```
// Copyright 2015-2016 Espressif Systems (Shanghai) PTE LTD
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
#include "esp_http_server.h"
#include "esp_timer.h"
#include "esp32-hal-ledc.h"
#include "sdkconfig.h"
#include "html.h"
#include "Arduino.h"
#include "weight_sensor.h"

#if defined(ARDUINO_ARCH_ESP32) && defined(CONFIG_ARDUHAL_ESP_LOG)
#include "esp32-hal-log.h"
#endif

httpd_handle_t camera_httpd = NULL;
extern WeightSensor LoadCell1;
extern WeightSensor LoadCell2;
extern WeightSensor LoadCell3;
extern WeightSensor LoadCell4;

static esp_err_t parse_get(httpd_req_t *req, char **obuf)
{
   char *buf = NULL;
   size_t buf_len = 0;

   buf_len = httpd_req_get_url_query_len(req) + 1;
   if (buf_len > 1) {
       buf = (char *)malloc(buf_len);
       if (!buf) {
           httpd_resp_send_500(req);
           return ESP_FAIL;
```

```
        }
        if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK) {
            *obuf = buf;
            return ESP_OK;
        }
        free(buf);
    }
    httpd_resp_send_404(req);
    return ESP_FAIL;
}

static esp_err_t parse_request_data(httpd_req_t *req)
{
 char *buf = NULL;
 static char date[100];
 static char height[20];
 static int height_int;
 static char age[20];
 static int age_int;
 static char gender[20];
 static char calibration[20];
 static float calibration_float;

 if (parse_get(req, &buf) != ESP_OK) {
     return ESP_FAIL;
 }

 if ( httpd_query_key_value(buf, "date", date, sizeof(date)) != ESP_OK ) {
     free(buf);
     //return ESP_FAIL;
 }
  if ( httpd_query_key_value(buf, "height", height, sizeof(date)) != ESP_OK ) {
     free(buf);
     //return ESP_FAIL;
 }

 Serial.print("height: ");
 Serial.println(height);
 height_int = atoi(height);
  if ( httpd_query_key_value(buf, "age", age, sizeof(date)) != ESP_OK ) {
     free(buf);
     //return ESP_FAIL;
```

```
 }

 Serial.print("age: ");
 Serial.println(age);
 age_int = atoi(age);

 if ( httpd_query_key_value(buf, "gender", gender, sizeof(date)) != ESP_OK ) {
     free(buf);
     //return ESP_FAIL;
 }

 Serial.print("gender: ");
 Serial.println(gender);

 if ( httpd_query_key_value(buf, "calibration", calibration, sizeof(date)) != ESP_OK )
{
     free(buf);
     //return ESP_FAIL;
 }
 Serial.print("calibration: ");
 Serial.println(calibration);

 calibration_float = atof(calibration);
 if ( calibration_float != 0 )
 {
   if ( LoadCell1.setNvsCalibrationVal(calibration_float) )
   {
     Serial.print("New calibration value is set for load cell1");
     Serial.println(calibration_float);
     LoadCell1.setCalibrationFactor(calibration_float);
   }
   if ( LoadCell2.setNvsCalibrationVal(calibration_float) )
   {
     Serial.print("New calibration value is set for load cell1");
     Serial.println(calibration_float);
     LoadCell2.setCalibrationFactor(calibration_float);
   }
   if ( LoadCell3.setNvsCalibrationVal(calibration_float) )
   {
     Serial.print("New calibration value is set for load cell1");
     Serial.println(calibration_float);
     LoadCell3.setCalibrationFactor(calibration_float);
```

```
    }
    if ( LoadCell4.setNvsCalibrationVal(calibration_float) )
    {
      Serial.print("New calibration value is set for load cell1");
      Serial.println(calibration_float);
      LoadCell4.setCalibrationFactor(calibration_float);
    }
  }

  LoadCell1.updateHistory(date, height_int, age_int, gender, calibration_float);
  LoadCell2.updateHistory(date, height_int, age_int, gender, calibration_float);
  LoadCell3.updateHistory(date, height_int, age_int, gender, calibration_float);
  LoadCell4.updateHistory(date, height_int, age_int, gender, calibration_float);

  free(buf);
  return ESP_OK;
}

static esp_err_t status_handler(httpd_req_t *req)
{
    static char json_response[4096];
    parse_request_data(req);

    char *p = json_response;
    *p++ = '{';
    p += sprintf(p, "\"history\":[");
    for (int i = 0; i < MAX_HISTORY; i++) {
      p += sprintf(p, "{\"date\": \"%s\", \"weight\": %.2f, \"height\" : %d, \"age\" :
%d, \"gender\": \"%s\" },",
                  LoadCell1.getHistoricDate(i), LoadCell1.getHistoricWeight(i),
                  LoadCell1.getHistoricHeight(i), LoadCell1.getHistoricAge(i),
                  LoadCell1.getHistoricGender(i) );
    }
    p--; /* remove comma */
    p += sprintf(p, "], \"info\" : [{ \"avg_weight\":%.2f, \"weight\":%.2f, \"height\"
: %d, \"age\" : %d, \"gender\": \"%s\", \"calibration\":%.2f}, ",
        LoadCell1.getAverageWeight(), LoadCell1.getWeight(), LoadCell1.getHeight(),
LoadCell1.getAge(), LoadCell1.getGender(), LoadCell1.getCalibrationFactor());
    p += sprintf(p, "{ \"avg_weight\":%.2f, \"weight\":%.2f, \"height\" : %d, \"age\" :
%d, \"gender\": \"%s\", \"calibration\":%.2f}, ",
        LoadCell2.getAverageWeight(), LoadCell2.getWeight(), LoadCell2.getHeight(),
LoadCell2.getAge(), LoadCell2.getGender(), LoadCell2.getCalibrationFactor());
```

```
   p += sprintf(p, "{ \"avg_weight\":%.2f, \"weight\":%.2f, \"height\" : %d, \"age\" :
%d, \"gender\": \"%s\", \"calibration\":%.2f}, ",
       LoadCell3.getAverageWeight(), LoadCell3.getWeight(), LoadCell3.getHeight(),
LoadCell3.getAge(), LoadCell3.getGender(), LoadCell3.getCalibrationFactor());
   p += sprintf(p, "{ \"avg_weight\":%.2f, \"weight\":%.2f, \"height\" : %d, \"age\" :
%d, \"gender\": \"%s\", \"calibration\":%.2f}] ",
       LoadCell4.getAverageWeight(), LoadCell4.getWeight(), LoadCell4.getHeight(),
LoadCell4.getAge(), LoadCell4.getGender(), LoadCell4.getCalibrationFactor());

   *p++ = '}';
   *p++ = 0;
   Serial.print("json output: ");
   Serial.println(json_response);
   httpd_resp_set_type(req, "application/json");
   httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
   return httpd_resp_send(req, json_response, strlen(json_response));
}

static esp_err_t index_handler(httpd_req_t *req)
{
 httpd_resp_set_type(req, "text/html");
 return httpd_resp_send(req, (const char *)index_html,
sizeof(index_html)/sizeof(index_html[0]));
}

void startCameraServer()
{
   httpd_config_t config = HTTPD_DEFAULT_CONFIG();
   config.max_uri_handlers = 16;

   httpd_uri_t index_uri = {
       .uri = "/",
       .method = HTTP_GET,
       .handler = index_handler,
       .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
       ,
       .is_websocket = true,
       .handle_ws_control_frames = false,
       .supported_subprotocol = NULL
#endif
   };
```

```
   httpd_uri_t status_uri = {
        .uri = "/status",
        .method = HTTP_GET,
        .handler = status_handler,
        .user_ctx = NULL
#ifdef CONFIG_HTTPD_WS_SUPPORT
        ,
        .is_websocket = true,
        .handle_ws_control_frames = false,
        .supported_subprotocol = NULL
#endif
   };

   log_i("Starting web server on port: '%d'", config.server_port);
   if (httpd_start(&camera_httpd, &config) == ESP_OK)
   {
     httpd_register_uri_handler(camera_httpd, &index_uri);
     httpd_register_uri_handler(camera_httpd, &status_uri);
   }
}
```