**Листинг 3 Кейса**

**Файл mbed\_app.json**

{

    "config": {

        "wifi-ssid": {

            "help": "WiFi SSID",

            "value": "\"Smart\_box-F816EC\""

        },

        "wifi-password": {

            "help": "WiFi Password",

            "value": "\"20122009\""

        },

        "tx": {

            "help": "TX pin for serial connection",

            "value": "D8"

        },

        "rx": {

            "help": "RX pin for serial connection",

            "value": "D2"

        },

        "socket-bufsize": {

            "help": "Max socket data heap usage",

            "value": "1024"

        }

    },

    "target\_overrides": {

        "\*": {

            "target.network-default-interface-type": "WIFI",

            "esp8266.tx": "D8",

            "esp8266.rx": "D2",

            "esp8266.socket-bufsize": "1024",

            "esp8266.debug": false,

            "platform.stdio-convert-newlines": true,

            "esp8266.provide-default" : true

        }

    }

}

**Файл mbed\_app\_esp8266.json**

{

    "config": {

        "wifi-ssid": {

            "help": "WiFi SSID",

            "value": "\"SSID\""

        },

        "wifi-password": {

            "help": "WiFi Password",

            "value": "\"PASSWORD\""

        }

    },

    "target\_overrides": {

        "\*": {

            "platform.stdio-convert-newlines": true,

            "esp8266.provide-default" : true

        }

    }

}

**Файл mbed\_app\_idw01m1.json**

{

    "config": {

        "wifi-ssid": {

            "help": "WiFi SSID",

            "value": "\"SSID\""

        },

        "wifi-password": {

            "help": "WiFi Password",

            "value": "\"PASSWORD\""

        }

    },

    "target\_overrides": {

        "\*": {

            "platform.stdio-convert-newlines": true,

            "idw0xx1.expansion-board": "IDW01M1",

            "idw0xx1.provide-default": true,

            "idw0xx1.tx": "PA\_9",

            "idw0xx1.rx": "PA\_10",

            "drivers.uart-serial-txbuf-size": 730,

            "drivers.uart-serial-rxbuf-size": 730

        }

    }

}

**Файл mbed\_app\_idw04a1.json**

{

    "config": {

    "wifi-shield": {

            "help": "Options are internal, WIFI\_ESP8266, WIFI\_IDW0XX1",

            "value": "WIFI\_IDW0XX1"

    },

        "wifi-ssid": {

            "help": "WiFi SSID",

            "value": "\"SSID\""

        },

        "wifi-password": {

            "help": "WiFi Password",

            "value": "\"PASSWORD\""

        }

    },

    "target\_overrides": {

        "\*": {

            "platform.stdio-convert-newlines": true,

            "idw0xx1.expansion-board": "IDW04A1",

            "idw0xx1.provide-default": true,

            "idw0xx1.tx": "PA\_9",

            "idw0xx1.rx": "PA\_10",

        "drivers.uart-serial-txbuf-size": 750,

        "drivers.uart-serial-rxbuf-size": 750

        }

    },

    "macros": ["IDW04A1\_WIFI\_HW\_BUG\_WA"]

}

**Файл main.cpp**

#include "mbed.h"

#include "MQTTmbed.h"

#include "MQTTClientMbedOs.h"

#include "Sht31.h"

WiFiInterface \*wifi;

int arrivedcount = 0;

const char \*sec2str(nsapi\_security\_t sec)

{

    switch (sec) {

        case NSAPI\_SECURITY\_NONE:

            return "None";

        case NSAPI\_SECURITY\_WEP:

            return "WEP";

        case NSAPI\_SECURITY\_WPA:

            return "WPA";

        case NSAPI\_SECURITY\_WPA2:

            return "WPA2";

        case NSAPI\_SECURITY\_WPA\_WPA2:

            return "WPA/WPA2";

        case NSAPI\_SECURITY\_UNKNOWN:

        default:

            return "Unknown";

    }

}

int scan\_demo(WiFiInterface \*wifi)

{

    WiFiAccessPoint \*ap;

    printf("Scan:\n");

    int count = wifi->scan(NULL,0);

    if (count <= 0) {

        printf("scan() failed with return value: %d\n", count);

        return 0;

    }

    /\* Limit number of network arbitrary to 15 \*/

    count = count < 15 ? count : 15;

    ap = new WiFiAccessPoint[count];

    count = wifi->scan(ap, count);

    if (count <= 0) {

        printf("scan() failed with return value: %d\n", count);

        return 0;

    }

    for (int i = 0; i < count; i++) {

        printf("Network: %s secured: %s BSSID: %hhX:%hhX:%hhX:%hhx:%hhx:%hhx RSSI: %hhd Ch: %hhd\n", ap[i].get\_ssid(),

               sec2str(ap[i].get\_security()), ap[i].get\_bssid()[0], ap[i].get\_bssid()[1], ap[i].get\_bssid()[2],

               ap[i].get\_bssid()[3], ap[i].get\_bssid()[4], ap[i].get\_bssid()[5], ap[i].get\_rssi(), ap[i].get\_channel());

    }

    printf("%d networks available.\n", count);

    delete[] ap;

    return count;

}

void http\_demo(NetworkInterface \*net)

{

    // Open a socket on the network interface, and create a TCP connection to mbed.org

    TCPSocket socket;

    socket.open(net);

    SocketAddress a;

    net->gethostbyname("ifconfig.io", &a);

    a.set\_port(80);

    socket.connect(a);

    // Send a simple http request

    char sbuffer[] = "GET / HTTP/1.1\r\nHost: ifconfig.io\r\n\r\n";

    int scount = socket.send(sbuffer, sizeof sbuffer);

    printf("sent %d [%.\*s]\n", scount, strstr(sbuffer, "\r\n") - sbuffer, sbuffer);

    // Recieve a simple http response and print out the response line

    char rbuffer[64];

    int rcount = socket.recv(rbuffer, sizeof rbuffer);

    printf("recv %d [%.\*s]\n", rcount, strstr(rbuffer, "\r\n") - rbuffer, rbuffer);

    // Close the socket to return its memory and bring down the network interface

    socket.close();

}

void messageArrived(MQTT::MessageData& md)

{

    MQTT::Message &message = md.message;

    printf("Message arrived: qos %d, retained %d, dup %d, packetid %d\r\n", message.qos, message.retained, message.dup, message.id);

    printf("Payload %.\*s\r\n", message.payloadlen, (char\*)message.payload);

    ++arrivedcount;

}

int main()

{

    printf("WiFi example\n");

#ifdef MBED\_MAJOR\_VERSION

    printf("Mbed OS version %d.%d.%d\n\n", MBED\_MAJOR\_VERSION, MBED\_MINOR\_VERSION, MBED\_PATCH\_VERSION);

#endif

    wifi = WiFiInterface::get\_default\_instance();

    if (!wifi) {

        printf("ERROR: No WiFiInterface found.\n");

        return -1;

    }

    int count = scan\_demo(wifi);

    if (count == 0) {

        printf("No WIFI APs found - can't continue further.\n");

        return -1;

    }

    printf("\nConnecting to %s...\n", MBED\_CONF\_APP\_WIFI\_SSID);

    int ret = wifi->connect(MBED\_CONF\_APP\_WIFI\_SSID, MBED\_CONF\_APP\_WIFI\_PASSWORD, NSAPI\_SECURITY\_WPA\_WPA2);

    if (ret != 0) {

        printf("\nConnection error: %d\n", ret);

        return -1;

    }

    printf("Success\n\n");

    printf("MAC: %s\n", wifi->get\_mac\_address());

    printf("IP: %s\n", wifi->get\_ip\_address());

    printf("Netmask: %s\n", wifi->get\_netmask());

    printf("Gateway: %s\n", wifi->get\_gateway());

    printf("RSSI: %d\n\n", wifi->get\_rssi());

    http\_demo(wifi);

    wifi->disconnect();

    printf("\nDone\n");

}