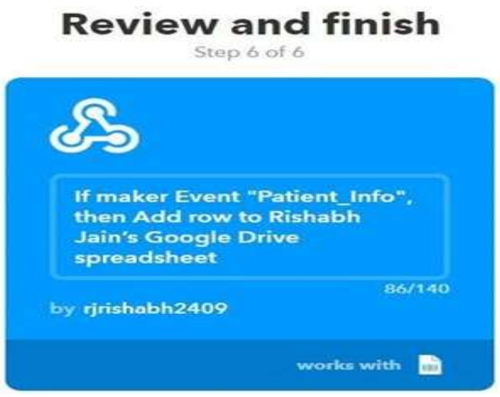
Will create a new spreadsheet if one with this title doesn't exist

Formatted row

{{OccurredAt}} ||| {{EventName}} ||| {{Value1}} |||{{Value2}}

Use "|||" to separate cells

Step 9: Review your applet and click on finish.


Review and finish

Step 6 of 6

If maker Event "Patient_Info", then Add row to Rishabh Jain's Google Drive spreadsheet

86/140

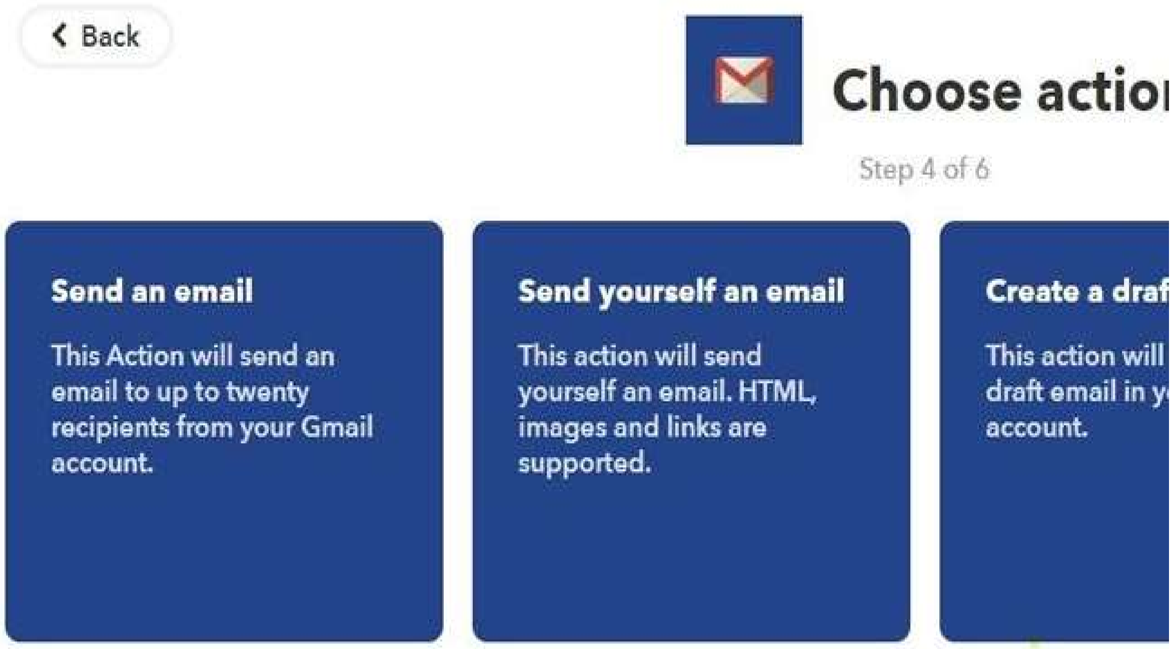
by rjrishabh2409

29

In the same way, **we have to make applet for sending email when Panic event is occurred**.

So again click on "+this" and select *Webhooks*, then in event name enter "Panic". In "+that" search for Gmail and click on it.

Now, click on Send an email.



Choose actio

Step 4 of 6

Send an email

This Action will send an email to up to twenty recipients from your Gmail account.

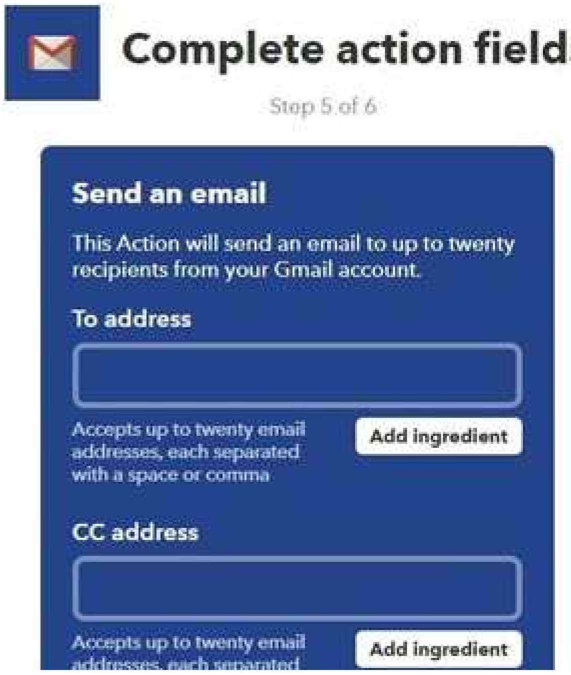
Send yourself an email

This action will send yourself an email. HTML, images and links are supported.

Create a draf

This action will draft email in y account.

Type the email addresses on which you wish to receive email when there is a panic to the patient.



Complete action fields

Step 5 of 6

Send an email

This Action will send an email to up to twenty recipients from your Gmail account.

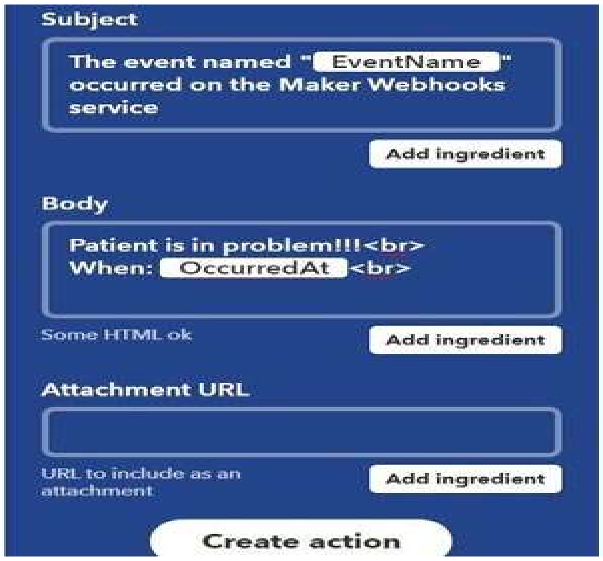
To address

Accepts up to twenty email addresses, each separated with a space or comma

CC address

Accepts up to twenty email addresses, each separated

Type the body content you wish to send in the email and click on create action. Review it and finish.


Subject

The event named " EventName " occurred on the Maker Webhooks service

Body

Patient is in problem!!!
When: OccurredAt

Some HTML ok

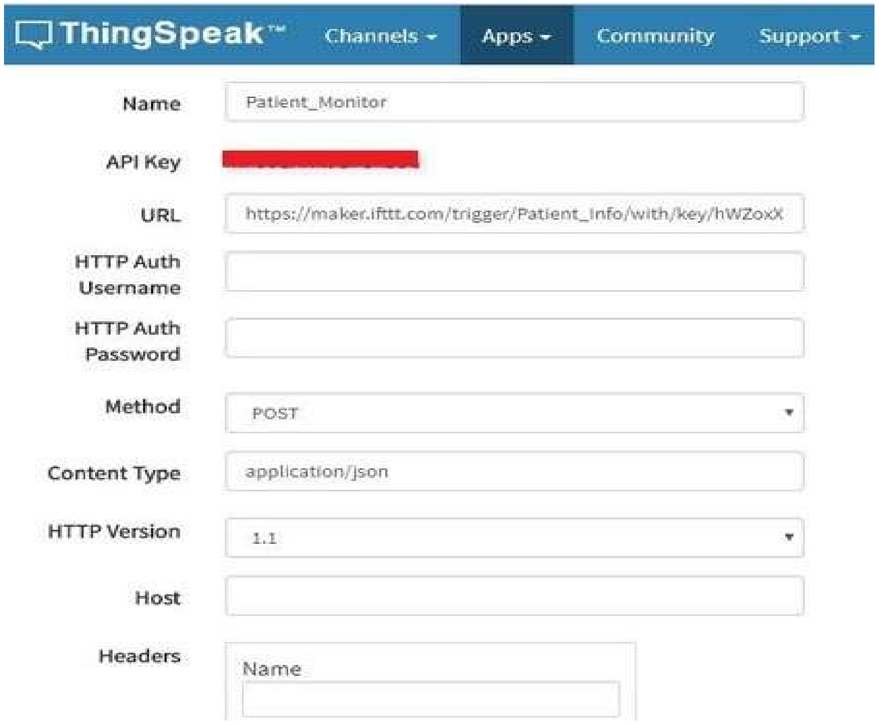
Attachment URL

URL to include as an attachment

We have made our applets to perform the tasks. Now, come back to *Thingspeak->Apps->ThingHTTP*.

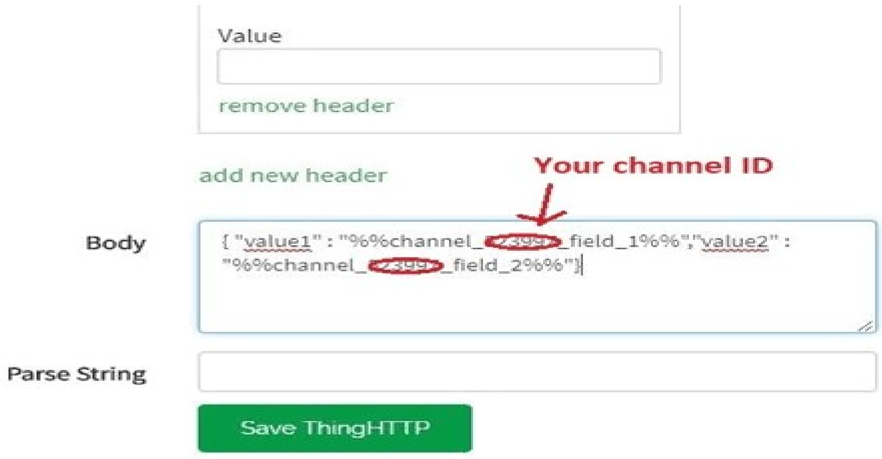
ThingHTTP for connecting ThingSpeak with IFTTT

Step 1: Click on *New ThingHTTP*. Give any name and Paste the URL that you copied from the webhooks documentation. Fill Remaining information as shown below.



Name	Patient_Monitor
API Key	[redacted]
URL	https://maker.ifttt.com/trigger/Patient_Info/with/key/hWZoxX
HTTP Auth Username	
HTTP Auth Password	
Method	POST
Content Type	application/json
HTTP Version	1.1
Host	
Headers	Name

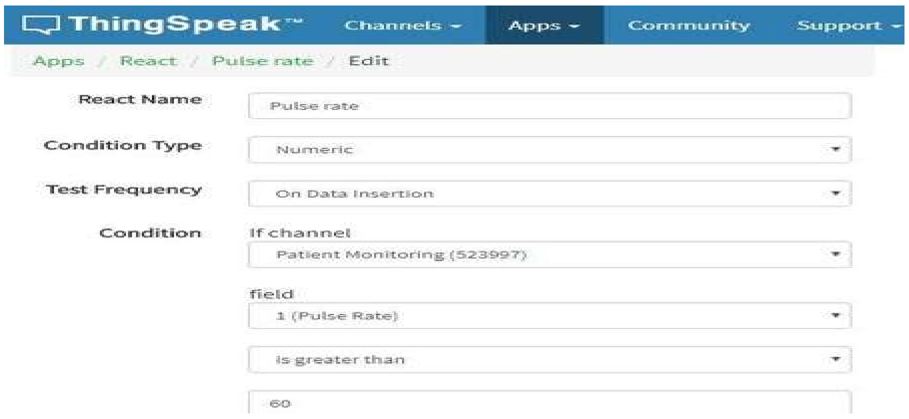
In Body, we have to write the information that we want to send to the IFTTT applet. We are sending patient Pulse reading and temperature. So the format is
{ "value1" : "%%channel_channelID_field_fieldNumber%%","value2" : "%%channel_channelID_field_fieldNumber%%"}



```
{ "value1" : "%%channel_2399_field_1%%","value2" :
"%%channel_2399_field_2%%"}
```

After filling these informations, click on *Save ThingHTTP*.
In the same manner, we have to make ThingHTTP for “Panic”. Follow the same steps.
In URL, write **Panic** in place of **Patient_Info.** Body remains empty and All other information are same as in previous ThingHTTP. Save it.
Now, we have to make *React* to trigger the URL.
React works with ThingHTTP app to perform actions when channel data meets a certain condition. To make React, click on Apps -> React. Click on **New React**.

Step 2: Give name to your React. Condition type as Numeric and Test Freaquency as on Data Insertion.
Choose the Condition on which you want to trigger the URL. Select your channel from the *If Channel* drop down menu. Choose field 1 i.e Pulse rate and make condition of *greater than* any value. I have used 60. As shown



React Name: Pulse rate

Condition Type: Numeric

Test Frequency: On Data Insertion

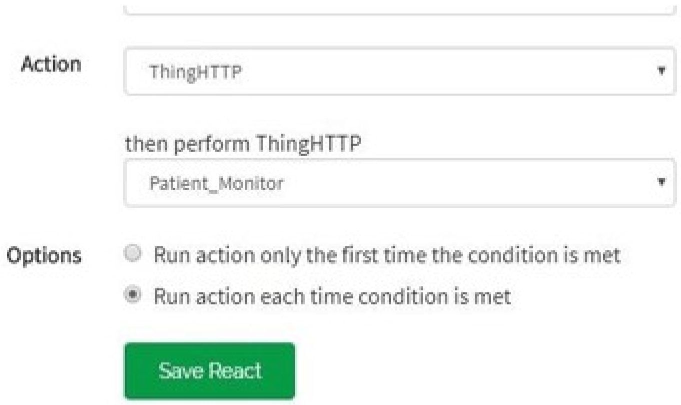
Condition: If channel Patient Monitoring (523997)

field 1 (Pulse Rate)

is greater than

60

Choose **ThingHTTP** from Action drop down menu and select the **ThingHTTP**. Select “Run action each time condition is met” and click on *Save React*.

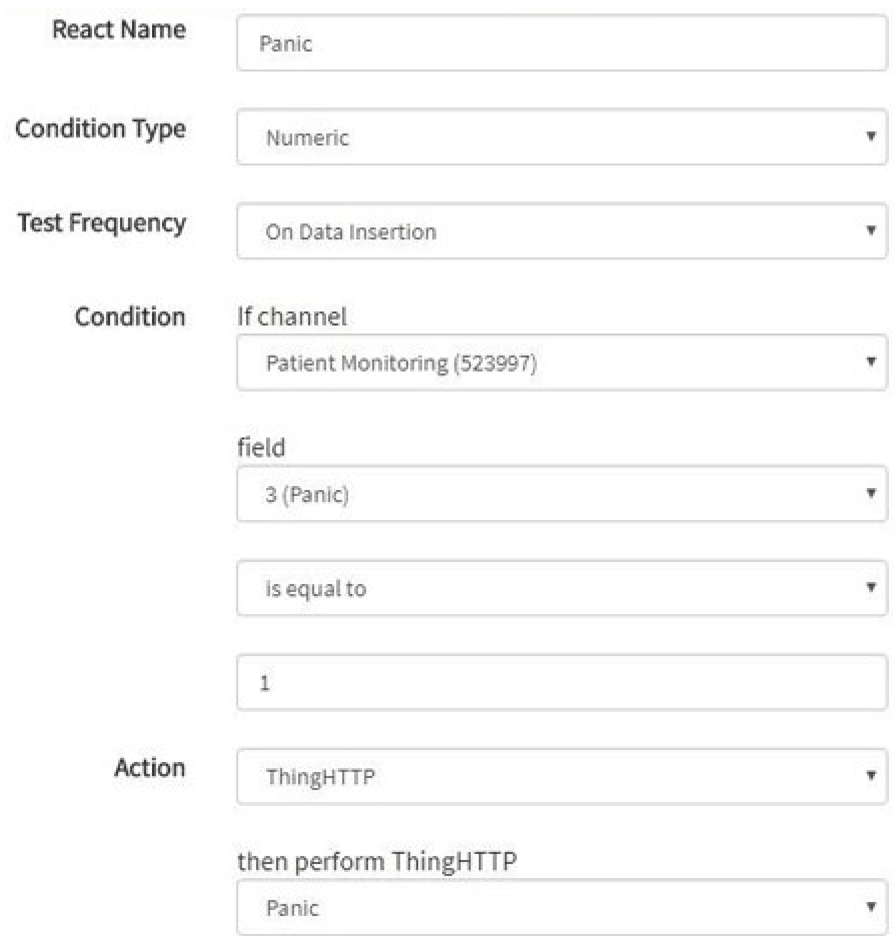


Action: ThingHTTP

then perform ThingHTTP: Patient_Monitor

Options: Run action only the first time the condition is met [✓ Run action each time condition is met]

In the same way, make React for the **Panic** as shown.


React Name: Panic
Condition Type: Numeric
Test Frequency: On Data Insertion
Condition: If channel: Patient Monitoring (523997)
field: 3 (Panic)
is equal to
1
Action: ThingHTTP
then perform ThingHTTP: Panic

Select “Run action each time condition is met” and click on Save React.

7.3 ARDUINO UNO CODE

First, we include all the libraries. We are using software serial to communicate with esp8266.

```
#include <SoftwareSerial.h>
#include "Timer.h"
#include <PulseSensorPlayground.h>     //pulse sensor library
```

Make instance for timer, SoftwareSerial and pulse sensor to use in our code.

```
Timer t;
PulseSensorPlayground pulseSensor;
SoftwareSerial esp8266(10,11);   //Rx,Tx
```

Set-up low-level interrupts for most accurate BPM match and enable DEBUG to show ongoing commands on serial monitor.

```
#define USE_ARDUINO_INTERRUPTS true
#define DEBUG true
```

Set your WiFi name , password and IP of thingspeak.com

```
#define SSID "*********"     // "your WiFiname"
#define PASS "**********"   // "wifi password"
#define IP "184.106.153.149"     // thingspeak.com ip
```

Declare String to update information on ThingSpeak channel. You will need API key for this, which can be found from your ThingSpeak channel-> API key . Copy **Write API key** and paste here.

```
String msg = "GET /update?key=Your Api Key";
```

In *setup* function, set the baud rate for serial communication between Arduino serial monitor and esp8266. Start the ESP communication by giving AT command to it and connect it by calling ***connectWiFi();*** function. After that we will initialize Timers by calling ***t.every(time_interval, do_this);*** which will take the readings of the sensors and update on the channel after every *time_interval* you defined.

```
void setup()
{
 Serial.begin(9600);
 esp8266.begin(115200);
  pulseSensor.analogInput(PulseWire);
 pulseSensor.blinkOnPulse(LED13);      //auto-magically blink Arduino's LED with heartbeat.
 pulseSensor.setThreshold(Threshold);
 // Double-check the "pulseSensor" object was created and "began" seeing a signal.
 if (pulseSensor.begin()) {
  Serial.println("We created a pulseSensor Object !");
 }
 Serial.println("AT");
 esp8266.println("AT");
 delay(3000);

 if(esp8266.find("OK"))
 {
  connectWiFi();
 }
 t.every(10000, getReadings);
```

```
  t.every(10000, updateInfo);
}
```

We have to make functions
for **connectWiFi()**, **panic_button()**, **update_info()** and **getReadings()**.

Make function for *connectWiFi()* which will return True or False depending upon Wi-Fi connected or not. **AT+CWMODE=1** will make ESP8266 work in *station mode*. **AT+CWJAP=\,** command, used in this function, is to connect to your Access Point (your Wi-Fi router).

```
boolean connectWiFi()
{
 Serial.println("AT+CWMODE=1");
 esp8266.println("AT+CWMODE=1");
 delay(2000);
 String cmd="AT+CWJAP=\"";
 cmd+=SSID;
 cmd+="\",\"";
 cmd+=PASS;
 cmd+="\"";
 Serial.println(cmd);
 esp8266.println(cmd);
……
…..
```

Make ***getReadings();*** function to take pulse sensor and LM35 readings and convert them to string using ***dtostrf();***function**.**

```
void getReadings(){
  raw_myTemp = analogRead(A1);
 Voltage = (raw_myTemp / 1023.0) * 5000;        // 5000 to get millivots.
 tempC = Voltage * 0.1;              //in degree C
 myTemp = (tempC * 1.8) + 32;            // conver to F
 Serial.println(myTemp);
 int myBPM = pulseSensor.getBeatsPerMinute();  // Calls function on our pulseSensor
object that returns BPM as an "int".
if (pulseSensor.sawStartOfBeat()) {           // Constantly test to see if "a beat happened".
Serial.println(myBPM);                      // Print the value inside of myBPM.
}
delay(20);
```

Define char array for BPM and temp and convert float value of these sensors to String using **dtostrf().**

```
   char buffer1[10];
    char buffer2[10];
   BPM = dtostrf(myBPM, 4, 1, buffer1);
   temp = dtostrf(myTemp, 4, 1, buffer2);
 }
```

Make function for updating sensor information on the ThingSpeak channel.
"AT+CIPSTART=\"TCP\",\"", AT Command will establish TCP command over port 80

```
void updateInfo()
{
  String cmd = "AT+CIPSTART=\"TCP\",\"";
  cmd += IP;
  cmd += "\",80";
  Serial.println(cmd);
  esp8266.println(cmd);
  delay(2000);
  if(esp8266.find("Error"))
  {
    return;
  }
```

Attach the readings with the GET URL using "&field1="; for pulse readings and "&field2="; for temperature readings. Send this information using "AT+CIPSEND=" command.

```
  cmd = msg ;
  cmd += "&field1=";    //field 1 for BPM
  cmd += BPM;
  cmd += "&field2=";  //field 2 for temperature
  cmd += temp;
  cmd += "\r\n";
  Serial.print("AT+CIPSEND=");
  esp8266.print("AT+CIPSEND=");
  Serial.println(cmd.length());
  esp8266.println(cmd.length());
  if(esp8266.find(">"))
  {
    Serial.print(cmd);
    esp8266.print(cmd);
  }
```

…
…

Similarly, make function for *panic_button*. When button goes to HIGH, esp8266 send the information to the server using AT+CIPSTART and AT+CIPSEND commands.

```
void panic_button(){
  panic = digitalRead(8);
    if(panic == HIGH){
    Serial.println(panic);
      String cmd = "AT+CIPSTART=\"TCP\",\"";
cmd += IP;
  cmd += "\",80";
  Serial.println(cmd);
  esp8266.println(cmd);
…..
..
```

Attach this information to "&field3=".

```
  cmd = msg ;
  cmd += "&field3=";
```

In *loop* function, call *panic_button()* and timers using *t.update()* function .

```
void loop()
{
  panic_button();
start: //label
    error=0;
    t.update();
……
……
```

7.4 Complete Arduino Code

```
#define USE_ARDUINO_INTERRUPTS true
#define DEBUG true
#define SSID "JioFi_10C0164"     // "SSID-WiFiname"
#define PASS "fr4ge0ine6" // "password"
#define IP "184.106.153.149"      // thingspeak.com ip

#include <SoftwareSerial.h>
#include "Timer.h"
```

```
#include <PulseSensorPlayground.h>     // Includes the PulseSensorPlayground Library.
Timer t;
PulseSensorPlayground pulseSensor;

String msg = "GET
https://api.thingspeak.com/update?api_key=O1MLT6Z2UQDJAUL3&field1=0";
SoftwareSerial esp8266(10,11);
//Variables
const int PulseWire = A0;       // PulseSensor PURPLE WIRE connected to ANALOG PIN 0
const int LED13 = 13;          // The on-board Arduino LED, close to PIN 13.
int Threshold = 550;           //for heart rate sensor
float myTemp;
int myBPM;
String BPM;
String temp;
int error;
int panic;
int raw_myTemp;
float Voltage;
float tempC;
void setup()
{

  Serial.begin(9600);
  esp8266.begin(115200);
  pulseSensor.analogInput(PulseWire);
  pulseSensor.blinkOnPulse(LED13);       //auto-magically blink Arduino's LED with heartbeat.
  pulseSensor.setThreshold(Threshold);

  // Double-check the "pulseSensor" object was created and "began" seeing a signal.
   if (pulseSensor.begin()) {
    Serial.println("We created a pulseSensor Object !");  //This prints one time at Arduino power-
up,  or on Arduino reset.
  }
  Serial.println("AT");
  esp8266.println("AT");

  delay(3000);

  if(esp8266.find("OK"))
  {
    connectWiFi();
  }
  t.every(10000, getReadings);
   t.every(10000, updateInfo);
}

void loop()
{
  panic_button();
start: //label
    error=0;
    t.update();
    //Resend if transmission is not completed
    if (error==1)
    {
      goto start; //go to label "start"
    }
 delay(4000);
}

void updateInfo()
{
  String cmd = "AT+CIPSTART=\"TCP\",\"";
  cmd += IP;
  cmd += "\",80";
  Serial.println(cmd);
  esp8266.println(cmd);
  delay(2000);
  if(esp8266.find("Error"))
  {
    return;
  }
  cmd = msg ;
  cmd += "&field1=";    //field 1 for BPM
  cmd += BPM;
  cmd += "&field2=";  //field 2 for temperature
  cmd += temp;
  cmd += "\r\n";
  Serial.print("AT+CIPSEND=");
  esp8266.print("AT+CIPSEND=");
  Serial.println(cmd.length());
  esp8266.println(cmd.length());
  if(esp8266.find(">"))
  {
    Serial.print(cmd);
    esp8266.print(cmd);
  }
  else
  {
    Serial.println("AT+CIPCLOSE");
    esp8266.println("AT+CIPCLOSE");
    //Resend...
    error=1;
  }
}
boolean connectWiFi()
{
  Serial.println("AT+CWMODE=1");
  esp8266.println("AT+CWMODE=1");
  delay(2000);
  String cmd="AT+CWJAP=\"";
  cmd+=SSID;
  cmd+="\",\"";
  cmd+=PASS;
  cmd+="\"";
  Serial.println(cmd);
  esp8266.println(cmd);
  delay(5000);
  if(esp8266.find("OK"))
  {
    return true;
  }
  else
  {
    return false;
  }
}

void getReadings(){
  raw_myTemp = analogRead(A1);
  Voltage = (raw_myTemp / 1023.0) * 5000; // 5000 to get millivots.
  tempC = Voltage * 0.1;
  myTemp = (tempC * 1.8) + 32; // conver to F
  Serial.println(myTemp);
  int myBPM = pulseSensor.getBeatsPerMinute();  // Calls function on our pulseSensor object
that returns BPM as an "int".
                                               // "myBPM" hold this BPM value now.
if (pulseSensor.sawStartOfBeat()) {            // Constantly test to see if "a beat happened".
Serial.println(myBPM);                        // Print the value inside of myBPM.
}

  delay(20);
    char buffer1[10];
     char buffer2[10];
    BPM = dtostrf(myBPM, 4, 1, buffer1);
    temp = dtostrf(myTemp, 4, 1, buffer2);
  }

void panic_button(){
    panic = digitalRead(8);
    if(panic == HIGH){
    Serial.println(panic);
      String cmd = "AT+CIPSTART=\"TCP\",\"";
  cmd += IP;
  cmd += "\",80";
  Serial.println(cmd);
  esp8266.println(cmd);
  delay(2000);
  if(esp8266.find("Error"))
  {
    return;
  }
  cmd = msg ;
  cmd += "&field3=";
  cmd += panic;
  cmd += "\r\n";
  Serial.print("AT+CIPSEND=");
  esp8266.print("AT+CIPSEND=");
  Serial.println(cmd.length());
  esp8266.println(cmd.length());
  if(esp8266.find(">"))
  {
    Serial.print(cmd);
    esp8266.print(cmd);
  }
  else
  {
    Serial.println("AT+CIPCLOSE");
    esp8266.println("AT+CIPCLOSE");
    //Resend...
    error=1;
  }
}
}
```