Serveur HTTP ESP32

Un serveur Web est un composant logiciel qui écoute les requêtes HTTP entrantes des navigateurs Web. A réception d'une demande, le serveur Web envoie une réponse. Il peut s'agir du retour d'un document HTML à afficher dans un navigateur ou des données qui forment une réponse à un appel de service. Une requête HTTP peut également inclure des données à envoyer vers l'ESP32 pour traitement. Il existe de nombreuses implémentations de serveurs Web qui peuvent s'exécuter dans un environnement ESP32.

Le *framework* **Espressif (ESP-IDF)** fournit une API pour implémenter un serveur Web léger sur ESP32.

Liens :

- API HTTP Server
- Simple HTTPD Server Example
- API Wi-Fi
- Wi-Fi Examples
- SPIFFS example
- Simple HTTP File Server

Code source : esp32-serveur-http.zip

Wifi

On crée un projet avec le fichier `platformio.ini` suivant :

```
[env:lolin32]
platform = espressif32
board = lolin32
framework = espidf
board_build.partitions = partitions_singleapp.csv
```

Liens :

- API Wi-Fi
- Wi-Fi Examples

L'API WiFi d'Espressif (ESP-IDF) prend en charge la fonctionnalité réseau WiFi de l'ESP32. Cela inclut la configuration pour :

- Mode station (aka mode STA ou mode client WiFi) : l'ESP32 se connecte à un point d'accès.
- Mode AP (aka mode Soft-AP ou mode point d'accès) : les stations se connectent à l'ESP32.
- Mode AP-STA combiné (ESP32 est simultanément un point d'accès et une station connectée à un autre point d'accès).
- Différents modes de sécurité (WPA, WPA2, WEP, etc …).
- Recherche (*scan*) de points d'accès (analyse active et passive).

Connexion en mode station :

```
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "esp_system.h"
#include "esp_wifi.h"
```

```c
#include "esp_log.h"
#include "esp_event.h"
#include "nvs_flash.h"

#define ESP_WIFI_SSID       "Nom du réseau Wifi"
#define ESP_WIFI_PASSWORD   "Mot de passe du réseau Wifi"
#define NB_RESEAUX_MAX      16

/*
    Exemples :
    - https://github.com/espressif/esp-idf/blob/8bc19ba893e5544d571a753d82b44a84799b94b1/examples/wifi/getting_started/station/main/station_example_main.c
    - https://github.com/espressif/esp-idf/blob/8bc19ba893e5544d571a753d82b44a84799b94b1/examples/wifi/scan/main/scan.c
*/

static const char *TAG = "Test Wifi";
static bool demandeConnexion = false;

// Gestionnaire d'évènements
static void event_handler(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_data)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START)
    {
        if(demandeConnexion)
        {
            ESP_LOGI(TAG, "Connexion ...");
            esp_wifi_connect();
        }
    }
    else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED)
    {
        // reconnexion ?
        ESP_LOGI(TAG, "Deconnexion !");
    }
    else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP)
    {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "Adresse IP : " IPSTR, IP2STR(&event->ip_info.ip));
    }
}

static esp_err_t init_wifi()
{
    // initialise la pile TCP/IP
    ESP_ERROR_CHECK(esp_netif_init());
    // crée une boucle d'évènements
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    // crée
    esp_netif_t *sta_netif = esp_netif_create_default_wifi_sta();
    assert(sta_netif);

    // initialise la configuration Wifi par défaut
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    // initialise le WiFi avec la configuration par défaut et démarre également la tâche WiFi.
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    // installe le gestionnaire d'évènements Wifi
    /*
```

```
    esp_event_handler_instance_t instance_any_id;
    esp_event_handler_instance_t instance_got_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT, ESP_EVENT_ANY_ID, &event_hand
ler, NULL, &instance_any_id));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT, IP_EVENT_STA_GOT_IP, &event_han
dler, NULL, &instance_got_ip));
    */
    // version obsolète !
    ESP_ERROR_CHECK(esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, &event_handler, NULL
));
    ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, &event_handler, NUL
L));

    // définit le mode de fonctionnement station pour le WiFi
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));

    // démarre le WiFi selon la configuration
    esp_err_t ret = esp_wifi_start();
    ESP_ERROR_CHECK(ret);

    return ret;
}

static bool scan_wifi()
{
    bool present = false;
    uint16_t number = NB_RESEAUX_MAX;
    wifi_ap_record_t ap_info[NB_RESEAUX_MAX];
    uint16_t ap_count = 0;
    memset(ap_info, 0, sizeof(ap_info));

    // démarre le scan
    ESP_ERROR_CHECK(esp_wifi_scan_start(NULL, true));
    ESP_ERROR_CHECK(esp_wifi_scan_get_ap_records(&number, ap_info));
    ESP_ERROR_CHECK(esp_wifi_scan_get_ap_num(&ap_count));

    ESP_LOGI(TAG, "Nb reseaux trouves = %u", ap_count);
    for (int i = 0; (i < NB_RESEAUX_MAX) && (i < ap_count); i++)
    {
        // cf. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/
esp_wifi.html?highlight=wifi_ap_record_t#_CPPv416wifi_ap_record_t
        ESP_LOGI(TAG, "SSID \t\t%s", ap_info[i].ssid);
        ESP_LOGI(TAG, "RSSI \t\t%d", ap_info[i].rssi);
        ESP_LOGI(TAG, "Channel \t\t%d\n", ap_info[i].primary);
        if(strcmp((char *)ap_info[i].ssid, ESP_WIFI_SSID) == 0)
            present = true;
    }

    return present;
}

static void restart_wifi()
{
    ESP_ERROR_CHECK(esp_wifi_stop());
    ESP_ERROR_CHECK(esp_wifi_start());
}

static void connect_wifi()
{
    // configure la connexion Wifi du point d'accès (AP)
```

```
    wifi_config_t wifi_config = {
        .sta = {
        .ssid = ESP_WIFI_SSID,
        .password = ESP_WIFI_PASSWORD,
        .threshold.authmode = WIFI_AUTH_WPA2_PSK,
        .pmf_cfg = {
            .capable = true,
            .required = false
            },
        },
    };
    ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config));
    demandeConnexion = true;
    restart_wifi();
}

void app_main()
{
    // initialize NVS (nécessaire pour le Wifi)
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
    {
      ESP_ERROR_CHECK(nvs_flash_erase());
      ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    if(init_wifi() == ESP_OK)
    {
        if(scan_wifi())
        {
            connect_wifi();
        }
    }
}
```

Serveur HTTP

Le composant HTTP Server d'Espressif (ESP-IDF) permet d'exécuter un serveur Web léger sur ESP32. Les deux appels de base de l'API sont :

- `httpd_start()` : crée une instance de serveur HTTP, lui alloue les ressources en fonction de la configuration spécifiée et génère un *handle* vers l'instance de serveur. Le serveur possèdera notamment une socket d'écoute (TCP) pour le trafic HTTP. La priorité de la tâche et la taille de la pile sont configurables lors de la création de l'instance de serveur en passant la structure `httpd_config_t` à `httpd_start()`. Le trafic TCP est analysé en tant que requêtes HTTP et, en fonction de l'URI demandé, les gestionnaires enregistrés seront appelés pour renvoyer des paquets de réponse HTTP.

- `httpd_stop()` : arrête le serveur avec le*handle* fourni et libère les ressources associées. Il s'agit d'une fonction de blocage qui signale d'abord un arrêt de la tâche serveur, puis attend que la tâche se termine. Lors de l'arrêt, la tâche ferme toutes les connexions ouvertes, supprime les gestionnaires d'URI enregistrés et réinitialise toutes les données de contexte de session pour qu'elles soient vides.

Pour traiter les requêtes HTTP envoyées au serveur, il faudra enregistrer des gestionnaires d'URI avec :

- `httpd_register_uri_handler()` : enregistre un gestionnaire d'URI en passant un objet de type structure `httpd_uri_t` qui a des membres incluant le nom de l'RI, le type de méthode (par exemple

`HTTPD_GET` / `HTTPD_POST` / `HTTPD_PUT` etc …), un pointeur de fonction de type `esp_err_t * handler (httpd_req_t * req)` et pointeur `user_ctx` vers les données de contexte.

Liens :

- API HTTP Server
- Simple HTTPD Server Example

On utilisera le gestionnaire d'évènements pour le Wifi (`WIFI_EVENT`) pour démarrer (`IP_EVENT_STA_GOT_IP` déclenchera l'appel à `start_webserver()`) et arrêter (`WIFI_EVENT_STA_DISCONNECTED` déclenchera l'appel à `stop_webserver()`) le serveur HTTP.

Le gestionnaire d'URI pour une requête `GET /test` (`get_handler()`) montre une utilisation simple des différents appels de l'API pour traiter ce type de requête et renvoyer une réponse au client.

```
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_log.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include <sys/param.h>

#define ESP_WIFI_SSID       "Nom du réseau Wifi"
#define ESP_WIFI_PASSWORD   "Mot de passe du réseau Wifi"
#define NB_RESEAUX_MAX      16

#define WEB_SERVER_PORT     80

static const char *TAG = "Test Serveur HTTP";
static bool demandeConnexion = false;

httpd_handle_t server_httpd = NULL;

httpd_handle_t start_webserver();
void stop_webserver(httpd_handle_t server);

// Gestionnaire d'évènements
static void event_handler(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_d
ata)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START)
    {
        if(demandeConnexion)
        {
            ESP_LOGI(TAG, "Connexion ...");
            esp_wifi_connect();
        }
    }
    else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED)
    {
        // reconnexion ?
        ESP_LOGI(TAG, "Deconnexion !");
        stop_webserver(server_httpd);
    }
```

```
    else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP)
    {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "Adresse IP : " IPSTR, IP2STR(&event->ip_info.ip));
        if (server_httpd == NULL)
        {
            ESP_LOGI(TAG, "Demarrage serveur HTTP");
            server_httpd = start_webserver();
        }
    }
}

static esp_err_t init_wifi()
{
    // ...

    return ret;
}

static bool scan_wifi()
{
    bool present = false;
    // ...

    return present;
}

static void restart_wifi()
{
    ESP_ERROR_CHECK(esp_wifi_stop());
    ESP_ERROR_CHECK(esp_wifi_start());
}

static void connect_wifi()
{
    // configure la connexion Wifi du point d'accès (AP)
    wifi_config_t wifi_config = {
        .sta = {
        .ssid = ESP_WIFI_SSID,
        .password = ESP_WIFI_PASSWORD,
        .threshold.authmode = WIFI_AUTH_WPA2_PSK,
        .pmf_cfg = {
            .capable = true,
            .required = false
            },
        },
    };
    ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config));
    demandeConnexion = true;
    restart_wifi();
}

// Gestionnaire d'URI pour une requête GET /test
esp_err_t get_handler(httpd_req_t *req)
{
    char*  buf;
    size_t buf_len;

    // recherche "Host" dans l'entête de la requête reçue
    buf_len = httpd_req_get_hdr_value_len(req, "Host") + 1;
```

```
    if (buf_len > 1)
    {
        buf = malloc(buf_len);
        // copie la valeur dans un buffer
        if (httpd_req_get_hdr_value_str(req, "Host", buf, buf_len) == ESP_OK)
        {
            ESP_LOGI(TAG, "Header => Host: %s", buf);
        }
        free(buf);
    }

    // récupère la longueur de la chaîne de la requête
    buf_len = httpd_req_get_url_query_len(req) + 1;
    if (buf_len > 1)
    {
        buf = malloc(buf_len);
        // récupère la requête
        if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK)
        {
            ESP_LOGI(TAG, "Requete => %s", buf);
            char param[32];
            // récupère la valeur d'un paramètre de la requête (exemple : ?param1=val1&param2=val
2)
            if (httpd_query_key_value(buf, "export", param, sizeof(param)) == ESP_OK)
            {
                ESP_LOGI(TAG, "Parametre => export=%s", param);
            }
        }
        free(buf);
    }

    // définit le type de contenu HTTP (par défaut : "text/html")
    if (httpd_resp_set_type(req, "text/plain") == ESP_OK)
    {
        // ajoute des en-têtes supplémentaires
        httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
    }

    // envoie une réponse
    const char resp[] = "It works !";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);

    return ESP_OK;
}

// Gestionnaire d'URI pour une requête POST /test
esp_err_t post_handler(httpd_req_t *req)
{
    char content[128];

    ESP_LOGI(TAG, "Longueur contenu POST : %d", req->content_len);

    // tronque si la longueur du contenu est supérieure au buffer
    size_t recv_size = MIN(req->content_len, sizeof(content));

    int ret = httpd_req_recv(req, content, recv_size);

    // déconnecté ?
    if (ret <= 0)
    {
```

```
        // timeout ?
        if (ret == HTTPD_SOCK_ERR_TIMEOUT)
        {
            // retourne une erreur HTTP 408 (Request Timeout)
            httpd_resp_send_408(req);
        }

        return ESP_FAIL;
    }
    ESP_LOGI(TAG, "Contenu POST : %s", content);

    // envoie une réponse
    const char resp[] = "Test Reponse POST";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);

    return ESP_OK;
}

httpd_handle_t start_webserver()
{
    // initialise la configuration par défaut du serveur HTTP
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = WEB_SERVER_PORT;

    // pour une requête GET /test
    httpd_uri_t uri_get = {
        .uri      = "/test",
        .method   = HTTP_GET,
        .handler  = get_handler,
        .user_ctx = NULL
    };

    // pour une requête POST /test
    httpd_uri_t uri_post = {
        .uri      = "/test",
        .method   = HTTP_POST,
        .handler  = post_handler,
        .user_ctx = NULL
    };

    ESP_LOGI(TAG, "Port serveur HTTP : '%d'", config.server_port);

    httpd_handle_t server = NULL;
    // Démarre le serveur HTTP
    if (httpd_start(&server, &config) == ESP_OK)
    {
        // enregistre les gestionnaires d'URI
        httpd_register_uri_handler(server, &uri_get);
        httpd_register_uri_handler(server, &uri_post);
    }

    return server;
}

void stop_webserver(httpd_handle_t server)
{
    if (server)
    {
        // arrête le serveur HTTP
        httpd_stop(server);
```

```
    }
}

void app_main()
{
    // initialize NVS (nécessaire pour le Wifi)
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
    {
      ESP_ERROR_CHECK(nvs_flash_erase());
      ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    if(init_wifi() == ESP_OK)
    {
        if(scan_wifi())
        {
            connect_wifi();
        }
    }
}
```

Exemple pour une requête **GET** `http://192.168.52.24/test?export=json&lang=fr` :

```
I (2984) Test Serveur HTTP: Connexion ...
I (3164) wifi:connected with Livebox-86D0, aid = 4, channel 1, BW20, bssid = a8:6a:bb:4e:86:d0
I (3164) wifi:security: WPA2-PSK, phy: bgn, rssi: -43
I (3174) wifi:pm start, type: 1
I (4164) esp_netif_handlers: sta ip: 192.168.52.24, mask: 255.255.255.0, gw: 192.168.52.1
I (4164) Test Serveur HTTP: Adresse IP : 192.168.52.24
I (4164) Test Serveur HTTP: Demarrage serveur HTTP
I (4174) Test Serveur HTTP: Port serveur HTTP : '80'
I (101194) Test Serveur HTTP: Header => Host: 192.168.52.24
I (101194) Test Serveur HTTP: Requete => export=json&lang=fr
I (101194) Test Serveur HTTP: Parametre => export=json
```

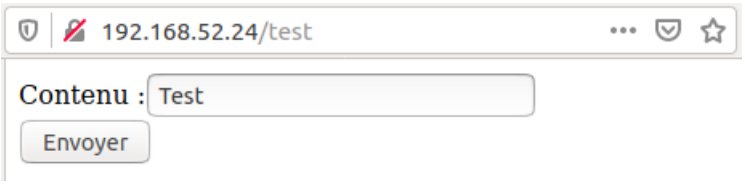
Remarque : Seuls les gestionnaires d'URI installés répondront aux requêtes

Exemple pour une requête **GET** `http://192.168.52.24/` :

```
W (5259844) httpd_uri: httpd_uri: URI '/' not found
W (5259844) httpd_txrx: httpd_resp_send_err: 404 Not Found - This URI does not exist
```

Pour tester la méthode **POST**, on va modifier le gestionnaire d'URI `GET /test` pour qu'il envoie un simple formulaire HTML :

```
// Gestionnaire d'URI pour une requête GET /test
esp_err_t get_handler(httpd_req_t *req)
{
    // définit le type de contenu HTTP
    if (httpd_resp_set_type(req, "text/html") == ESP_OK)
    {
        // ajoute des en-têtes supplémentaires
        httpd_resp_set_hdr(req, "Content-Encoding", "identity");
    }

    const char header[] = "<!doctype html><html><head><title>Serveur HTTP ESP32</title></head><bo
dy><form action=\"http://";
    const char footer[] = "/test\" method=\"post\"><label for=\"mytext\">Contenu :</label><input
type=\"text\" id=\"mytext\" name=\"mytext\" value=\"Test\"/><br/><button>Envoyer</button></form><
/body></html>";
    char response[512] = "";
    strcat(response, header);
    strcat(response, adresseIP);
    strcat(response, footer);

    httpd_resp_send(req, response, HTTPD_RESP_USE_STRLEN);

    return ESP_OK;
}

// Gestionnaire d'URI pour une requête POST /test
esp_err_t post_handler(httpd_req_t *req)
{
    char content[128];

    ESP_LOGI(TAG, "Longueur contenu POST : %d", req->content_len);

    // tronque si la longueur du contenu est supérieure au buffer
    size_t recv_size = MIN(req->content_len, sizeof(content));

    int ret = httpd_req_recv(req, content, recv_size);

    // déconnecté ?
    if (ret <= 0)
    {
        // timeout ?
        if (ret == HTTPD_SOCK_ERR_TIMEOUT)
        {
            // retourne une erreur HTTP 408 (Request Timeout)
            httpd_resp_send_408(req);
        }

        return ESP_FAIL;
    }
    ESP_LOGI(TAG, "Contenu POST : %s", content);

    // envoie une réponse
    const char resp[] = "Test Reponse POST";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);

    return ESP_OK;
}
```



On obtient après avoir envoyé le formulaire :

```
I (4664) Test Serveur HTTP: Demarrage serveur HTTP
I (4674) Test Serveur HTTP: Port serveur HTTP : '80'
I (12754) Test Serveur HTTP: Longueur contenu POST : 11
I (12764) Test Serveur HTTP: Contenu POST : mytext=Test?
```

Il est possible de tester la méthode**POST** avec `curl` et l'option `-d`, par exemple :

```
$ curl --header "Content-Type: application/json" -d "{\"data\":\"value\"}" http://192.168.52.24/test
Test Reponse POST
```

Le système de fichier SPIFFS

Il est possible de stocker le contenu du site web (fichier HTML et CSS) dans la mémoire de l'ESP32 en utilisant le système de fichiers SPIFFS.

Dans PlatformIO, il faut commencer par créer un dossier `data` (même niveau que le dossier `src`) puis y placez les fichiers.

Remarque : il est possible de spécifier son propre emplacement. Dans ce cas, il faut renseigner la variable `data_dir` dans le fichier `platform.ini`.

Pour l'exemple, on va placer les deux fichiers suivants :

- `index.html` :

```
<!DOCTYPE html>
<html>
<head>
  <title>Serveur HTTP ESP32</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" href="data:,">
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <h1>Serveur HTTP ESP32</h1>
  <p>It works !</p>
</body>
</html>
```

- `style.css` :

```
html{font-family:Helvetica;display:inline-block;margin:0px auto;text-align:center;}
h1{color:#0F3376;padding:2vh;}
p{font-size:1.5rem;}
```

Ensuite, il faut *uploader* les fichiers dans l'ESP32 avec la commande :

```
$  platformio run --target uploadfs
```

Lien : SPIFFS example

Pour utiliser SPIFFS, il faudra l'initialiser avec cette fonction :

```
// initialise SPIFFS
static esp_err_t init_spiffs()
{
    ESP_LOGI(TAG, "Initialise SPIFFS");

    esp_vfs_spiffs_conf_t conf =
    {
      .base_path = "/spiffs",
      .partition_label = NULL,
      .max_files = 5,
      .format_if_mount_failed = true
    };

    esp_err_t ret = esp_vfs_spiffs_register(&conf);
    if (ret != ESP_OK)
    {
        if (ret == ESP_FAIL)
        {
            ESP_LOGE(TAG, "Failed to mount or format filesystem !");
        } else if (ret == ESP_ERR_NOT_FOUND)
        {
            ESP_LOGE(TAG, "Failed to find SPIFFS partition !");
        } else
        {
            ESP_LOGE(TAG, "Failed to initialize SPIFFS (%s) !", esp_err_to_name(ret));
        }
        return ESP_FAIL;
    }

    size_t total = 0, used = 0;
    ret = esp_spiffs_info(NULL, &total, &used);
    if (ret != ESP_OK)
    {
        ESP_LOGE(TAG, "Failed to get SPIFFS partition information (%s) !", esp_err_to_name(ret));
        return ESP_FAIL;
    }

    ESP_LOGI(TAG, "Taille partition : total = %d - utilise = %d", total, used);

    return ESP_OK;
}
```

On crée le fichier `partitions_example.csv` à la racine du projet de PlatformIO qui spécifie la partition SPIFFS :

```
# Name,   Type, SubType, Offset,  Size, Flags
# Note: if you have increased the bootloader size, make sure to update the offsets to avoid overl
ap
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 1M,
storage,  data, spiffs,  ,        0xF0000,
```

Et on modifie le fichier `platformio.ini` pour prendre en compte la modification suivante :

```
[env:lolin32]
platform = espressif32
board = lolin32
framework = espidf
#board_build.partitions = partitions_singleapp.csv
board_build.partitions = partitions_example.csv
```

Site web

On va maintenant modifier le serveur HTTP pour intégrer la page HTML `index.html`.

Lien : Simple HTTP File Server Example

Tout d'abord dans la fonction `start_webserver()`, on va rediriger toutes les requêtes GET vers le gestionnaire d'URI `get_handler()` :

```
httpd_handle_t start_webserver()
{
    // initialise la configuration par défaut du serveur HTTP
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = WEB_SERVER_PORT;
    config.uri_match_fn = httpd_uri_match_wildcard;

    // pour une requête GET
    httpd_uri_t uri_get = {
        .uri      = "/*", // cf. httpd_uri_match_wildcard
        .method   = HTTP_GET,
        .handler  = get_handler,
        .user_ctx = NULL
    };

    ESP_LOGI(TAG, "Port serveur HTTP : '%d'", config.server_port);

    httpd_handle_t server = NULL;
    // Démarre le serveur HTTP
    if (httpd_start(&server, &config) == ESP_OK)
    {
        // enregistre les gestionnaires d'URI
        httpd_register_uri_handler(server, &uri_get);
    }

    return server;
}
```

Le gestionnaire d'URI `get_handler()` va maintenant réaliser le traitement suivant :

- lister les fichiers de la partition SPIFFS (fonction `http_resp_dir_html()`) pour une requête GET sur la racine `/`
- rediriger vers la fonction `index_html_get_handler()` pour une requête GET `/index.htm`
- sinon lire et renvoyer le contenu du fichier demandé (une requête GET `/index.html` par exemple)

```
// Gestionnaire d'URI pour les requêtes GET
esp_err_t get_handler(httpd_req_t *req)
{
    char filepath[FILE_PATH_MAX];
    FILE *fd = NULL;
    struct stat file_stat;

    ESP_LOGI(TAG, "URI : %s", req->uri);

    const char *filename = get_path_from_uri(filepath, base_path, req->uri, sizeof(filepath));
    if (!filename)
    {
        ESP_LOGE(TAG, "Filename is too long");
        // retourne une erreur 500 (Internal Server Error)
        httpd_resp_send_err(req, HTTPD_500_INTERNAL_SERVER_ERROR, "Filename too long");
        return ESP_FAIL;
    }

    ESP_LOGI(TAG, "filename : %s", filename);

    if (filename[strlen(filename) - 1] == '/')
    {
        return http_resp_dir_html(req, filepath);
    }

    if (stat(filepath, &file_stat) == -1)
    {
        if (strcmp(filename, "/index.htm") == 0)
        {
            return index_html_get_handler(req);
        }
        ESP_LOGE(TAG, "Failed to stat file : %s", filepath);
        // retourne une erreur 404 Not Found
        httpd_resp_send_err(req, HTTPD_404_NOT_FOUND, "File does not exist");
        return ESP_FAIL;
    }

    fd = fopen(filepath, "r");
    if (!fd)
    {
        ESP_LOGE(TAG, "Failed to read existing file : %s", filepath);
        // retourne une erreur 500 Internal Server Error
        httpd_resp_send_err(req, HTTPD_500_INTERNAL_SERVER_ERROR, "Failed to read existing file")
;
        return ESP_FAIL;
    }

    ESP_LOGI(TAG, "Sending file : %s (%ld bytes)...", filename, file_stat.st_size);
    set_content_type_from_file(req, filename);

    // Retrieve the pointer to scratch buffer for temporary storage
    char*  buf;
    size_t chunksize;
    buf = malloc(BUFSIZE);
```

```
    do
    {
        // lit une partie du fichier
        chunksize = fread(buf, 1, BUFSIZE, fd);

        if (chunksize > 0)
        {
            // envoie le contenu du buffer
            if (httpd_resp_send_chunk(req, buf, chunksize) != ESP_OK)
            {
                fclose(fd);
                ESP_LOGE(TAG, "File sending failed !");
                httpd_resp_sendstr_chunk(req, NULL);
                // retourne une erreur 500 Internal Server Error
                httpd_resp_send_err(req, HTTPD_500_INTERNAL_SERVER_ERROR, "Failed to send file");
               return ESP_FAIL;
            }
        }
    } while (chunksize != 0);

    fclose(fd);
    free(buf);
    ESP_LOGI(TAG, "File sending complete");

    httpd_resp_send_chunk(req, NULL, 0);

    return ESP_OK;
}
```

La fonction `http_resp_dir_html()` liste les fichiers de la partition SPIFFS et fournit un affichage sous forme d'un simple tableau HTML :

```
static esp_err_t http_resp_dir_html(httpd_req_t *req, const char *dirpath)
{
    char entrypath[FILE_PATH_MAX];
    char entrysize[16];
    const char *entrytype;

    struct dirent *entry;
    struct stat entry_stat;

    DIR *dir = opendir(dirpath);
    const size_t dirpath_len = strlen(dirpath);

    strlcpy(entrypath, dirpath, sizeof(entrypath));

    if (!dir)
    {
        ESP_LOGE(TAG, "Failed to stat dir : %s", dirpath);
        // retourne une erreur 404
        httpd_resp_send_err(req, HTTPD_404_NOT_FOUND, "Directory does not exist");
        return ESP_FAIL;
    }

    // envoie l'en-tête HTML
    httpd_resp_sendstr_chunk(req, "<!DOCTYPE html><html><body>");
```

```
    // envoie l'en-tête du tableau HTML
    httpd_resp_sendstr_chunk(req,
        "<table class=\"fixed\" border=\"1\">"
        "<col width=\"800px\" /><col width=\"300px\" /><col width=\"300px\" />"
        "<thead><tr><th>Nom</th><th>Type</th><th>Taille (octets)</th></tr></thead>"
        "<tbody>");

    // lit les noms et tailles des fichiers dans le répertoire
    while ((entry = readdir(dir)) != NULL)
    {
        entrytype = (entry->d_type == DT_DIR ? "directory" : "file");

        strlcpy(entrypath + dirpath_len, entry->d_name, sizeof(entrypath) - dirpath_len);
        if (stat(entrypath, &entry_stat) == -1)
        {
            ESP_LOGE(TAG, "Failed to stat %s : %s", entrytype, entry->d_name);
            continue;
        }
        sprintf(entrysize, "%ld", entry_stat.st_size);
        ESP_LOGI(TAG, "Found %s : %s (%s bytes)", entrytype, entry->d_name, entrysize);

        httpd_resp_sendstr_chunk(req, "<tr><td><a href=\"");
        httpd_resp_sendstr_chunk(req, req->uri);
        httpd_resp_sendstr_chunk(req, entry->d_name);
        if (entry->d_type == DT_DIR)
        {
            httpd_resp_sendstr_chunk(req, "/");
        }
        httpd_resp_sendstr_chunk(req, "\">");
        httpd_resp_sendstr_chunk(req, entry->d_name);
        httpd_resp_sendstr_chunk(req, "</a></td><td>");
        httpd_resp_sendstr_chunk(req, entrytype);
        httpd_resp_sendstr_chunk(req, "</td><td>");
        httpd_resp_sendstr_chunk(req, entrysize);
        httpd_resp_sendstr_chunk(req, "</td></tr>\n");
    }
    closedir(dir);

    httpd_resp_sendstr_chunk(req, "</tbody></table>");
    httpd_resp_sendstr_chunk(req, "</body></html>");
    httpd_resp_sendstr_chunk(req, NULL);
    return ESP_OK;
}
```

La fonction `index_html_get_handler()` assure une redirection vers la racine `/` du site web :

```
static esp_err_t index_html_get_handler(httpd_req_t *req)
{
    httpd_resp_set_status(req, "307 Temporary Redirect");
    httpd_resp_set_hdr(req, "Location", "/");
    httpd_resp_send(req, NULL, 0);
    return ESP_OK;
}
```

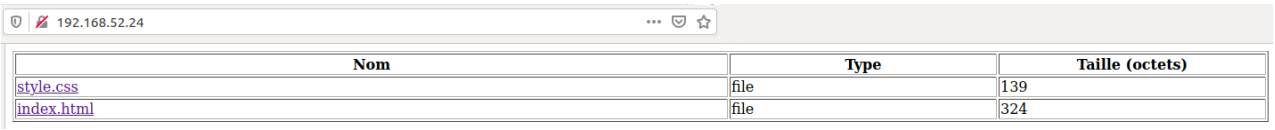
Pour finir, queques fonctions utilitaires récupérées de l'exemple Simple HTTP File Server :

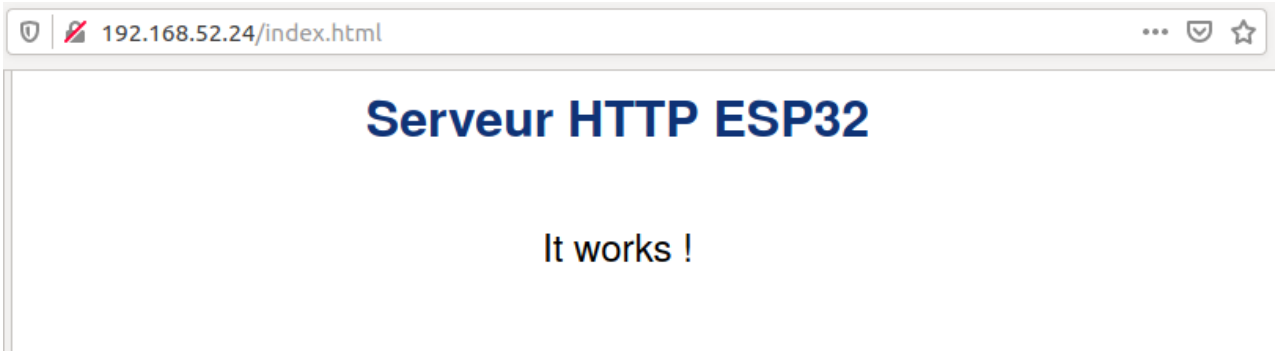
```
#include <stdio.h>
#include <string.h>
```

```
#include <sys/param.h>
#include <sys/unistd.h>
#include <sys/stat.h>
#include <dirent.h>
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_log.h"
#include "esp_event.h"
#include "esp_http_server.h"
#include "esp_vfs.h"
#include "esp_spiffs.h"
#include "nvs_flash.h"
#include <sys/param.h>

#define ESP_WIFI_SSID       "Votre WIFI_SSID"
#define ESP_WIFI_PASSWORD   "Votre WIFI_PASSWORD"
#define NB_RESEAUX_MAX      16

#define WEB_SERVER_PORT     80

#define FILE_PATH_MAX (ESP_VFS_PATH_MAX + CONFIG_SPIFFS_OBJ_NAME_LEN)
#define BUFSIZE             8192

static const char *TAG = "Test Serveur HTTP";
static bool demandeConnexion = false;

httpd_handle_t server_httpd = NULL;
char adresseIP[24];
const char *base_path = "/spiffs";

...

static const char* get_path_from_uri(char *dest, const char *base_path, const char *uri, size_t d
estsize)
{
    const size_t base_pathlen = strlen(base_path);
    size_t pathlen = strlen(uri);

    const char *quest = strchr(uri, '?');
    if (quest)
    {
        pathlen = MIN(pathlen, quest - uri);
    }
    const char *hash = strchr(uri, '#');
    if (hash)
    {
        pathlen = MIN(pathlen, hash - uri);
    }

    if (base_pathlen + pathlen + 1 > destsize)
    {
        return NULL;
    }

    // construit le chemin complet (base + path)
    strcpy(dest, base_path);
    strlcpy(dest + base_pathlen, uri, pathlen + 1);
```

```
    // retourne le pointeur ver les chemin (sans la base)
    return dest + base_pathlen;
}

#define IS_FILE_EXT(filename, ext) \
    (strcasecmp(&filename[strlen(filename) - sizeof(ext) + 1], ext) == 0)

static esp_err_t set_content_type_from_file(httpd_req_t *req, const char *filename)
{
    if (IS_FILE_EXT(filename, ".pdf"))
    {
        return httpd_resp_set_type(req, "application/pdf");
    }
    else if (IS_FILE_EXT(filename, ".html"))
    {
        return httpd_resp_set_type(req, "text/html");
    }
    else if (IS_FILE_EXT(filename, ".css"))
    {
        return httpd_resp_set_type(req, "text/css");
    }
    else if (IS_FILE_EXT(filename, ".jpeg"))
    {
        return httpd_resp_set_type(req, "image/jpeg");
    }
    else if (IS_FILE_EXT(filename, ".ico"))
    {
        return httpd_resp_set_type(req, "image/x-icon");
    }
    return httpd_resp_set_type(req, "text/plain");
}

void app_main()
{
    // initialize NVS (nécessaire pour le Wifi)
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
    {
      ESP_ERROR_CHECK(nvs_flash_erase());
      ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    init_spiffs();

    if(init_wifi() == ESP_OK)
    {
        if(scan_wifi())
        {
            connect_wifi();
        }
    }
}
```

Le serveur HTTP en action :





On obtient dans le moniteur :

```
I (4257) Test Serveur HTTP: Adresse IP : 192.168.52.24
I (4257) Test Serveur HTTP: Demarrage serveur HTTP
I (4267) Test Serveur HTTP: Port serveur HTTP : '80'
I (19457) Test Serveur HTTP: URI : /
I (19457) Test Serveur HTTP: filename : /
I (19467) Test Serveur HTTP: Found file : style.css (139 bytes)
I (19507) Test Serveur HTTP: Found file : index.html (324 bytes)
I (36957) Test Serveur HTTP: URI : /index.html
I (36957) Test Serveur HTTP: filename : /index.html
I (36957) Test Serveur HTTP: Sending file : /index.html (324 bytes)...
I (36967) Test Serveur HTTP: File sending complete
I (37267) Test Serveur HTTP: URI : /style.css
I (37267) Test Serveur HTTP: filename : /style.css
I (37507) Test Serveur HTTP: Sending file : /style.css (139 bytes)...
I (37517) Test Serveur HTTP: File sending complete
```

Code source : esp32-serveur-http.zip

Framework Arduino

Le *framework* **Arduino** s'appuie sur le *framework* **Espressif (ESP-IDF)** et permet d'intégrer de nombreuses bibliothèques.

Dans PlatformIO, il faudra le spécifier à la création du projet. Le fichier `platformio.ini` sera le suivant par exemple :

```
[env:lolin32]
platform = espressif32
board = lolin32
framework = arduino
```

Liens :

- API Wi-Fi
- Examples

Exemple pour une connexion Wifi en mode station :

```
#include <WiFi.h>

const char* ssid = "Nom du réseau Wifi";
const char* password = "Mot de passe du réseau Wifi";

void setup()
{
    Serial.begin(115200);

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

void loop()
{
}
```

Exemple pour une connexion Wifi en mode point d'accès :

```
#include <WiFi.h>

const char* ssid = "Nom du réseau Wifi";
const char* password = "Mot de passe du réseau Wifi";

void setup()
{
    Serial.begin(115200);

    WiFi.softAP(ssid, password);
    Serial.print("Adresse IP : ");
    Serial.println(WiFi.softAPIP());
}

void loop()
{
}
```

etc …

Voir aussi

- WiFiManager
- WebServer
- ESPAsyncWebServer