const char\* ssid = "SUDH2003";

const char\* password = "30020190";

const char\* apssid = "SUDH2003";

const char\* appassword = "30020190";

#include <WiFi.h>

#include <WiFiClientSecure.h>

#include "esp\_camera.h"

#include "soc/soc.h"

#include "soc/rtc\_cntl\_reg.h"

String Feedback="";

String Command="",cmd="",P1="",P2="",P3="",P4="",P5="",P6="",P7="",P8="",P9="";

byte ReceiveState=0,cmdState=1,strState=1,questionstate=0,equalstate=0,semicolonstate=0;

#define PWDN\_GPIO\_NUM 32

#define RESET\_GPIO\_NUM -1

#define XCLK\_GPIO\_NUM 0

#define SIOD\_GPIO\_NUM 26

#define SIOC\_GPIO\_NUM 27

#define Y9\_GPIO\_NUM 35

#define Y8\_GPIO\_NUM 34

#define Y7\_GPIO\_NUM 39

#define Y6\_GPIO\_NUM 36

#define Y5\_GPIO\_NUM 21

#define Y4\_GPIO\_NUM 19

#define Y3\_GPIO\_NUM 18

#define Y2\_GPIO\_NUM 5

#define VSYNC\_GPIO\_NUM 25

#define HREF\_GPIO\_NUM 23

#define PCLK\_GPIO\_NUM 22

WiFiServer server(80);

void ExecuteCommand()

{

//Serial.println("");

//Serial.println("Command: "+Command);

if (cmd!="getstill") {

Serial.println("cmd= "+cmd+" ,P1= "+P1+" ,P2= "+P2+" ,P3= "+P3+" ,P4= "+P4+" ,P5= "+P5+" ,P6= "+P6+" ,P7= "+P7+" ,P8= "+P8+" ,P9= "+P9);

Serial.println("");

}

if (cmd=="your cmd") {

// You can do anything.

// Feedback="<font color=\"red\">Hello World</font>";

}

else if (cmd=="ip") {

Feedback="AP IP: "+WiFi.softAPIP().toString();

Feedback+=", ";

Feedback+="STA IP: "+WiFi.localIP().toString();

}

else if (cmd=="mac") {

Feedback="STA MAC: "+WiFi.macAddress();

}

else if (cmd=="resetwifi") {

WiFi.begin(P1.c\_str(), P2.c\_str());

Serial.print("Connecting to ");

Serial.println(P1);

long int StartTime=millis();

while (WiFi.status() != WL\_CONNECTED)

{

delay(500);

if ((StartTime+5000) < millis()) break;

}

Serial.println("");

Serial.println("STAIP: "+WiFi.localIP().toString());

Feedback="STAIP: "+WiFi.localIP().toString();

}

else if (cmd=="restart") {

ESP.restart();

}

else if (cmd=="digitalwrite") {

ledcDetachPin(P1.toInt());

pinMode(P1.toInt(), OUTPUT);

digitalWrite(P1.toInt(), P2.toInt());

}

else if (cmd=="analogwrite") {

if (P1="4") {

ledcAttachPin(4, 4);

ledcSetup(4, 5000, 8);

ledcWrite(4,P2.toInt());

}

else {

ledcAttachPin(P1.toInt(), 5);

ledcSetup(5, 5000, 8);

ledcWrite(5,P2.toInt());

}

}

else if (cmd=="flash") {

ledcAttachPin(4, 4);

ledcSetup(4, 5000, 8);

int val = P1.toInt();

ledcWrite(4,val);

}

else if (cmd=="framesize") {

sensor\_t \* s = esp\_camera\_sensor\_get();

if (P1=="QQVGA")

s->set\_framesize(s, FRAMESIZE\_QQVGA);

else if (P1=="HQVGA")

s->set\_framesize(s, FRAMESIZE\_HQVGA);

else if (P1=="QVGA")

s->set\_framesize(s, FRAMESIZE\_QVGA);

else if (P1=="CIF")

s->set\_framesize(s, FRAMESIZE\_CIF);

else if (P1=="VGA")

s->set\_framesize(s, FRAMESIZE\_VGA);

else if (P1=="SVGA")

s->set\_framesize(s, FRAMESIZE\_SVGA);

else if (P1=="XGA")

s->set\_framesize(s, FRAMESIZE\_XGA);

else if (P1=="SXGA")

s->set\_framesize(s, FRAMESIZE\_SXGA);

else if (P1=="UXGA")

s->set\_framesize(s, FRAMESIZE\_UXGA);

else

s->set\_framesize(s, FRAMESIZE\_QVGA);

}

else if (cmd=="quality") {

sensor\_t \* s = esp\_camera\_sensor\_get();

int val = P1.toInt();

s->set\_quality(s, val);

}

else if (cmd=="contrast") {

sensor\_t \* s = esp\_camera\_sensor\_get();

int val = P1.toInt();

s->set\_contrast(s, val);

}

else if (cmd=="brightness") {

sensor\_t \* s = esp\_camera\_sensor\_get();

int val = P1.toInt();

s->set\_brightness(s, val);

}

else if (cmd=="serial") {

Serial.println(P1);

}

else if (cmd=="detectCount") {

Serial.println(P1+" = "+P2);

}

else if (cmd=="tcp") {

String domain=P1;

int port=P2.toInt();

String request=P3;

int wait=P4.toInt(); // wait = 0 or 1

if ((port==443)||(domain.indexOf("https")==0)||(domain.indexOf("HTTPS")==0))

Feedback=tcp\_https(domain,request,port,wait);

else

Feedback=tcp\_http(domain,request,port,wait);

}

else if (cmd=="linenotify") { //message=xxx&stickerPackageId=xxx&stickerId=xxx

String token = P1;

String request = P2;

Feedback=LineNotify(token,request,1);

if (Feedback.indexOf("status")!=-1) {

int s=Feedback.indexOf("{");

Feedback=Feedback.substring(s);

int e=Feedback.indexOf("}");

Feedback=Feedback.substring(0,e);

Feedback.replace("\"","");

Feedback.replace("{","");

Feedback.replace("}","");

}

}

else if (cmd=="sendCapturedImageToLineNotify") {

Feedback=sendCapturedImageToLineNotify(P1);

if (Feedback=="") Feedback="The image failed to send. <br>The framesize may be too large.";

}

else {

Feedback="Command is not defined.";

}

if (Feedback=="") Feedback=Command;

}

void setup() {

WRITE\_PERI\_REG(RTC\_CNTL\_BROWN\_OUT\_REG, 0);

Serial.begin(115200);

Serial.setDebugOutput(true);

Serial.println();

camera\_config\_t config;

config.ledc\_channel = LEDC\_CHANNEL\_0;

config.ledc\_timer = LEDC\_TIMER\_0;

config.pin\_d0 = Y2\_GPIO\_NUM;

config.pin\_d1 = Y3\_GPIO\_NUM;

config.pin\_d2 = Y4\_GPIO\_NUM;

config.pin\_d3 = Y5\_GPIO\_NUM;

config.pin\_d4 = Y6\_GPIO\_NUM;

config.pin\_d5 = Y7\_GPIO\_NUM;

config.pin\_d6 = Y8\_GPIO\_NUM;

config.pin\_d7 = Y9\_GPIO\_NUM;

config.pin\_xclk = XCLK\_GPIO\_NUM;

config.pin\_pclk = PCLK\_GPIO\_NUM;

config.pin\_vsync = VSYNC\_GPIO\_NUM;

config.pin\_href = HREF\_GPIO\_NUM;

config.pin\_sscb\_sda = SIOD\_GPIO\_NUM;

config.pin\_sscb\_scl = SIOC\_GPIO\_NUM;

config.pin\_pwdn = PWDN\_GPIO\_NUM;

config.pin\_reset = RESET\_GPIO\_NUM;

config.xclk\_freq\_hz = 20000000;

config.pixel\_format = PIXFORMAT\_JPEG;

//init with high specs to pre-allocate larger buffers

if(psramFound()){

config.frame\_size = FRAMESIZE\_UXGA;

config.jpeg\_quality = 10; //0-63 lower number means higher quality

config.fb\_count = 2;

} else {

config.frame\_size = FRAMESIZE\_SVGA;

config.jpeg\_quality = 12; //0-63 lower number means higher quality

config.fb\_count = 1;

}

// camera init

esp\_err\_t err = esp\_camera\_init(&config);

if (err != ESP\_OK) {

Serial.printf("Camera init failed with error 0x%x", err);

delay(1000);

ESP.restart();

}

//drop down frame size for higher initial frame rate

sensor\_t \* s = esp\_camera\_sensor\_get();

s->set\_framesize(s, FRAMESIZE\_QVGA); //UXGA|SXGA|XGA|SVGA|VGA|CIF|QVGA|HQVGA|QQVGA

ledcAttachPin(4, 4);

ledcSetup(4, 5000, 8);

WiFi.mode(WIFI\_AP\_STA);

//WiFi.config(IPAddress(192, 168, 201, 100), IPAddress(192, 168, 201, 2), IPAddress(255, 255, 255, 0));

WiFi.begin(ssid, password);

delay(1000);

Serial.println("");

Serial.print("Connecting to ");

Serial.println(ssid);

long int StartTime=millis();

while (WiFi.status() != WL\_CONNECTED)

{

delay(500);

if ((StartTime+10000) < millis()) break;

}

if (WiFi.status() == WL\_CONNECTED) {

WiFi.softAP((WiFi.localIP().toString()+"\_"+(String)apssid).c\_str(), appassword);

Serial.println("");

Serial.println("STAIP address: ");

Serial.println(WiFi.localIP());

for (int i=0;i<5;i++) {

ledcWrite(4,10);

delay(200);

ledcWrite(4,0);

delay(200);

}

}

else {

WiFi.softAP((WiFi.softAPIP().toString()+"\_"+(String)apssid).c\_str(), appassword);

for (int i=0;i<2;i++) {

ledcWrite(4,10);

delay(1000);

ledcWrite(4,0);

delay(1000);

}

}

//WiFi.softAPConfig(IPAddress(192, 168, 4, 1), IPAddress(192, 168, 4, 1), IPAddress(255, 255, 255, 0));

Serial.println("");

Serial.println("APIP address: ");

Serial.println(WiFi.softAPIP());

pinMode(4, OUTPUT);

digitalWrite(4, LOW);

server.begin();

}

static const char PROGMEM INDEX\_HTML[] = R"rawliteral(

<!DOCTYPE html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width,initial-scale=1">

<script src="https:\/\/ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js"></script>

<script src="https:\/\/cdn.jsdelivr.net/npm/@tensorflow/tfjs@1.3.1/dist/tf.min.js"> </script>

<script src="https:\/\/cdn.jsdelivr.net/npm/@tensorflow-models/coco-ssd@2.1.0"> </script>

</head><body>

<img id="ShowImage" src="" style="display:none">

<canvas id="canvas" width="0" height="0"></canvas>

<table>

<tr>

<td><input type="button" id="restart" value="Restart"></td>

<td colspan="2"><input type="button" id="getStill" value="Start Detect" style="display:none"></td>

</tr>

<tr>

<td>Object</td>

<td colspan="2">

<select id="object" onchange="count.innerHTML='';">

<option value="person">person</option>

<option value="bicycle">bicycle</option>

<option value="car">car</option>

<option value="motorcycle">motorcycle</option>

<option value="airplane">airplane</option>

<option value="bus">bus</option>

<option value="train">train</option>

<option value="truck">truck</option>

<option value="boat">boat</option>

<option value="traffic light">traffic light</option>

<option value="fire hydrant">fire hydrant</option>

<option value="stop sign">stop sign</option>

<option value="parking meter">parking meter</option>

<option value="bench">bench</option>

<option value="bird">bird</option>

<option value="cat">cat</option>

<option value="dog">dog</option>

<option value="horse">horse</option>

<option value="sheep">sheep</option>

<option value="cow">cow</option>

<option value="elephant">elephant</option>

<option value="bear">bear</option>

<option value="zebra">zebra</option>

<option value="giraffe">giraffe</option>

<option value="backpack">backpack</option>

<option value="umbrella">umbrella</option>

<option value="handbag">handbag</option>

<option value="tie">tie</option>

<option value="suitcase">suitcase</option>

<option value="frisbee">frisbee</option>

<option value="skis">skis</option>

<option value="snowboard">snowboard</option>

<option value="sports ball">sports ball</option>

<option value="kite">kite</option>

<option value="baseball bat">baseball bat</option>

<option value="baseball glove">baseball glove</option>

<option value="skateboard">skateboard</option>

<option value="surfboard">surfboard</option>

<option value="tennis racket">tennis racket</option>

<option value="bottle">bottle</option>

<option value="wine glass">wine glass</option>

<option value="cup">cup</option>

<option value="fork">fork</option>

<option value="knife">knife</option>

<option value="spoon">spoon</option>

<option value="bowl">bowl</option>

<option value="banana">banana</option>

<option value="apple">apple</option>

<option value="sandwich">sandwich</option>

<option value="orange">orange</option>

<option value="broccoli">broccoli</option>

<option value="carrot">carrot</option>

<option value="hot dog">hot dog</option>

<option value="pizza">pizza</option>

<option value="donut">donut</option>

<option value="cake">cake</option>

<option value="chair">chair</option>

<option value="couch">couch</option>

<option value="potted plant">potted plant</option>

<option value="bed">bed</option>

<option value="dining table">dining table</option>

<option value="toilet">toilet</option>

<option value="tv">tv</option>

<option value="laptop">laptop</option>

<option value="mouse">mouse</option>

<option value="remote">remote</option>

<option value="keyboard">keyboard</option>

<option value="cell phone">cell phone</option>

<option value="microwave">microwave</option>

<option value="oven">oven</option>

<option value="toaster">toaster</option>

<option value="sink">sink</option>

<option value="refrigerator">refrigerator</option>

<option value="book">book</option>

<option value="clock">clock</option>

<option value="vase">vase</option>

<option value="scissors">scissors</option>

<option value="teddy bear">teddy bear</option>

<option value="hair drier">hair drier</option>

<option value="toothbrush">toothbrush</option>

</select>

</td>

<td><span id="count" style="color:red"><span></td>

</tr>

<tr>

<td>ScoreLimit</td>

<td colspan="2">

<select id="score">

<option value="1.0">1</option>

<option value="0.9">0.9</option>

<option value="0.8">0.8</option>

<option value="0.7">0.7</option>

<option value="0.6">0.6</option>

<option value="0.5">0.5</option>

<option value="0.4">0.4</option>

<option value="0.3">0.3</option>

<option value="0.2">0.2</option>

<option value="0.1">0.1</option>

<option value="0" selected="selected">0</option>

</select>

</td>

</tr>

<tr>

<td>MirrorImage</td>

<td colspan="2">

<select id="mirrorimage">

<option value="1">yes</option>

<option value="0">no</option>

</select>

</td>

</tr>

<tr>

<td>Resolution</td>

<td colspan="2">

<select id="framesize">

<option value="UXGA">UXGA(1600x1200)</option>

<option value="SXGA">SXGA(1280x1024)</option>

<option value="XGA">XGA(1024x768)</option>

<option value="SVGA">SVGA(800x600)</option>

<option value="VGA">VGA(640x480)</option>

<option value="CIF">CIF(400x296)</option>

<option value="QVGA" selected="selected">QVGA(320x240)</option>

<option value="HQVGA">HQVGA(240x176)</option>

<option value="QQVGA">QQVGA(160x120)</option>

</select>

</td>

</tr>

<tr>

<td>Flash</td>

<td colspan="2"><input type="range" id="flash" min="0" max="255" value="0"></td>

</tr>

<tr>

<td>Quality</td>

<td colspan="2"><input type="range" id="quality" min="10" max="63" value="10"></td>

</tr>

<tr>

<td>Brightness</td>

<td colspan="2"><input type="range" id="brightness" min="-2" max="2" value="0"></td>

</tr>

<tr>

<td>Contrast</td>

<td colspan="2"><input type="range" id="contrast" min="-2" max="2" value="0"></td>

</tr>

</table>

<div id="result" style="color:red"><div>

</body>

</html>

<script>

var getStill = document.getElementById('getStill');

var ShowImage = document.getElementById('ShowImage');

var canvas = document.getElementById("canvas");

var context = canvas.getContext("2d");

var object = document.getElementById('object');

var score = document.getElementById("score");

var mirrorimage = document.getElementById("mirrorimage");

var count = document.getElementById('count');

var result = document.getElementById('result');

var flash = document.getElementById('flash');

var lastValue="";

var myTimer;

var restartCount=0;

var Model;

getStill.onclick = function (event) {

clearInterval(myTimer);

myTimer = setInterval(function(){error\_handle();},5000);

ShowImage.src=location.origin+'/?getstill='+Math.random();

}

function error\_handle() {

restartCount++;

clearInterval(myTimer);

if (restartCount<=2) {

result.innerHTML = "Get still error. <br>Restart ESP32-CAM "+restartCount+" times.";

myTimer = setInterval(function(){getStill.click();},10000);

}

else

result.innerHTML = "Get still error. <br>Please close the page and check ESP32-CAM.";

}

ShowImage.onload = function (event) {

clearInterval(myTimer);

restartCount=0;

canvas.setAttribute("width", ShowImage.width);

canvas.setAttribute("height", ShowImage.height);

if (mirrorimage.value==1) {

context.translate((canvas.width + ShowImage.width) / 2, 0);

context.scale(-1, 1);

context.drawImage(ShowImage, 0, 0, ShowImage.width, ShowImage.height);

context.setTransform(1, 0, 0, 1, 0, 0);

}

else

context.drawImage(ShowImage,0,0,ShowImage.width,ShowImage.height);

if (Model) {

DetectImage();

}

}

restart.onclick = function (event) {

fetch(location.origin+'/?restart=stop');

}

framesize.onclick = function (event) {

fetch(document.location.origin+'/?framesize='+this.value+';stop');

}

flash.onchange = function (event) {

fetch(location.origin+'/?flash='+this.value+';stop');

}

quality.onclick = function (event) {

fetch(document.location.origin+'/?quality='+this.value+';stop');

}

brightness.onclick = function (event) {

fetch(document.location.origin+'/?brightness='+this.value+';stop');

}

contrast.onclick = function (event) {

fetch(document.location.origin+'/?contrast='+this.value+';stop');

}

function ObjectDetect() {

result.innerHTML = "Please wait for loading model.";

cocoSsd.load().then(cocoSsd\_Model => {

Model = cocoSsd\_Model;

result.innerHTML = "";

getStill.style.display = "block";

});

}

function DetectImage() {

Model.detect(canvas).then(Predictions => {

var s = (canvas.width>canvas.height)?canvas.width:canvas.height;

var objectCount=0;

//console.log('Predictions: ', Predictions);

if (Predictions.length>0) {

result.innerHTML = "";

for (var i=0;i<Predictions.length;i++) {

const x = Predictions[i].bbox[0];

const y = Predictions[i].bbox[1];

const width = Predictions[i].bbox[2];

const height = Predictions[i].bbox[3];

context.lineWidth = Math.round(s/200);

context.strokeStyle = "#00FFFF";

context.beginPath();

context.rect(x, y, width, height);

context.stroke();

context.lineWidth = "2";

context.fillStyle = "red";

context.font = Math.round(s/30) + "px Arial";

context.fillText(Predictions[i].class, x, y);

//context.fillText(i, x, y);

result.innerHTML+= "[ "+i+" ] "+Predictions[i].class+", "+Math.round(Predictions[i].score\*100)+"%, "+Math.round(x)+", "+Math.round(y)+", "+Math.round(width)+", "+Math.round(height)+"<br>";

if (Predictions[i].class==object.value&&Predictions[i].score>=score.value) {

objectCount++;

}

}

count.innerHTML = objectCount;

}

else {

result.innerHTML = "Unrecognizable";

count.innerHTML = "0";

}

//if (count.innerHTML != lastValue) {

lastValue = count.innerHTML;

if (objectCount>0) {

//$.ajax({url: document.location.origin+'/?serial='+object.value+';stop', async: false});

$.ajax({url: document.location.origin+'/?detectCount='+object.value+';'+String(objectCount)+';stop', async: false});

}

//}

try {

document.createEvent("TouchEvent");

setTimeout(function(){getStill.click();},250);

}

catch(e) {

setTimeout(function(){getStill.click();},150);

}

});

}

function getFeedback(target) {

var data = $.ajax({

type: "get",

dataType: "text",

url: target,

success: function(response)

{

result.innerHTML = response;

},

error: function(exception)

{

result.innerHTML = 'fail';

}

});

}

window.onload = function () { ObjectDetect(); }

</script>

)rawliteral";

void loop() {

Feedback="";Command="";cmd="";P1="";P2="";P3="";P4="";P5="";P6="";P7="";P8="";P9="";

ReceiveState=0,cmdState=1,strState=1,questionstate=0,equalstate=0,semicolonstate=0;

WiFiClient client = server.available();

if (client) {

String currentLine = "";

while (client.connected()) {

if (client.available()) {

char c = client.read();

getCommand(c);

if (c == '\n') {

if (currentLine.length() == 0) {

if (cmd=="getstill") {

camera\_fb\_t \* fb = NULL;

fb = esp\_camera\_fb\_get();

if(!fb) {

Serial.println("Camera capture failed");

delay(1000);

ESP.restart();

}

client.println("HTTP/1.1 200 OK");

client.println("Access-Control-Allow-Origin: \*");

client.println("Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept");

client.println("Access-Control-Allow-Methods: GET,POST,PUT,DELETE,OPTIONS");

client.println("Content-Type: image/jpeg");

client.println("Content-Disposition: form-data; name=\"imageFile\"; filename=\"picture.jpg\"");

client.println("Content-Length: " + String(fb->len));

client.println("Connection: close");

client.println();

uint8\_t \*fbBuf = fb->buf;

size\_t fbLen = fb->len;

for (size\_t n=0;n<fbLen;n=n+1024) {

if (n+1024<fbLen) {

client.write(fbBuf, 1024);

fbBuf += 1024;

}

else if (fbLen%1024>0) {

size\_t remainder = fbLen%1024;

client.write(fbBuf, remainder);

}

}

esp\_camera\_fb\_return(fb);

pinMode(4, OUTPUT);

digitalWrite(4, LOW);

}

else {

client.println("HTTP/1.1 200 OK");

client.println("Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept");

client.println("Access-Control-Allow-Methods: GET,POST,PUT,DELETE,OPTIONS");

client.println("Content-Type: text/html; charset=utf-8");

client.println("Access-Control-Allow-Origin: \*");

client.println("Connection: close");

client.println();

String Data="";

if (cmd!="")

Data = Feedback;

else {

Data = String((const char \*)INDEX\_HTML);

}

int Index;

for (Index = 0; Index < Data.length(); Index = Index+1000) {

client.print(Data.substring(Index, Index+1000));

}

client.println();

}

Feedback="";

break;

} else {

currentLine = "";

}

}

else if (c != '\r') {

currentLine += c;

}

if ((currentLine.indexOf("/?")!=-1)&&(currentLine.indexOf(" HTTP")!=-1)) {

if (Command.indexOf("stop")!=-1) { http://192.168.xxx.xxx/?cmd=aaa;bbb;ccc;stop

client.println();

client.println();

client.stop();

}

currentLine="";

Feedback="";

ExecuteCommand();

}

}

}

delay(1);

client.stop();

}

}

void getCommand(char c)

{

if (c=='?') ReceiveState=1;

if ((c==' ')||(c=='\r')||(c=='\n')) ReceiveState=0;

if (ReceiveState==1)

{

Command=Command+String(c);

if (c=='=') cmdState=0;

if (c==';') strState++;

if ((cmdState==1)&&((c!='?')||(questionstate==1))) cmd=cmd+String(c);

if ((cmdState==0)&&(strState==1)&&((c!='=')||(equalstate==1))) P1=P1+String(c);

if ((cmdState==0)&&(strState==2)&&(c!=';')) P2=P2+String(c);

if ((cmdState==0)&&(strState==3)&&(c!=';')) P3=P3+String(c);

if ((cmdState==0)&&(strState==4)&&(c!=';')) P4=P4+String(c);

if ((cmdState==0)&&(strState==5)&&(c!=';')) P5=P5+String(c);

if ((cmdState==0)&&(strState==6)&&(c!=';')) P6=P6+String(c);

if ((cmdState==0)&&(strState==7)&&(c!=';')) P7=P7+String(c);

if ((cmdState==0)&&(strState==8)&&(c!=';')) P8=P8+String(c);

if ((cmdState==0)&&(strState>=9)&&((c!=';')||(semicolonstate==1))) P9=P9+String(c);

if (c=='?') questionstate=1;

if (c=='=') equalstate=1;

if ((strState>=9)&&(c==';')) semicolonstate=1;

}

}

String tcp\_http(String domain,String request,int port,byte wait)

{

WiFiClient client\_tcp;

if (client\_tcp.connect(domain.c\_str(), port))

{

Serial.println("GET " + request);

client\_tcp.println("GET " + request + " HTTP/1.1");

client\_tcp.println("Host: " + domain);

client\_tcp.println("Connection: close");

client\_tcp.println();

String getResponse="",Feedback="";

boolean state = false;

int waitTime = 3000; // timeout 3 seconds

long startTime = millis();

while ((startTime + waitTime) > millis())

{

while (client\_tcp.available())

{

char c = client\_tcp.read();

if (state==true) Feedback += String(c);

if (c == '\n')

{

if (getResponse.length()==0) state=true;

getResponse = "";

}

else if (c != '\r')

getResponse += String(c);

if (wait==1)

startTime = millis();

}

if (wait==0)

if ((state==true)&&(Feedback.length()!= 0)) break;

}

client\_tcp.stop();

return Feedback;

}

else

return "Connection failed";

}

String tcp\_https(String domain,String request,int port,byte wait)

{

WiFiClientSecure client\_tcp;

client\_tcp.setInsecure(); //run version 1.0.5 or above

if (client\_tcp.connect(domain.c\_str(), port))

{

Serial.println("GET " + request);

client\_tcp.println("GET " + request + " HTTP/1.1");

client\_tcp.println("Host: " + domain);

client\_tcp.println("Connection: close");

client\_tcp.println();

String getResponse="",Feedback="";

boolean state = false;

int waitTime = 3000; // timeout 3 seconds

long startTime = millis();

while ((startTime + waitTime) > millis())

{

while (client\_tcp.available())

{

char c = client\_tcp.read();

if (state==true) Feedback += String(c);

if (c == '\n')

{

if (getResponse.length()==0) state=true;

getResponse = "";

}

else if (c != '\r')

getResponse += String(c);

if (wait==1)

startTime = millis();

}

if (wait==0)

if ((state==true)&&(Feedback.length()!= 0)) break;

}

client\_tcp.stop();

return Feedback;

}

else

return "Connection failed";

}

String LineNotify(String token, String request, byte wait)

{

request.replace("%","%25");

request.replace(" ","%20");

request.replace("&","%20");

request.replace("#","%20");

//request.replace("\'","%27");

request.replace("\"","%22");

request.replace("\n","%0D%0A");

request.replace("%3Cbr%3E","%0D%0A");

request.replace("%3Cbr/%3E","%0D%0A");

request.replace("%3Cbr%20/%3E","%0D%0A");

request.replace("%3CBR%3E","%0D%0A");

request.replace("%3CBR/%3E","%0D%0A");

request.replace("%3CBR%20/%3E","%0D%0A");

request.replace("%20stickerPackageId","&stickerPackageId");

request.replace("%20stickerId","&stickerId");

WiFiClientSecure client\_tcp;

client\_tcp.setInsecure(); //run version 1.0.5 or above

if (client\_tcp.connect("notify-api.line.me", 443))

{

client\_tcp.println("POST /api/notify HTTP/1.1");

client\_tcp.println("Connection: close");

client\_tcp.println("Host: notify-api.line.me");

client\_tcp.println("User-Agent: ESP8266/1.0");

client\_tcp.println("Authorization: Bearer " + token);

client\_tcp.println("Content-Type: application/x-www-form-urlencoded");

client\_tcp.println("Content-Length: " + String(request.length()));

client\_tcp.println();

client\_tcp.println(request);

client\_tcp.println();

String getResponse="",Feedback="";

boolean state = false;

int waitTime = 3000; // timeout 3 seconds

long startTime = millis();

while ((startTime + waitTime) > millis())

{

while (client\_tcp.available())

{

char c = client\_tcp.read();

if (state==true) Feedback += String(c);

if (c == '\n')

{

if (getResponse.length()==0) state=true;

getResponse = "";

}

else if (c != '\r')

getResponse += String(c);

if (wait==1)

startTime = millis();

}

if (wait==0)

if ((state==true)&&(Feedback.length()!= 0)) break;

}

client\_tcp.stop();

return Feedback;

}

else

return "Connection failed";

}

String sendCapturedImageToLineNotify(String token)

{

String getAll="", getBody = "";

camera\_fb\_t \* fb = NULL;

fb = esp\_camera\_fb\_get();

if(!fb) {

Serial.println("Camera capture failed");

delay(1000);

ESP.restart();

return "";

}

WiFiClientSecure client\_tcp;

client\_tcp.setInsecure(); //run version 1.0.5 or above

Serial.println("Connect to notify-api.line.me");

if (client\_tcp.connect("notify-api.line.me", 443)) {

Serial.println("Connection successful");

String message = "Welcome to Taiwan.";

String head = "--Taiwan\r\nContent-Disposition: form-data; name=\"message\"; \r\n\r\n" + message + "\r\n--Taiwan\r\nContent-Disposition: form-data; name=\"imageFile\"; filename=\"esp32-cam.jpg\"\r\nContent-Type: image/jpeg\r\n\r\n";

String tail = "\r\n--Taiwan--\r\n";

uint16\_t imageLen = fb->len;

uint16\_t extraLen = head.length() + tail.length();

uint16\_t totalLen = imageLen + extraLen;

client\_tcp.println("POST /api/notify HTTP/1.1");

client\_tcp.println("Connection: close");

client\_tcp.println("Host: notify-api.line.me");

client\_tcp.println("Authorization: Bearer " + token);

client\_tcp.println("Content-Length: " + String(totalLen));

client\_tcp.println("Content-Type: multipart/form-data; boundary=Taiwan");

client\_tcp.println();

client\_tcp.print(head);

uint8\_t \*fbBuf = fb->buf;

size\_t fbLen = fb->len;

for (size\_t n=0;n<fbLen;n=n+1024) {

if (n+1024<fbLen) {

client\_tcp.write(fbBuf, 1024);

fbBuf += 1024;

}

else if (fbLen%1024>0) {

size\_t remainder = fbLen%1024;

client\_tcp.write(fbBuf, remainder);

}

}

client\_tcp.print(tail);

esp\_camera\_fb\_return(fb);

int waitTime = 10000; // timeout 10 seconds

long startTime = millis();

boolean state = false;

while ((startTime + waitTime) > millis())

{

Serial.print(".");

delay(100);

while (client\_tcp.available())

{

char c = client\_tcp.read();

if (state==true) getBody += String(c);

if (c == '\n')

{

if (getAll.length()==0) state=true;

getAll = "";

}

else if (c != '\r')

getAll += String(c);

startTime = millis();

}

if (getBody.length()>0) break;

}

client\_tcp.stop();

//Serial.println(getAll);

Serial.println(getBody);

}

else {

getAll="Connected to notify-api.line.me failed.";

getBody="Connected to notify-api.line.me failed.";

Serial.println("Connected to notify-api.line.me failed.");

}

//return getAll;

return getBody;

}