```
#include <ArduinoWebsockets.h>
#include "esp_http_server.h"
#include "esp_timer.h"
#include "esp_camera.h"
#include "camera_index.h"
#include "Arduino.h"
#include "fd_forward.h"
#include "fr_forward.h"
#include "fr_flash.h"

const char* ssid = "Fernando";
const char* password = "12345678";

#define ENROLL_CONFIRM_TIMES 5
#define FACE_ID_SAVE_NUMBER 7

// Select camera model
//#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"

using namespace websockets;
WebsocketsServer socket_server;

camera_fb_t * fb = NULL;

long current_millis;
long last_detected_millis = 0;
short i = 0;

#define accesorios 12
#define arranque 13
#define pulsador 15
#define led 2
#define flash 4
```

```
#define bomba 14

unsigned long ignition_start_millis = 0;
long interval = 60000;
bool face_recognised = false;

void app_facenet_main();
void app_httpserver_init();

typedef struct{
  uint8_t *image;
  box_array_t *net_boxes;
  dl_matrix3d_t *face_id;
} http_img_process_result;


static inline mtmn_config_t app_mtmn_config(){
  mtmn_config_t mtmn_config = {0};
  mtmn_config.type = FAST;
  mtmn_config.min_face = 80;
  mtmn_config.pyramid = 0.707;
  mtmn_config.pyramid_times = 4;
  mtmn_config.p_threshold.score = 0.6;
  mtmn_config.p_threshold.nms = 0.7;
  mtmn_config.p_threshold.candidate_number = 20;
  mtmn_config.r_threshold.score = 0.7;
  mtmn_config.r_threshold.nms = 0.7;
  mtmn_config.r_threshold.candidate_number = 10;
  mtmn_config.o_threshold.score = 0.7;
  mtmn_config.o_threshold.nms = 0.7;
  mtmn_config.o_threshold.candidate_number = 1;
  return mtmn_config;
}
mtmn_config_t mtmn_config = app_mtmn_config();

face_id_name_list st_face_list;
static dl_matrix3du_t *aligned_face = NULL;

httpd_handle_t camera_httpd = NULL;
```

```
typedef enum{
  START_STREAM,
  START_DETECT,
  SHOW_FACES,
  START_RECOGNITION,
  START_ENROLL,
  ENROLL_COMPLETE,
  DELETE_ALL,
} en_fsm_state;
en_fsm_state g_state;

typedef struct{
  char enroll_name[ENROLL_NAME_LEN];
} httpd_resp_value;

httpd_resp_value st_name;

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  pinMode(accesorios, INPUT);
  pinMode(arranque, OUTPUT);
  pinMode(pulsador, INPUT);
  pinMode(led, OUTPUT);
  pinMode(flash, OUTPUT);
  pinMode(bomba, OUTPUT);

  digitalWrite(arranque, LOW);
  digitalWrite(led, LOW);
  digitalWrite(flash, LOW);
  digitalWrite(bomba, LOW);

  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;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

#if defined(CAMERA_MODEL_ESP_EYE)
  pinMode(13, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
#endif

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
```

```
    Serial.printf("Inicio de la camara fallo con el error
0x%x", err);
    return;
  }

  sensor_t * s = esp_camera_sensor_get();
  s->set_framesize(s, FRAMESIZE_QVGA);

#if defined(CAMERA_MODEL_M5STACK_WIDE)
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
#endif

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi conectado");

  app_httpserver_init();
  app_facenet_main();
  socket_server.listen(82);

  Serial.print("Camara Lista! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' para conectarse");
}

static esp_err_t index_handler(httpd_req_t *req) {
  httpd_resp_set_type(req, "text/html");
  httpd_resp_set_hdr(req, "Content-Encoding", "gzip");
  return httpd_resp_send(req, (const char
*)index_ov2640_html_gz, index_ov2640_html_gz_len);
}

httpd_uri_t index_uri = {
```

```
  .uri       = "/",
  .method    = HTTP_GET,
  .handler   = index_handler,
  .user_ctx  = NULL
};

void app_httpserver_init (){
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  if (httpd_start(&camera_httpd, &config) == ESP_OK)
    Serial.println("httpd_start");
  {
    httpd_register_uri_handler(camera_httpd, &index_uri);
  }
}

void app_facenet_main(){
  face_id_name_init(&st_face_list, FACE_ID_SAVE_NUMBER,
ENROLL_CONFIRM_TIMES);
  aligned_face = dl_matrix3du_alloc(1, FACE_WIDTH,
FACE_HEIGHT, 3);
  read_face_id_from_flash_with_name(&st_face_list);
}

static inline int do_enrollment(face_id_name_list
*face_list, dl_matrix3d_t *new_id){
  ESP_LOGD(TAG, "START ENROLLING");
  int left_sample_face =
enroll_face_id_to_flash_with_name(face_list, new_id,
st_name.enroll_name);
  ESP_LOGD(TAG, "Face ID %s Enrollment: Sample %d",
           st_name.enroll_name,
           ENROLL_CONFIRM_TIMES - left_sample_face);
  return left_sample_face;
}

static esp_err_t send_face_list(WebsocketsClient
&client){
```

```
  client.send("Eliminando rosotros"); // tell browser to
delete all faces
  face_id_node *head = st_face_list.head;
  char add_face[64];
  for (int i = 0; i < st_face_list.count; i++){ // loop
current faces
    sprintf(add_face, "listface:%s", head->id_name);
    client.send(add_face); //send face to browser
    head = head->next;
  }
}

static esp_err_t delete_all_faces(WebsocketsClient
&client){
  delete_face_all_in_flash_with_name(&st_face_list);
  client.send("Eliminando rostros");
}

void handle_message(WebsocketsClient &client,
WebsocketsMessage msg){
  if (msg.data() == "stream") {
    g_state = START_STREAM;
    client.send("Transmitiendo");
  }
  if (msg.data() == "detect") {
    g_state = START_DETECT;
    client.send("Detectando");
  }
  if (msg.data().substring(0, 8) == "capture:") {
    g_state = START_ENROLL;
    char person[FACE_ID_SAVE_NUMBER * ENROLL_NAME_LEN] =
{0,};
    msg.data().substring(8).toCharArray(person,
sizeof(person));
    memcpy(st_name.enroll_name, person, strlen(person) +
1);
    client.send("Capturando");
  }
```

```
  if (msg.data() == "recognise") {
    g_state = START_RECOGNITION;
    client.send("Reconociendo");
  }
  if (msg.data().substring(0, 7) == "remove:") {
    char person[ENROLL_NAME_LEN * FACE_ID_SAVE_NUMBER];
    msg.data().substring(7).toCharArray(person,
sizeof(person));
    delete_face_id_in_flash_with_name(&st_face_list,
person);
    send_face_list(client); // reset faces in the browser
  }
  if (msg.data() == "delete_all") {
    delete_all_faces(client);
  }
}

void ignition_start(WebsocketsClient &client) {
  if (digitalRead(accesorios) == HIGH) {
    digitalWrite(bomba, HIGH);
    digitalWrite(led,HIGH);
    digitalWrite(arranque,HIGH);
    Serial.println("Rostro Reconocido");
    client.send("Rostro Reconocido");
    ignition_start_millis = millis(); // time relay
closed and door opened
  }
}

void loop() {
  auto client = socket_server.accept();
  client.onMessage(handle_message);
  dl_matrix3du_t *image_matrix = dl_matrix3du_alloc(1,
320, 240, 3);
  http_img_process_result out_res = {0};
  out_res.image = image_matrix->item;

  send_face_list(client);
```

```
  client.send("Transmitiendo");

  if (digitalRead(pulsador) == LOW){
      delay(200);
      i = 1 - i;
  }
  if (i == 1) digitalWrite(flash,HIGH);
  else digitalWrite(flash,LOW);


  while (client.available()) {
    client.poll();

    if (millis() - interval > ignition_start_millis) { //
current time - face recognised time > 5 secs
      digitalWrite(bomba, LOW); //open relay
      digitalWrite(led, LOW); //open relay
      digitalWrite(arranque, LOW); //open relay
    }

    fb = esp_camera_fb_get();

    if (g_state == START_DETECT || g_state ==
START_ENROLL || g_state == START_RECOGNITION)
    {
      out_res.net_boxes = NULL;
      out_res.face_id = NULL;

      fmt2rgb888(fb->buf, fb->len, fb->format,
out_res.image);

      out_res.net_boxes = face_detect(image_matrix,
&mtmn_config);

      if (out_res.net_boxes){
        if (align_face(out_res.net_boxes, image_matrix,
aligned_face) == ESP_OK){
          out_res.face_id = get_face_id(aligned_face);
```

```
          last_detected_millis = millis();
          if (g_state == START_DETECT) {
            client.send("Rostro no detectado");
          }

          if (g_state == START_ENROLL){
            int left_sample_face =
do_enrollment(&st_face_list, out_res.face_id);
            char enrolling_message[64];
            sprintf(enrolling_message, "Numero de
muestras %d para %s", ENROLL_CONFIRM_TIMES -
left_sample_face, st_name.enroll_name);
            client.send(enrolling_message);
            if (left_sample_face == 0)
            {
              ESP_LOGI(TAG, "Enrolled Face ID: %s",
st_face_list.tail->id_name);
              g_state = START_STREAM;
              char captured_message[64];
              sprintf(captured_message, "Rostro capturado
para %s", st_face_list.tail->id_name);
              client.send(captured_message);
              send_face_list(client);

            }
          }

          if (g_state == START_RECOGNITION  &&
(st_face_list.count > 0))
          {
            face_id_node *f =
recognize_face_with_name(&st_face_list, out_res.face_id);
            if (f)
            {
              char recognised_message[64];
              sprintf(recognised_message, "Auto activado
para %s", f->id_name);
              ignition_start(client);
```

```
              client.send(recognised_message);
            }
            else
            {
              client.send("FACE NOT RECOGNISED");
            }
          }
          dl_matrix3d_free(out_res.face_id);
        }

      }
      else
      {
        if (g_state != START_DETECT) {
          client.send("NO FACE DETECTED");
        }
      }

      if (g_state == START_DETECT && millis() -
last_detected_millis > 500) { // Detecting but no face
detected
        client.send("DETECTING");
      }

    }

    client.sendBinary((const char *)fb->buf, fb->len);

    esp_camera_fb_return(fb);
    fb = NULL;
  }
}
```