Laboratoire 6 - Programmation OTA (Over The Air)

6.1 Introduction

La **programmation OTA** permet la mise à jour - téléchargement d'un nouveau programme sur ESP32 v**ia Wi-Fi** au lieu de forcer l'utilisateur à connecter l'ESP32 à un ordinateur via USB pour effectuer la mise à jour. La fonctionnalité **OTA** est extrêmement utile s'il n'y a pas d'accès physique au module ESP. Cela permet de réduire le temps passé à mettre à jour chaque module ESP pendant la maintenance.
Une caractéristique importante de l'**OTA** est qu'un seul emplacement central peut envoyer une mise à jour à **plusieurs ESP** partageant le même réseau.
Le seul inconvénient est que vous devez ajouter du **code supplémentaire** pour OTA à chaque programme que vous téléchargez, afin de pouvoir utiliser OTA dans la prochaine mise à jour.

6.1.1 Mémoire flash de ESP32

Avant de passer à la programmation nous allons étudier les partitions de la mémoire flash d'un ESP32.
La mémoire flash est divisée en plusieurs partitions logiques pour stocker divers composants. La manière typique de procéder est illustrée dans la figure.

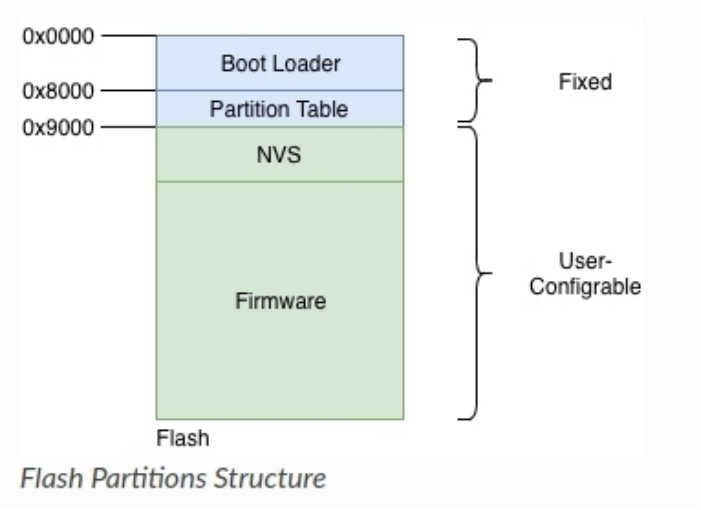

Figure 6.1 Partitions de la mémoire flash

Comme on peut le voir sur la figure dessus, la structure est statique jusqu'à l'adresse flash `0x9000`.
La première partie du flash contient le code du **boot**, qui est immédiatement suivi de la table de partition. La table de partition stocke ensuite la manière dont le reste du flash doit être interprété.
En règle générale, une installation aura au moins 1 partition **NVS** (**Non Volatile Storage** – **WiFi credentials** , ..) et 1 partition pour le code utilisateur

6.1.2 Mécanisme OTA

Pour les mises à niveau du code, un schéma de partition **actif-passif** est utilisé.

Deux partitions flash sont réservées au composant 'firmware', comme indiqué sur la figure suivante.
La partition **OTA-Data** se souvient laquelle de ces dernières est la partition active.

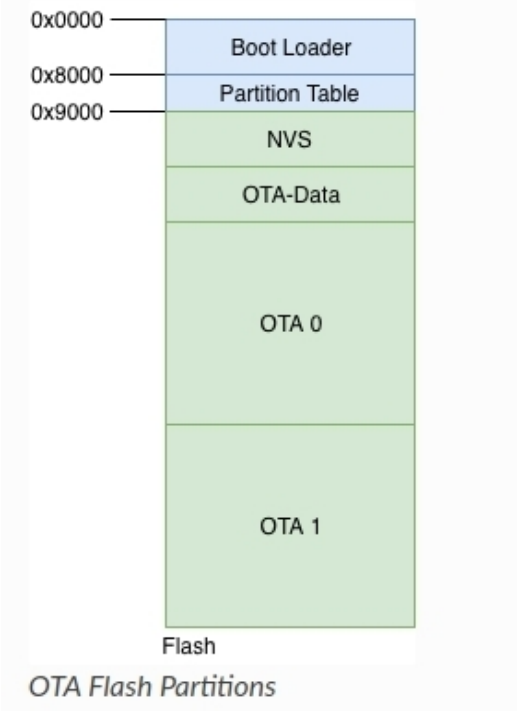

Figure 6.2 Partition **OTA-Data** et partitions **active-passive** pour le code

Les changements d'état typiques qui se produisent dans le flux de travail de mise à niveau du **code OTA** sont indiqués dans la **Figure 6.3.**

1. **Étape 0**: **OTA 0** est le **code actif**. La partition de données **OTA** stocke ces informations comme on peut le voir.
2. **Étape 1**: Le processus de **mise à niveau du code** commence. La partition passive est identifiée, effacée et un nouveau code est en cours d'écriture sur la partition **OTA 1**.
3. **Étape 2**: La mise à niveau du code est entièrement écrite et la vérification est en cours.
4. **Étape 3**: la mise à niveau du code est réussie, la partition de données OTA est mise à jour pour indiquer que **OTA 1** est désormais la **partition active**. Au prochain démarrage, le code de cette partition démarrera.

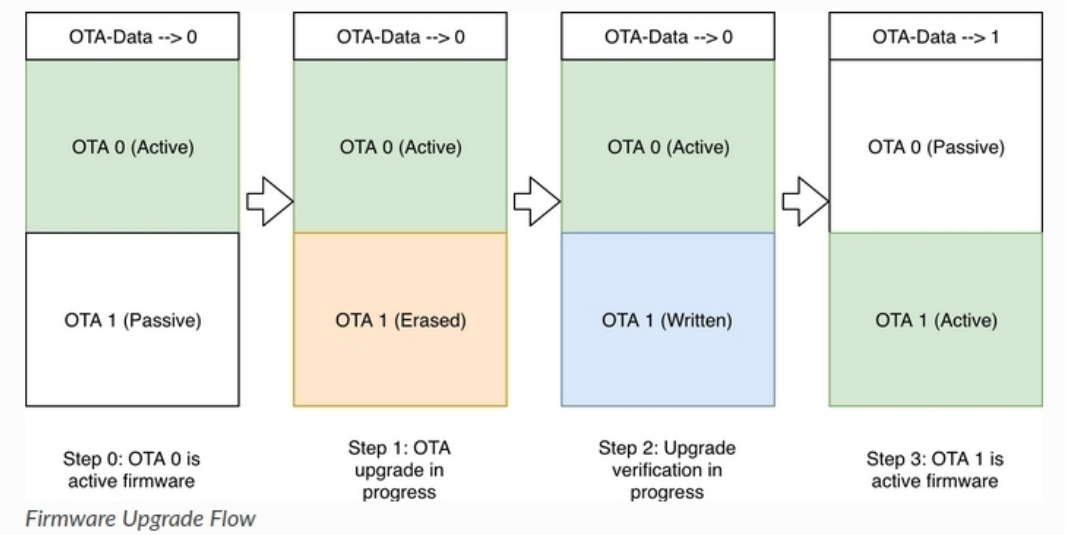

Figure 6.3 Flux de la mise à niveau du code dans la mémoire flash

6.2 Implémentation d'OTA sur carte ESP32 par OTA de base

Il existe **trois façons** d'implémenter la fonctionnalité OTA dans ESP32.

1. **OTA de base** - Les mises à jour Over-The-Air sont envoyées via **Arduino IDE** ou **PlatformIO**.
2. **Web Updater OTA** - Les mises à jour en direct sont envoyées via un **navigateur Web**.
3. La bibliothèque **WebOTA** permet également d'envoyer le croquis Arduino compilé (`.bin`) directement via l'**interface WEB**.

Chacun moyen a ses propres avantages. Vous pouvez les implémenter selon les besoins de votre projet.

6.2.1 La mise en œuvre de l'OTA de base

Pour commencer, téléchargez le micrologiciel OTA de base via un port série. Il s'agit d'une étape obligatoire pour pouvoir effectuer les prochaines mises à jour - téléchargements via WiFi.

Dans la phase suivante, vous pouvez télécharger de nouveaux programmes sur ESP32 à partir d'Arduino IDE via les airs - **WiFi avec une adresse IP**.

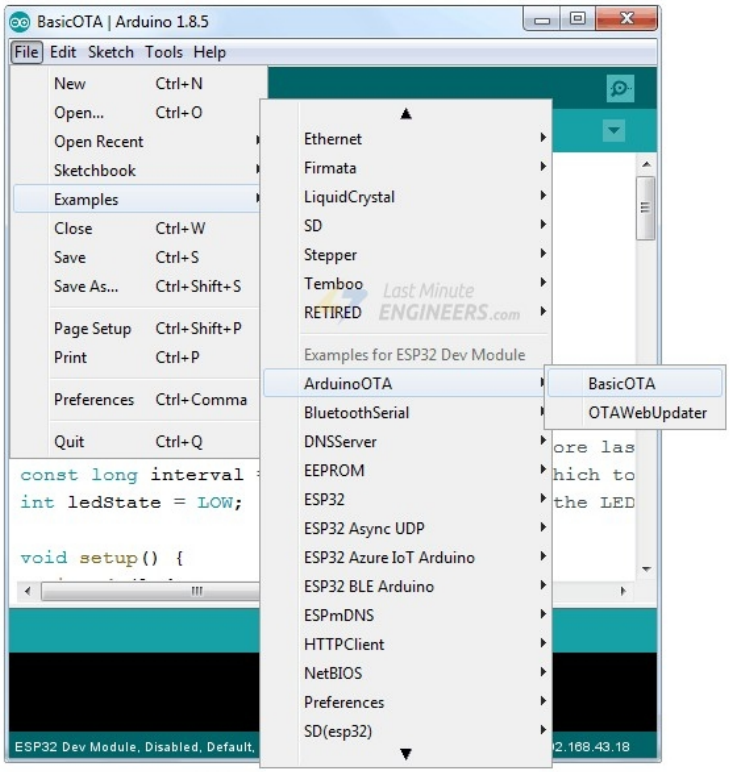

Figure 6.4 La sélection du code **OTA** de base

Avant de télécharger le code, vous devez apporter des modifications pour le rendre opérationnel. Vous devez modifier les deux variables suivantes avec vos informations d'identification réseau, afin qu'ESP32 puisse établir une connexion avec le réseau existant.

```
const char* ssid = ".........";
const char* password = ".........";
```

Une fois que vous avez terminé, téléchargez le code par **câble USB**.

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
const char* ssid = "PhoneAP";
const char* password = "smartcomputerlab";

void setup() {
  Serial.begin(9600);
  Serial.println("Booting");
  pinMode(led,OUTPUT);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
  }
  ArduinoOTA.onStart([]() {
      String type;
      if (ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
      else // U_SPIFFS
```

```
        type = "filesystem";
      Serial.println("Start updating " + type);
    })
    .onEnd([]() {
      Serial.println("\nEnd");
    })
    .onProgress([](unsigned int progress, unsigned int total) {
      Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
    })
    .onError([](ota_error_t error) {
      Serial.printf("Error[%u]: ", error);
      if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
      else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
      else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
      else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
      else if (error == OTA_END_ERROR) Serial.println("End Failed");
    });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  ArduinoOTA.handle();
}
```

Maintenant, ouvrez le moniteur série à un débit de 9600 bauds. Si tout va bien, l'**adresse IP dynamique** obtenue de votre routeur sera affichée. **Écrivez le**.

6.2.2 Télécharger un nouveau code via WiFi

Maintenant, téléchargeons un nouveau programme.
Il est nécessaire d'**ajouter le code** pour **OTA** dans chaque croquis que vous téléchargez. Sinon, vous perdrez la capacité OTA et ne pourrez plus effectuer de futurs téléchargements en direct. Il est donc recommandé de modifier le code ci-dessus pour inclure votre nouveau code.

À titre d'exemple, nous allons inclure un simple sketch Blink dans le code OTA de base. N'oubliez pas de modifier les variables **`ssid`** et **`password`** avec vos informations d'identification réseau.

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
const char* ssid = "PhoneAP";
const char* password = "smartcomputerlab";
//variables for blinking an LED with Millis
const int led = 25; // ESP32 Pin to which onboard LED is connected
unsigned long previousMillis = 0;  // will store last time LED was updated
const long interval = 1000;  // interval at which to blink (milliseconds)
int ledState = LOW;  // ledState used to set the LED

void setup() {
  Serial.begin(9600);
  Serial.println("Booting");
  pinMode(led,OUTPUT);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000); ESP.restart();
```

```
  }

  ArduinoOTA.onStart([]() {
      String type;
      if (ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
      else // U_SPIFFS
        type = "filesystem";
      // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS
using SPIFFS.end()
      Serial.println("Start updating " + type);
    })
    .onEnd([]() {
      Serial.println("\nEnd");
    })
    .onProgress([](unsigned int progress, unsigned int total) {
      Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
    })
    .onError([](ota_error_t error) {
      Serial.printf("Error[%u]: ", error);
      if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
      else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
      else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
      else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
      else if (error == OTA_END_ERROR) Serial.println("End Failed");
    });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}
void loop() {
  ArduinoOTA.handle();
  //loop to blink without delay
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
  // save the last time you blinked the LED
  previousMillis = currentMillis;
  // if the LED is off turn it on and vice-versa:
  ledState = not(ledState);
  // set the LED with the ledState of the variable:
  digitalWrite(led,  ledState);
  }
}
```

Remarque

Dans le programme ci-dessus, nous n'avons pas utilisé **`delay()`** pour faire clignoter la LED, car ESP32 met votre programme en pause pendant **`delay()`**. Si la prochaine demande OTA est générée alors qu'Arduino attend le délai d'expiration(), **votre programme manquera cette demande**.

Pour réaliser le delai nous avons utilisé le **timer** : **`currentMillis = millis()`**

Une fois que vous avez copié le croquis ci-dessus sur votre IDE Arduino, accédez à l'option Tools-> Port et vous devriez obtenir la sortie suivante: **`esp32-xxxxxx at esp_ip_address`**

Si vous ne le trouvez pas, vous devrez peut-être redémarrer votre IDE.

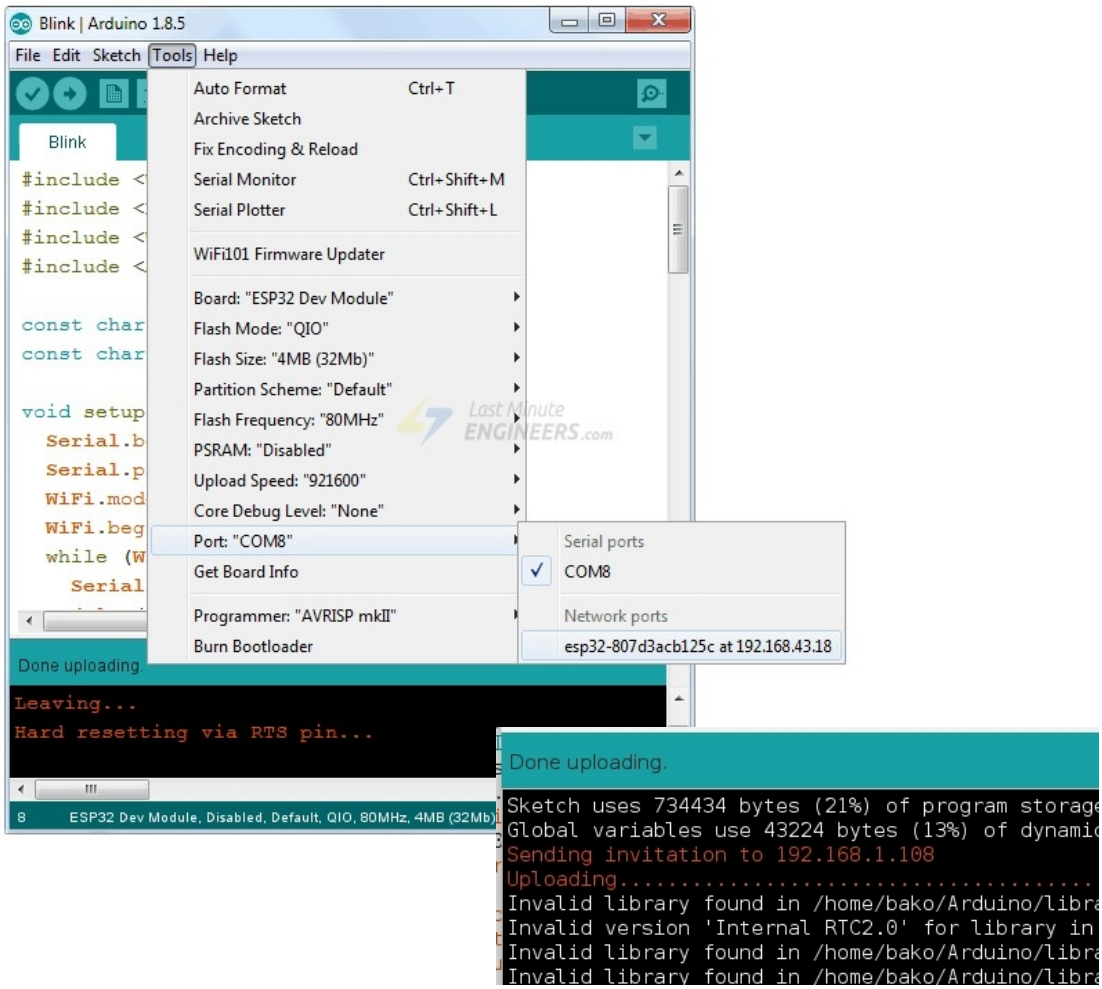

Figure 6.5 Sélection du port de transfert WiFi et de l'adresse IP

Sélectionnez le port et cliquez sur le bouton **`Download`**. Dans quelques secondes, le nouveau programme sera téléchargé. Et vous devriez voir la LED intégrée clignoter.

Pour vérifier le processus, vous pouvez modifier le programme en modifiant (par exemple) la valeur de:

```
int long const = 5000;
```

et téléchargez-le à nouveau via WiFi pour voir le résultat.

A faire :

1. Testez les programmes ci-dessus.
2. Modifiez le code en ajoutant un affichage sur l'écran OLED
3. Modifiez le code en ajoutant une lecture du capteur et un affichage sur l'écran OLED.

6.3 OTA sur carte ESP32 avec serveur WEB

Comme dans la solution précédente la première étape consiste à télécharger par USB le code contenant la routine OTA. Il s'agit d'une étape obligatoire pour pouvoir effectuer de futures mises à jour/téléchargements via WiFi.

Le nouveau programme OTA crée un **serveur Web** en mode **STA**, accessible via un navigateur Web. Une fois que vous êtes connecté au serveur Web, vous pouvez télécharger de nouveaux programmes avec la routine OTA.
Vous pouvez maintenant télécharger de nouveaux programmes sur l'ESP32 en générant et en téléchargeant le fichier **`.bin`** compilé dans l'environnement Arduino, via un **serveur Web**.

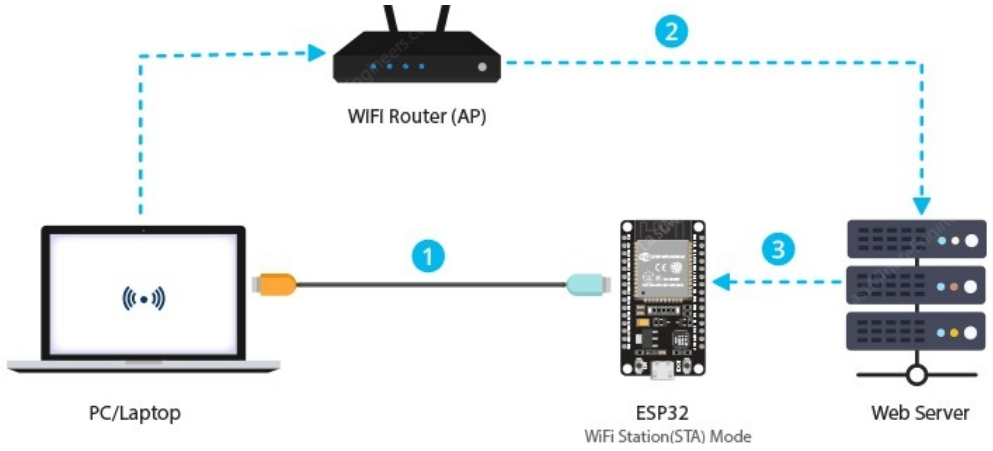

Figure 6.6 Transfert du code (fichier) binaire par le serveur WEB sur la carte ESP32

The ESP32 add-on for the Arduino IDE comes with an OTA library and an **`OTAWebUpdater`** example. You can access it via **`File> Examples> ArduinoOTA> OTAWebUpdater`** or via **`github.`**

Le toolkit d'ESP32 pour l'IDE Arduino est livré avec une bibliothèque OTA et un exemple **`OTAWebUpdater`**. Vous pouvez y accéder via **`File> Examples> ArduinoOTA> OTAWebUpdater`** ou via **`github`**.

Pour commencer, connectez votre ESP32 sur votre ordinateur et téléchargez le code ci-dessous.
Comme d'habitude vous devrait proposer les identificateurs WiFi de votre point d'accès. afin qu'ESP32 puisse établir une connexion avec le réseau existant.

6.3.1 Le programme de départ avec Webserver

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Update.h>

const char* host = "esp32";
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";

WebServer server(80);

// Login page

const char* loginIndex =
 "<form name='loginForm'>"
    "<table width='20%' bgcolor='A09F9F' align='center'>"
        "<tr>"
            "<td colspan=2>"
```

```
                "<center><font size=4><b>ESP32 Login Page</b></font></center>"
                "<br>"
            "</td>"
            "<br>"
            "<br>"
        "</tr>"
        "<td>Username:</td>"
        "<td><input type='text' size=25 name='userid'><br></td>"
        "</tr>"
        "<br>"
        "<br>"
        "<tr>"
            "<td>Password:</td>"
            "<td><input type='Password' size=25 name='pwd'><br></td>"
            "<br>"
            "<br>"
        "</tr>"
        "<tr>"
            "<td><input type='submit' onclick='check(this.form)'
value='Login'></td>"
        "</tr>"
    "</table>"
"</form>"
"<script>"
    "function check(form)"
    "{"
    "if(form.userid.value=='admin' && form.pwd.value=='admin')"
    "{"
    "window.open('/serverIndex')"
    "}"
    "else"
    "{"
    " alert('Error Password or Username')/*displays error message*/"
    "}"
    "}"
"</script>";

//Server Index Page

const char* serverIndex =
"<script
src='https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></
script>"
"<form method='POST' action='#' enctype='multipart/form-data' id='upload_form'>"
   "<input type='file' name='update'>"
        "<input type='submit' value='Update'>"
    "</form>"
 "<div id='prg'>progress: 0%</div>"
 "<script>"
  "$('form').submit(function(e){"
  "e.preventDefault();"
  "var form = $('#upload_form')[0];"
  "var data = new FormData(form);"
  " $.ajax({"
  "url: '/update',"
  "type: 'POST',"
  "data: data,"
  "contentType: false,"
  "processData:false,"
  "xhr: function() {"
  "var xhr = new window.XMLHttpRequest();"
```

```
  "xhr.upload.addEventListener('progress', function(evt) {"
  "if (evt.lengthComputable) {"
  "var per = evt.loaded / evt.total;"
  "$('#prg').html('progress: ' + Math.round(per*100) + '%');"
  "}"
  "}, false);"
  "return xhr;"
  "},"
  "success:function(d, s) {"
  "console.log('success!')"
 "},"
 "error: function (a, b, c) {"
 "}"
 "});"
 "});"
 "</script>";

void setup(void) {
  Serial.begin(9600);
  // Connect to WiFi network
  WiFi.begin(ssid, password);
  Serial.println("");
  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  /*use mdns for host name resolution*/
  if (!MDNS.begin(host)) { //http://esp32.local
    Serial.println("Error setting up MDNS responder!");
    while (1) {
      delay(1000);
    }
  }
  Serial.println("mDNS responder started");
  /*return index page which is stored in serverIndex */
  server.on("/", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", loginIndex);
  });
  server.on("/serverIndex", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", serverIndex);
  });
  /*handling uploading firmware file */
  server.on("/update", HTTP_POST, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
    ESP.restart();
  }, []() {
    HTTPUpload& upload = server.upload();
    if (upload.status == UPLOAD_FILE_START) {
      Serial.printf("Update: %s\n", upload.filename.c_str());
      if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available size
        Update.printError(Serial);
      }
```

```
    } else if (upload.status == UPLOAD_FILE_WRITE) {
      /* flashing firmware to ESP*/
      if (Update.write(upload.buf, upload.currentSize) != upload.currentSize) {
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_END) {
      if (Update.end(true)) { //true to set the size to the current progress
        Serial.printf("Update Success: %u\nRebooting...\n", upload.totalSize);
      } else {
        Update.printError(Serial);
      }
    }
  });
  server.begin();
}

void loop(void) {
  server.handleClient();
  delay(1);
}
```

6.3.2 Accéder au serveur Web

Le programme **`OTAWebUpdater`** crée un serveur Web en mode **STA** accessible via un navigateur Web et qui permet de télécharger de nouveaux programmes sur votre ESP32 via WiFi.
Pour accéder au serveur Web, ouvrez le moniteur série à une vitesse de transmission de 9600 bauds. Si tout va bien, l'**adresse IP dynamique** obtenue de votre routeur (carte) sera affichée.

Ensuite, chargez un navigateur et pointez-le vers l'adresse IP indiquée sur le moniteur série. L'ESP32 doit servir de page Web demandant les informations de connexion (par défaut : **`admin`** et **`admin`**).

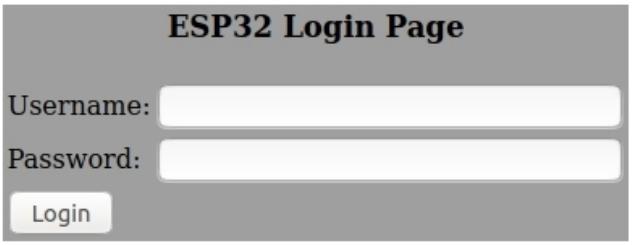

Figure 6.7 Page initiale du serveur WEB sur la carte ESP32

Si vous souhaitez modifier l'ID utilisateur et le mot de passe, modifiez le code ci-dessous dans votre programme.

```
"if (form.userid.value == 'admin' && form.pwd.value == 'admin')"
```

Une fois connecté au serveur, vous serez redirigé vers la page **`/serverIndex`**.

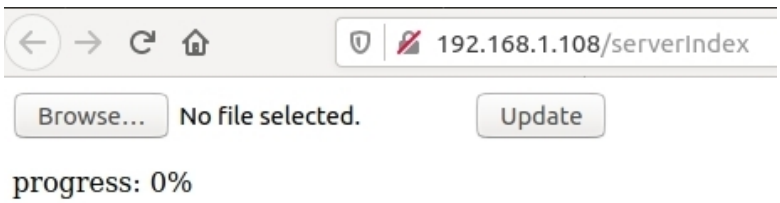

Figure 6.8 Page de recherche et de chargement d'un fichier **`.bin`**

Cette page vous permet de télécharger de nouveaux programmes sur votre ESP32 via WiFi. Ce nouveau programme en cours de téléchargement doit être au format **binaire** **`.bin`** .

6.3.3 Téléchargez un nouveau programme via WiFi (.bin)

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Update.h>

const char* host = "esp32";
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";

//variabls for blinking an LED with Millis
const int led = 25; // ESP32 Pin to which onboard LED is connected
unsigned long previousMillis = 0;  // will store last time LED was updated
const long interval = 1000;  // interval at which to blink (milliseconds)
int ledState = LOW;  // ledState used to set the LED

WebServer server(80);
/* Style */
String style =
"<style>#file-input,input{width:100%;height:44px;border-radius:4px;margin:10px
auto;font-size:15px}"
"input{background:#f1f1f1;border:0;padding:0 15px}body{background:#3498db;font-
family:sans-serif;font-size:14px;color:#777}"
"#file-input{padding:0;border:1px solid #ddd;line-height:44px;text-
align:left;display:block;cursor:pointer}"
"#bar,#prgbar{background-color:#f1f1f1;border-radius:10px}#bar{background-
color:#3498db;width:0%;height:10px}"
"form{background:#fff;max-width:258px;margin:75px auto;padding:30px;border-
radius:5px;text-align:center}"
".btn{background:#3498db;color:#fff;cursor:pointer}</style>";
/* Login page */
String loginIndex =
"<form name=loginForm>"
"<h1>ESP32 Login</h1>"
"<input name=userid placeholder='User ID'> "
"<input name=pwd placeholder=Password type=Password> "
"<input type=submit onclick=check(this.form) class=btn value=Login></form>"
"<script>"
"function check(form) {"
"if(form.userid.value=='admin' && form.pwd.value=='admin')"
"{window.open('/serverIndex')}"
"else"
"{alert('Error Password or Username')}"
"}"
"</script>" + style;

String serverIndex =
"<script
src='https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></
script>"
"<form method='POST' action='#' enctype='multipart/form-data' id='upload_form'>"
"<input type='file' name='update' id='file' onchange='sub(this)'
style=display:none>"
"<label id='file-input' for='file'>   Choose file...</label>"
"<input type='submit' class=btn value='Update'>"
"<br><br>"
"<div id='prg'></div>"
"<br><div id='prgbar'><div id='bar'></div></div><br></form>"
```

```
"<script>"
"function sub(obj){"
"var fileName = obj.value.split('\\\\');"
"document.getElementById('file-input').innerHTML = '   '+
fileName[fileName.length-1];"
"};"
"$('form').submit(function(e){"
"e.preventDefault();"
"var form = $('#upload_form')[0];"
"var data = new FormData(form);"
"$.ajax({"
"url: '/update',"
"type: 'POST',"
"data: data,"
"contentType: false,"
"processData:false,"
"xhr: function() {"
"var xhr = new window.XMLHttpRequest();"
"xhr.upload.addEventListener('progress', function(evt) {"
"if (evt.lengthComputable) {"
"var per = evt.loaded / evt.total;"
"$('#prg').html('progress: ' + Math.round(per*100) + '%');"
"$('#bar').css('width',Math.round(per*100) + '%');"
"}"
"}, false);"
"return xhr;"
"},"
"success:function(d, s) {"
"console.log('success!') "
"},"
"error: function (a, b, c) {"
"}"
"});"
"});"
"</script>" + style;

void setup(void) {
  Serial.begin(9600);pinMode(led,OUTPUT);
  WiFi.begin(ssid, password);Serial.println("");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
  }
  Serial.println("");Serial.print("Connected to ");
  Serial.println(ssid);Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  /*use mdns for host name resolution*/
  if (!MDNS.begin(host)) { //http://esp32.local
    Serial.println("Error setting up MDNS responder!");
    while (1) {  delay(1000);}
  }
  Serial.println("mDNS responder started");
  /*return index page which is stored in serverIndex */
  server.on("/", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", loginIndex);
  });
  server.on("/serverIndex", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", serverIndex);
  });
  /*handling uploading firmware file */
```

```
  server.on("/update", HTTP_POST, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
    ESP.restart();
  }, []() {
    HTTPUpload& upload = server.upload();
    if (upload.status == UPLOAD_FILE_START) {
      Serial.printf("Update: %s\n", upload.filename.c_str());
      if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available size
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_WRITE) {
      /* flashing firmware to ESP*/
      if (Update.write(upload.buf, upload.currentSize) != upload.currentSize) {
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_END) {
      if (Update.end(true)) { //true to set the size to the current progress
        Serial.printf("Update Success: %u\nRebooting...\n", upload.totalSize);
      } else {
        Update.printError(Serial);
      }
    }
  });
  server.begin();
}

void loop(void) {
  server.handleClient();
  delay(1);
 unsigned long currentMillis = millis();
 if (currentMillis - previousMillis >= interval) {
 previousMillis = currentMillis;
 ledState = not(ledState);
 digitalWrite(led,  ledState);
 }
}
```

6.3.4 Générez un fichier `.bin` dans l'IDE Arduino

Afin de télécharger un nouveau croquis sur l'ESP32, nous devons d'abord générer le fichier binaire `.bin` compilé de votre programme. Pour ce faire, sélectionnez **`Sketch> Export compiled Binary`**

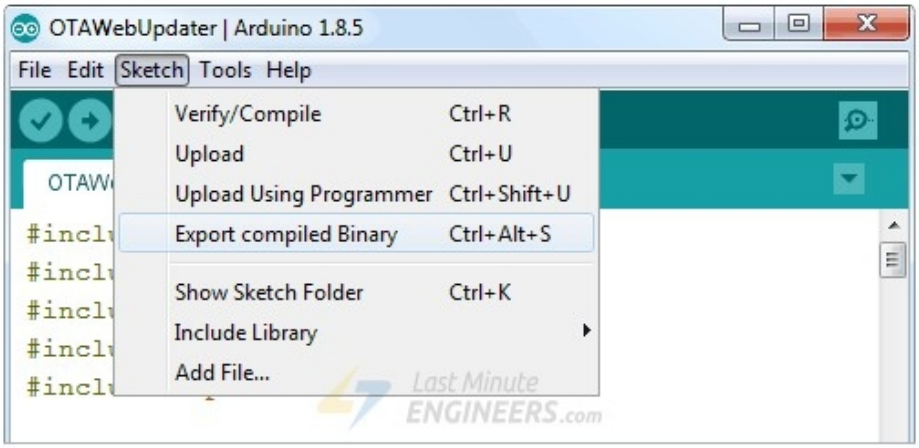

Figure 6.9 Génération du code binaire pour transfert sur la carte ESP32

6.3.5 Download a new sketch live on the ESP32

Once the **`.bin`** file is generated, you are now ready to upload the new sketch to the ESP32 over WiFi. Open the **`/serverIndex`** page in your browser. Click **`Choose File…`** Select the generated **`.bin`** file, then click **`Update`**.
Une fois le fichier **`.bin`** généré, vous êtes maintenant prêt à télécharger le nouveau code sur l'ESP32 via WiFi. Ouvrez la page **`/serverIndex`** dans votre navigateur. Cliquez sur Choisir un fichier… Sélectionnez le fichier **`.bin`** généré (dans le même répertoire que votre sketch **`.ino`**) , puis cliquez sur **Update**.

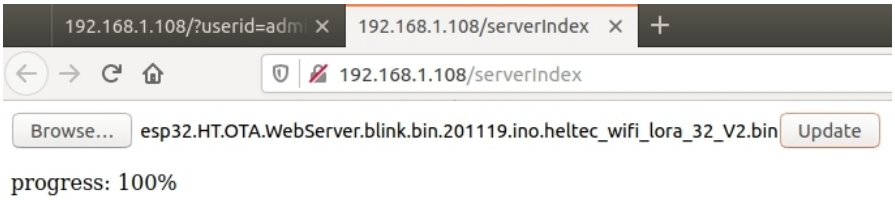

Figure 6.10 Transfert du code binaire sur la carte ESP32

In a few seconds the new sketch will be uploaded. And you should see the built-in LED blinking.

To do :

1. Test the above programs.
2. Modify the code by adding a display on the OLED screen
3. Change the code by adding a sensor reading and display on the OLED screen.

Dans quelques secondes, le nouveau programme sera téléchargé. Et vous devriez voir la LED intégrée clignoter.

A faire

1. **Testez** les programmes ci-dessus.
2. **Modifiez** le code en ajoutant un affichage sur l'écran OLED
3. **Modifiez** le code en ajoutant une lecture du capteur et un affichage sur l'écran OLED.

6.4 Implémentation d'OTA avec la bibliothèque `WebOTA`

La solution précédente nécessite l'insertion du **code HTML** de la page WEB créée pour le transfert du code `.bin` vers la carte ESP32. La bibliothèque **WebOTA** promet de simplifier cette préparation.
Vous aurez besoin d'une inclusion, bien sûr. Si cela ne vous dérange pas d'utiliser le port `8080` et le chemin par défaut `/webota`, appelez simplement `handle_webota`) depuis votre boucle principale (`loop()`).
Si vous souhaitez modifier les valeurs par défaut, vous devez ajouter un appel supplémentaire dans votre configuration. Vous devez également configurer quelques variables globales pour spécifier vos paramètres réseau.
Le seul inconvénient est que les déclarations avec un long délai (`delay()`) dans votre boucle peuvent empêcher certaines choses de fonctionner correctement et ne sont pas une bonne idée.
Si vous en avez, vous pouvez remplacer tous vos appels `delay()` par `webota_delay()`, ce qui empêchera le système d'ignorer les demandes de mise à jour.
Le code initial doit être chargé par l'environnement Arduino. Pour effectuer une mise à jour, accédez simplement à la carte ESP32 avec un navigateur Web et utilisez le numéro de port et le chemin corrects. De là, vous pouvez télécharger une nouvelle image binaire créée dans l'IDE Arduino.

Remarque :

Le code à charger n'est pas authentifié. Cela signifie que n'importe qui peut télécharger du code sur votre ESP. Cela peut être acceptable sur un réseau privé, mais sur Internet, c'est certainement problématique.

6.4.1 Code initial

```
#include <WebOTA.h>
#define LED_PIN 25
const char* host     = "ESP-OTA"; // Used for MDNS resolution
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";

void setup() {
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  init_wifi(ssid, password, host);
  // Defaults to 8080 and "/webota" , webota.init(80, "/update");
}

void loop() {
  int md = 5000;
  digitalWrite(LED_PIN, HIGH);
  webota.delay(md);
  digitalWrite(LED_PIN, LOW);
  webota.delay(md);
  webota.handle();
}
```

6.4.2 Chargement initial et final

Dans la phase de chargement initial, nous obtenons l'adresse **IP** du serveur WEB sur la carte.

```
WebOTA url   : http://ESP-OTA.local:8080/webota

Firmware update initiated: esp32.HT.WebOTA.init.201119.ino.heltec_wifi_lora_32_V2.bin
50k 100k 150k 200k 250k 300k 350k 400k 450k 500k 550k 600k 650k 700k
Firmware update successful: 759856 bytes
Rebooting...
        ϓϓϓ1ϓX j$Iϓ\ ∏  ϓ
Connecting to Wifi.
Connected to 'Livebox-08B0'

IP address   : 192.168.1.108
MAC address  : 24:6F:28:96:EF:48
mDNS started : ESP-OTA.local
WebOTA url   : http://ESP-OTA.local:8080/webota
```

Figure 6.11 Affichage des paramètres: IP et URL du serveur WEB

L'affichage se fait jusqu'à la ligne avec l'url **WebOTA**. Le serveur est accessible avec ;
`192.168.1.108:8080/webota`

Il vous demande de fournir l'emplacement de votre code Arduino compilé en **`.bin`**.

`esp32.HT.WebOTA.init.201119.ino.heltec_wifi_lora32_V2.bin`

Figure 6.12 La page initiale de chargement du code binaire depuis le serveur WEB

A faire :

1. Testez l'exemple ci-dessus.
2. Modifiez le code en ajoutant un affichage sur l'écran OLED
3. Modifiez le code en ajoutant une lecture du capteur et un affichage sur l'écran OLED.