# Confimode. h

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266HTTPUpdateServer.h>
#include <DNSServer.h>

ESP8266WebServer server(80);
ESP8266HTTPUpdateServer httpUpdater;
DNSServer dnsServer;
const byte DNS_PORT = 53;


#ifdef BLYNK_USE_SPIFFS
  #include <FS.h>
#else
  const char* config_form = R"html(
<!DOCTYPE HTML>
<html>
<head>
  <title>WiFi setup</title>
  <style>
  body {
    background-color: #fcfcfc;
    box-sizing: border-box;
  }
  body, input {
    font-family: Roboto, sans-serif;
    font-weight: 400;
    font-size: 16px;
  }
  .centered {
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);

    padding: 20px;
    background-color: #ccc;
    border-radius: 4px;
  }
  td { padding:0 0 0 5px; }
  label { white-space:nowrap; }
  input { width: 20em; }
  input[name="port"] { width: 5em; }
  input[type="submit"], img { margin: auto; display: block; width: 30%; }
  </style>
</head>
<body>
<div class="centered">
  <form method="get" action="config">
    <table>
    <tr><td><label for="ssid">WiFi SSID:</label></td> <td><input type="text" name="ssid" length=64 required="required"></td></tr>
    <tr><td><label for="pass">Password:</label></td> <td><input type="text" name="pass" length=64></td></tr>
    <tr><td><label for="blynk">Auth token:</label></td><td><input type="text" name="blynk" placeholder="a0b1c2d..." pattern="[-_a-zA-Z0-9]
{32}" maxlength="32" required="required"></td></tr>
    <tr><td><label for="host">Host:</label></td> <td><input type="text" name="host" value="blynk.cloud" length=64></td></tr>
    <tr><td><label for="port_ssl">Port:</label></td> <td><input type="number" name="port_ssl" value="443" min="1" max="65535"></td>
</tr>
    </table><br/>
    <input type="submit" value="Apply">
  </form>
</div>
</body>
</html>
)html";
#endif

void restartMCU() {
  ESP.restart();
  delay(10000);
  ESP.reset();
  while(1) {};
}
```

```
void getWiFiName(char* buff, size_t len, bool withPrefix = true) {
  byte mac[6] = { 0, };
  WiFi.macAddress(mac);

  uint32_t unique = 0;
  for (int i=0; i<4; i++) {
    unique = BlynkCRC32(&mac, sizeof(mac), unique);
  }
  unique &= 0xFFFFF;

  String devName = String(BLYNK_DEVICE_NAME).substring(0, 31-6-6);

  if (withPrefix) {
    snprintf(buff, len, "Blynk %s-%05X", devName.c_str(), unique);
  } else {
    snprintf(buff, len, "%s-%05X", devName.c_str(), unique);
  }
}

void enterConfigMode()
{
  char ssidBuff[64];
  getWiFiName(ssidBuff, sizeof(ssidBuff));

  WiFi.mode(WIFI_OFF);
  delay(100);
  WiFi.mode(WIFI_AP_STA);
  WiFi.softAPConfig(WIFI_AP_IP, WIFI_AP_IP, WIFI_AP_Subnet);
  WiFi.softAP(ssidBuff);
  delay(500);

  IPAddress myIP = WiFi.softAPIP();
  DEBUG_PRINT(String("AP SSID: ") + ssidBuff);
  DEBUG_PRINT(String("AP IP: ") + myIP[0] + "." + myIP[1] + "." + myIP[2] + "." + myIP[3]);

  if (myIP == (uint32_t)0)
  {
    config_set_last_error(BLYNK_PROV_ERR_INTERNAL);
    BlynkState::set(MODE_ERROR);
    return;
  }

  // Set up DNS Server
  dnsServer.setTTL(300); // Time-to-live 300s
  dnsServer.setErrorReplyCode(DNSReplyCode::ServerFailure); // Return code for non-accessible domains
#ifdef WIFI_CAPTIVE_PORTAL_ENABLE
  dnsServer.start(DNS_PORT, "*", WiFi.softAPIP()); // Point all to our IP
  server.onNotFound(handleRoot);
#else
  dnsServer.start(DNS_PORT, CONFIG_AP_URL, WiFi.softAPIP());
  DEBUG_PRINT(String("AP URL: ") + CONFIG_AP_URL);
#endif

  httpUpdater.setup(&server, "/update");

#ifndef BLYNK_USE_SPIFFS
  server.on("/", []() {
    server.send(200, "text/html", config_form);
  });
#endif
  server.on("/config", []() {
    DEBUG_PRINT("Applying configuration...");
    String ssid = server.arg("ssid");
    String ssidManual = server.arg("ssidManual");
    String pass = server.arg("pass");
    if (ssidManual != "") {
      ssid = ssidManual;
    }
    String token = server.arg("blynk");
    String host = server.arg("host");
    String port = server.arg("port_ssl");

    String ip = server.arg("ip");
    String mask = server.arg("mask");
    String gw = server.arg("gw");
    String dns = server.arg("dns");
    String dns2 = server.arg("dns2");

    bool save = server.arg("save").toInt();

    String content;
```

```
      DEBUG_PRINT(String("WiFi SSID: ") + ssid + " Pass: " + pass);
      DEBUG_PRINT(String("Blynk cloud: ") + token + " @ " + host + ":" + port);

      if (token.length() == 32 && ssid.length() > 0) {
        configStore.setFlag(CONFIG_FLAG_VALID, false);
        CopyString(ssid, configStore.wifiSSID);
        CopyString(pass, configStore.wifiPass);
        CopyString(token, configStore.cloudToken);
        if (host.length()) {
          CopyString(host, configStore.cloudHost);
        }
        if (port.length()) {
          configStore.cloudPort = port.toInt();
        }

        IPAddress addr;

        if (ip.length() && addr.fromString(ip)) {
          configStore.staticIP = addr;
          configStore.setFlag(CONFIG_FLAG_STATIC_IP, true);
        } else {
          configStore.setFlag(CONFIG_FLAG_STATIC_IP, false);
        }
        if (mask.length() && addr.fromString(mask)) {
          configStore.staticMask = addr;
        }
        if (gw.length() && addr.fromString(gw)) {
          configStore.staticGW = addr;
        }
        if (dns.length() && addr.fromString(dns)) {
          configStore.staticDNS = addr;
        }
        if (dns2.length() && addr.fromString(dns2)) {
          configStore.staticDNS2 = addr;
        }

        if (save) {
          configStore.setFlag(CONFIG_FLAG_VALID, true);
          config_save();

          content = R"json({"status":"ok","msg":"Configuration saved"})json";
        } else {
          content = R"json({"status":"ok","msg":"Trying to connect..."})json";
        }
        server.send(200, "application/json", content);

        BlynkState::set(MODE_SWITCH_TO_STA);
      } else {
        DEBUG_PRINT("Configuration invalid");
        content = R"json({"status":"error","msg":"Configuration invalid"})json";
        server.send(500, "application/json", content);
      }
    });
    server.on("/board_info.json", []() {
      DEBUG_PRINT("Sending board info...");
      const char* tmpl = BLYNK_TEMPLATE_ID;
      char ssidBuff[64];
      getWiFiName(ssidBuff, sizeof(ssidBuff));
      char buff[512];
      snprintf(buff, sizeof(buff),

  R"json({"board":"%s","tmpl_id":"%s","fw_type":"%s","fw_ver":"%s","ssid":"%s","bssid":"%s","mac":"%s","last_error":%d,"wifi_scan":true,"static_ip":true})json
        BLYNK_DEVICE_NAME,
        tmpl ? tmpl : "Unknown",
        BLYNK_FIRMWARE_TYPE,
        BLYNK_FIRMWARE_VERSION,
        ssidBuff,
        WiFi.softAPmacAddress().c_str(),
        WiFi.macAddress().c_str(),
        configStore.last_error
      );
      server.send(200, "application/json", buff);
    });
    server.on("/wifi_scan.json", []() {
      DEBUG_PRINT("Scanning networks...");
      int wifi_nets = WiFi.scanNetworks(true, true);
      const uint32_t t = millis();
      while (wifi_nets < 0 &&
             millis() - t < 20000)
      {
        delay(20);
        wifi_nets = WiFi.scanComplete();
      }
      DEBUG_PRINT(String("Found networks: ") + wifi_nets);

      if (wifi_nets > 0) {
        // Sort networks
        int indices[wifi_nets];
        for (int i = 0; i < wifi_nets; i++) {
          indices[i] = i;
        }
        for (int i = 0; i < wifi_nets; i++) {
          for (int j = i + 1; j < wifi_nets; j++) {
            if (WiFi.RSSI(indices[j]) > WiFi.RSSI(indices[i])) {
              std::swap(indices[i], indices[j]);
            }
          }
        }

        wifi_nets = BlynkMin(15, wifi_nets); // Show top 15 networks

        // TODO: skip empty names
        server.setContentLength(CONTENT_LENGTH_UNKNOWN);
        server.send(200, "application/json", "[\n");


        char buff[256];
        for (int i = 0; i < wifi_nets; i++){
          int id = indices[i];

          const char* sec;
          switch (WiFi.encryptionType(id)) {
          case ENC_TYPE_WEP: sec = "WEP"; break;
          case ENC_TYPE_TKIP: sec = "WPA/PSK"; break;
          case ENC_TYPE_CCMP: sec = "WPA2/PSK"; break;
          case ENC_TYPE_AUTO: sec = "WPA/WPA2/PSK"; break;
          case ENC_TYPE_NONE: sec = "OPEN"; break;
          default: sec = "unknown"; break;
          }

          snprintf(buff, sizeof(buff),
            R"json( {"ssid":"%s","bssid":"%s","rssi":%i,"sec":"%s","ch":%i,"hidden":%d})json",
            WiFi.SSID(id).c_str(),
            WiFi.BSSIDstr(id).c_str(),
            WiFi.RSSI(id),
            sec,
            WiFi.channel(id),
            WiFi.isHidden(id)
          );

          server.sendContent(buff);
          if (i != wifi_nets-1) server.sendContent(",\n");
        }
        server.sendContent("\n]");
      } else {
        server.send(200, "application/json", "[]");
      }
    });
    server.on("/reset", []() {
      BlynkState::set(MODE_RESET_CONFIG);
      server.send(200, "application/json", R"json({"status":"ok","msg":"Configuration reset"})json");
    });
    server.on("/reboot", []() {
      restartMCU();
    });

  #ifdef BLYNK_USE_SPIFFS
    if (SPIFFS.begin()) {
      server.serveStatic("/img", SPIFFS, "/img");
      server.serveStatic("/", SPIFFS, "/index.html");
    } else {
      DEBUG_PRINT("Webpage: No SPIFFS");
    }
  #endif

    server.begin();

    while (BlynkState::is(MODE_WAIT_CONFIG) || BlynkState::is(MODE_CONFIGURING)) {
      delay(10);
      dnsServer.processNextRequest();
      server.handleClient();
      app_loop();
      if (BlynkState::is(MODE_WAIT_CONFIG) && WiFi.softAPgetStationNum() > 0) {
        BlynkState::set(MODE_CONFIGURING);
      } else if (BlynkState::is(MODE_CONFIGURING) && WiFi.softAPgetStationNum() == 0) {
        BlynkState::set(MODE_WAIT_CONFIG);
      }
    }

    server.stop();

  #ifdef BLYNK_USE_SPIFFS
    SPIFFS.end();
  #endif
  }

  void enterConnectNet() {
    BlynkState::set(MODE_CONNECTING_NET);
    DEBUG_PRINT(String("Connecting to WiFi: ") + configStore.wifiSSID);

    WiFi.mode(WIFI_STA);

    char ssidBuff[64];
    getWiFiName(ssidBuff, sizeof(ssidBuff));
    String hostname(ssidBuff);
    hostname.replace(" ", "-");
    WiFi.hostname(hostname.c_str());

    if (configStore.getFlag(CONFIG_FLAG_STATIC_IP)) {
      if (!WiFi.config(configStore.staticIP,
                      configStore.staticGW,
                      configStore.staticMask,
                      configStore.staticDNS,
                      configStore.staticDNS2)
      ) {
        DEBUG_PRINT("Failed to configure Static IP");
        config_set_last_error(BLYNK_PROV_ERR_CONFIG);
        BlynkState::set(MODE_ERROR);
        return;
      }
    }

    if (!WiFi.begin(configStore.wifiSSID, configStore.wifiPass)) {
      config_set_last_error(BLYNK_PROV_ERR_CONFIG);
      BlynkState::set(MODE_ERROR);
      return;
    }

    unsigned long timeoutMs = millis() + WIFI_NET_CONNECT_TIMEOUT;
    while ((timeoutMs > millis()) && (WiFi.status() != WL_CONNECTED))
    {
      delay(10);
      app_loop();

      if (!BlynkState::is(MODE_CONNECTING_NET)) {
        WiFi.disconnect();
        return;
      }
    }

    if (WiFi.status() == WL_CONNECTED) {
      IPAddress localip = WiFi.localIP();
      if (configStore.getFlag(CONFIG_FLAG_STATIC_IP)) {
        BLYNK_LOG_IP("Using Static IP: ", localip);
      } else {
        BLYNK_LOG_IP("Using Dynamic IP: ", localip);
      }

      BlynkState::set(MODE_CONNECTING_CLOUD);
    } else {
      config_set_last_error(BLYNK_PROV_ERR_NETWORK);
      BlynkState::set(MODE_ERROR);
    }
  }

  void enterConnectCloud() {
    BlynkState::set(MODE_CONNECTING_CLOUD);

    Blynk.config(configStore.cloudToken, configStore.cloudHost, configStore.cloudPort);
    Blynk.connect(0);

    unsigned long timeoutMs = millis() + WIFI_CLOUD_CONNECT_TIMEOUT;
    while ((timeoutMs > millis()) &&
          (!Blynk.isTokenInvalid()) &&
          (Blynk.connected() == false))
    {
      delay(10);
      Blynk.run();
      app_loop();
      if (!BlynkState::is(MODE_CONNECTING_CLOUD)) {
        Blynk.disconnect();
        return;
      }
    }

    if (millis() > timeoutMs) {
      DEBUG_PRINT("Timeout");
    }

    if (Blynk.isTokenInvalid()) {
      config_set_last_error(BLYNK_PROV_ERR_TOKEN);
      BlynkState::set(MODE_WAIT_CONFIG);
    } else if (Blynk.connected()) {
      BlynkState::set(MODE_RUNNING);

      if (!configStore.getFlag(CONFIG_FLAG_VALID)) {
        configStore.last_error = BLYNK_PROV_ERR_NONE;
        configStore.setFlag(CONFIG_FLAG_VALID, true);
        config_save();
      }
    } else {
      config_set_last_error(BLYNK_PROV_ERR_CLOUD);
      BlynkState::set(MODE_ERROR);
    }
  }

  void enterSwitchToSTA() {
    BlynkState::set(MODE_SWITCH_TO_STA);

    DEBUG_PRINT("Switching to STA...");

    delay(1000);
    WiFi.mode(WIFI_OFF);
    delay(100);
    WiFi.mode(WIFI_STA);

    BlynkState::set(MODE_CONNECTING_NET);
  }

  void enterError() {
    BlynkState::set(MODE_ERROR);

    unsigned long timeoutMs = millis() + 10000;
    while (timeoutMs > millis() || g_buttonPressed)
    {
      delay(10);
      app_loop();
      if (!BlynkState::is(MODE_ERROR)) {
        return;
      }
    }
    DEBUG_PRINT("Restarting after error.");
    delay(10);

    restartMCU();
  }
```