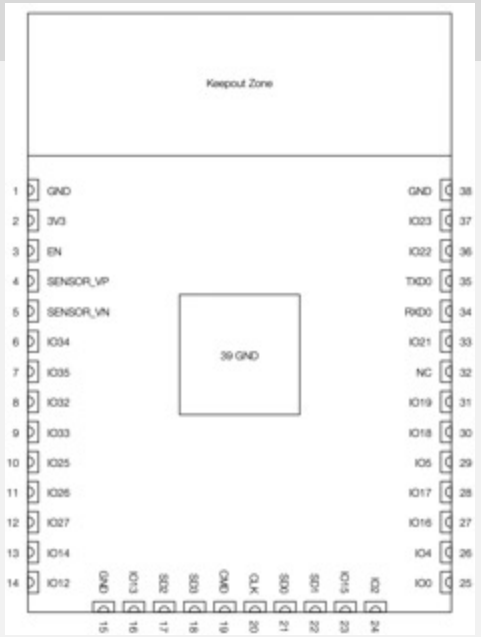
ESP 32 – Introduction

Objectifs

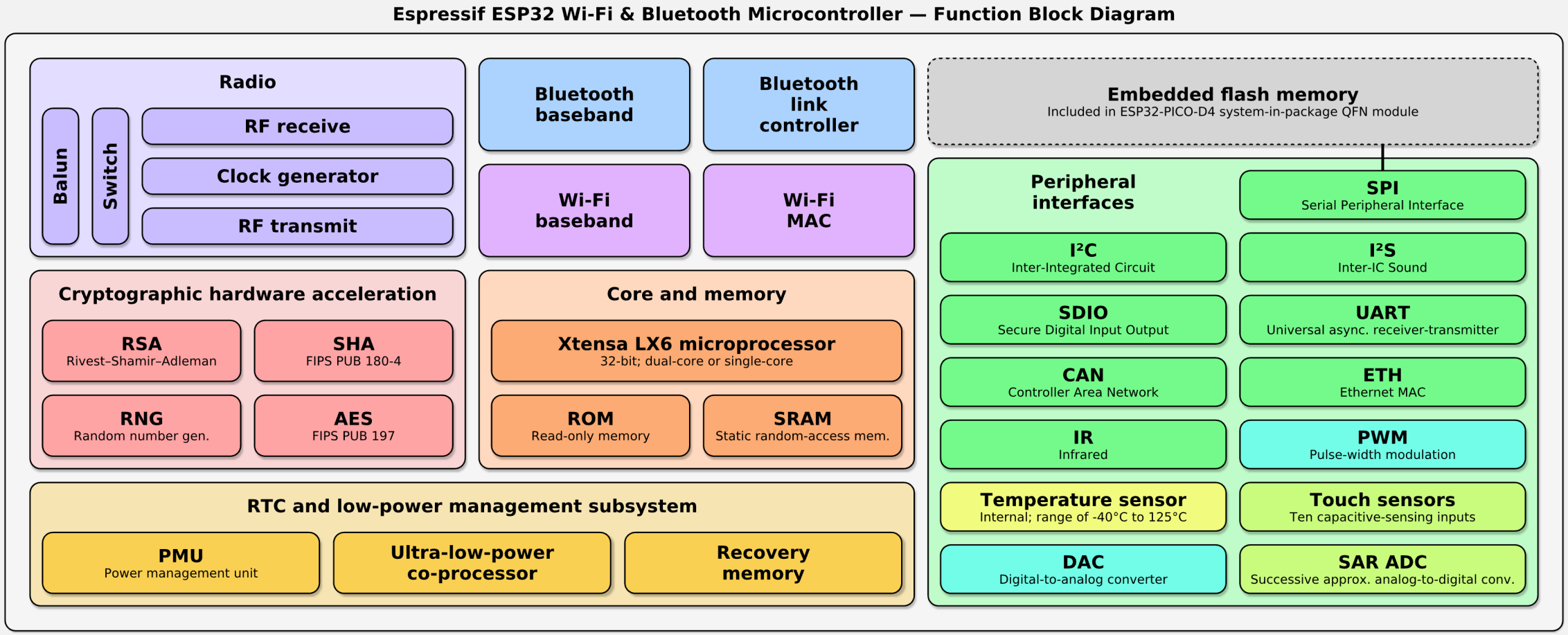
- Architecture du microcontrôleur ESP32
- Configurer PlatformIO pour utiliser le microcontrôleur ESP32
- Se connecter à un réseau WiFi
- Faire une requête sur le web
- Sérialiser / désérialiser des données au format JSON

ESP32-WROOM-32D



- Compagnie : Espressif
- Module générique avec Wi-Fi, BT, BLE
- Caractéristiques :
  - 2 cœurs LX6 32 bits
  - ROM (448 Ko) : démarrage et fonctions
  - SRAM (520 Ko) : données et instructions
  - SRAM in RTC FAST 8 Ko : accessible pour stocker des données, accessible par le processeur principal
  - SRAM in RTC SLOW 8 Ko : accessible pour stocker des données, accessible par le co-processeur durant le sommeil du module
  - 4 Mo de mémoire flash externe (GPIO 6 à 11)

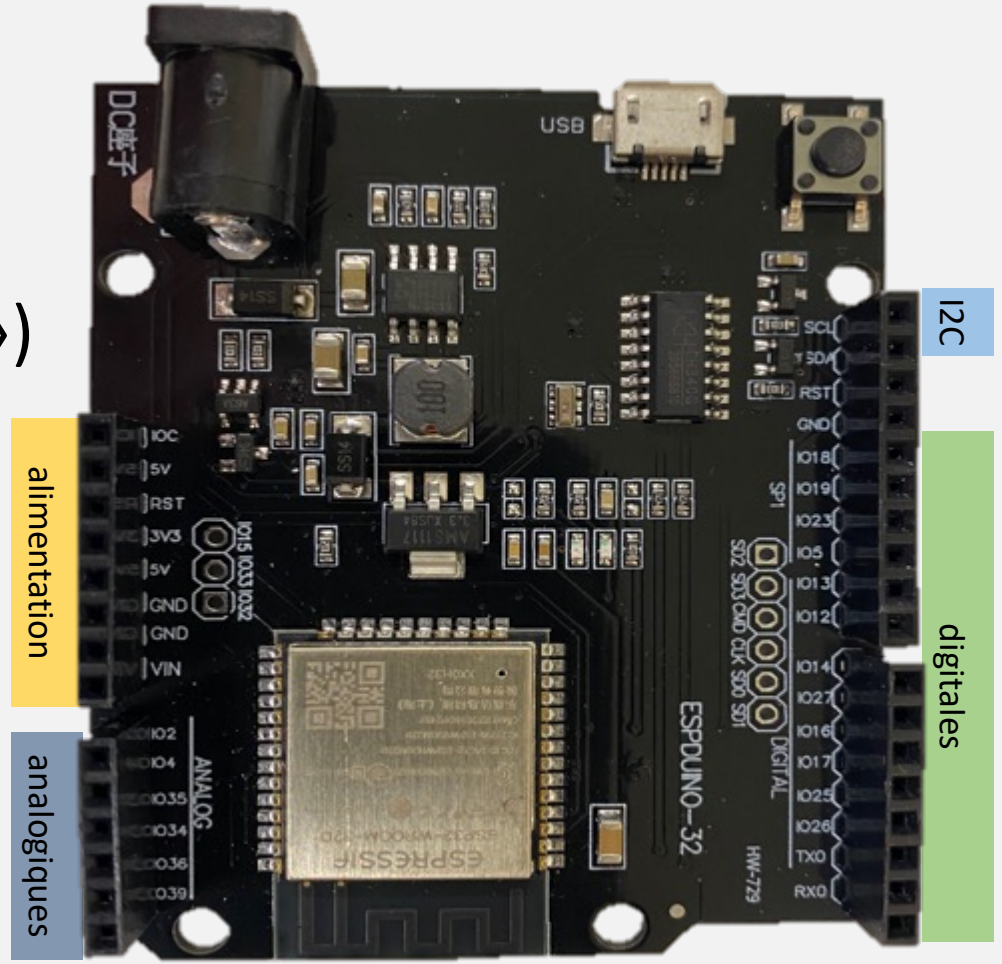
ESP32-WROOM-32D



https://commons.wikimedia.org/wiki/File:Espressif_ESP32_Chip_Function_Block_Diagram.svg

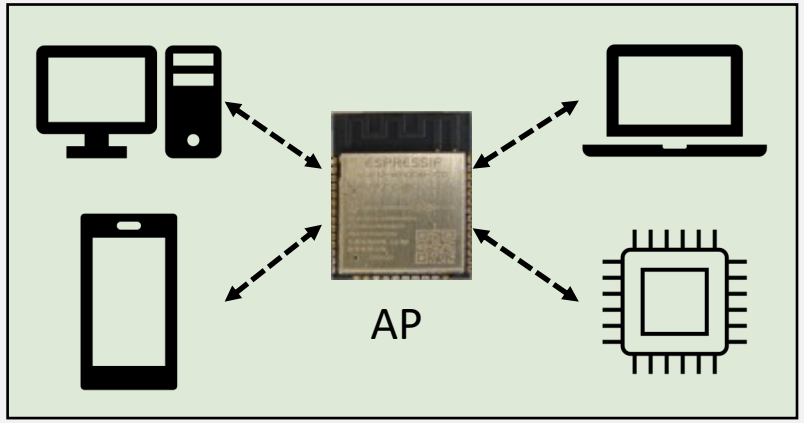
Modèle esp32doit-devkit-v1

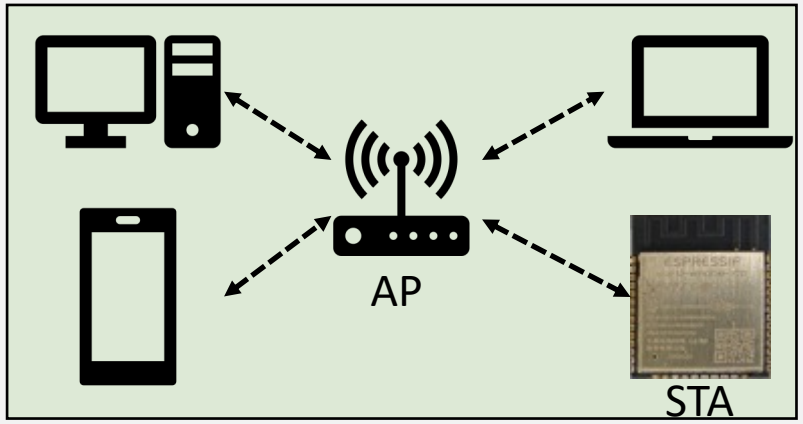
- Alimentée par
  - Prise micro-USB 5 V
  - Pile 9 vols avec régulateur de tension
- Prises femelles 2,4 mm (type « Arduino »)
- Bornes principales
  - Délivre 5 V, 3,3V
  - SCL /SDA pour I2C
  - 6 bornes analogues
  - 12 bornes digitales



Présentation du microcontrôleur

- ESP32 est équipé d'une infrastructure varié offrant des possibilités pour la communication sans fil et nécessitant que peu de puissance
- Fonctions principales
  - Routeur sans fil
  - Serveur web
  - Client web
  - Bluetooth : transfert fichier, clavier, haut-parleur, etc.
  - Bluetooth Low Energy : capteur médicaux, montres connectées, beacon





Gestion de l'énergie – Sleep mode

Mode	Wi-Fi / BT	ESP32	ULP co-processeur	RTC	Consommation
Active mode	X	X	X	X	160 à 260 mA
Modem sleep mode	A	X	X	X	3 à 20 mA
Light sleep mode		Pause	X	X	0,8mA
Deep sleep mode			X	X	10uA
Hibernation mode				X	2,5uA

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html

https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/

Programmation

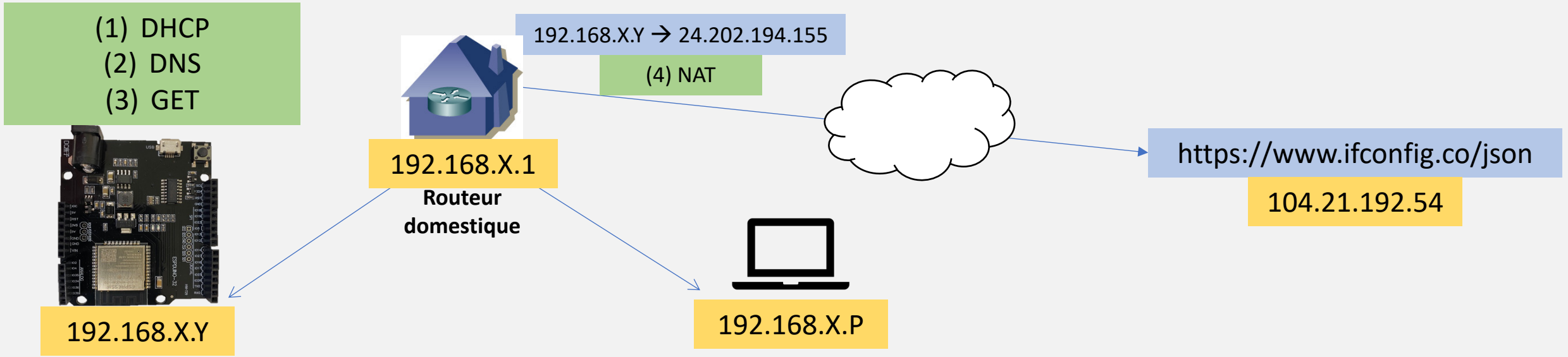
- Environnement PlateformIO
- Fichier PlateformIO.ini

```
[env:esp32doit-devkit-v1]
platform = espressif32
board = esp32doit-devkit-v1
framework = arduino
monitor_speed = 115200
```

Dépannage: vitesse de la console doit être 115200 bps
https://community.platformio.org/t/serial-monitor-speed-inconsistent-with-settings/5362

Client web

- Obtention d'une ressource HTTP
  - ESP32 doit avoir une adresse IP locale délivrée par DHCP (1)
  - ESP32 doit résoudre l'adresse DNS du site qui contient la ressource (2)
  - ESP32 peut ensuite envoyer une requête HTTP(S) (3)
  - ESP32 peut accéder à Internet à partir de l'adresse locale qui est traduite (NAT) par votre adresse IP globale délivrée par votre fournisseur d'accès internet (4)



https://fr.wikipedia.org/wiki/Network_address_translation https://fr.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol

Client WiFi – Se connecter sur votre réseau

```
#define WIFI_SSID     "VotreSSIDIci"
#define WIFI_PASSWORD "VotreMotDePasseIci"
```

```
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

Serial.print("Connexion : ");
while (nbEssais < nbEssaisMaximum && WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    nbEssais++;
}
Serial.println("");

if (WiFi.status() == WL_CONNECTED) {
    Serial.print("Connecté au réseau WiFi, adresse IP : ");
    Serial.println(WiFi.localIP());
    Serial.println("");
}
```

Client HTTP – Effectuer une requête GET

```
String obtenirIPPublique() {
    String res = "";
    if (WiFi.status() == WL_CONNECTED) {
        String url = "https://ifconfig.co/json";
        HTTPClient httpClient;
        httpClient.begin(url);
        int codeStatut = httpClient.GET();

        if (codeStatut != 200) {
                Serial.println(HTTPClient::errorToString(codeStatut));
        } else {
                res = httpClient.getString();
        }
    } else {
        Serial.println("Non connecté au WiFi !");
    }

    return res;
}
```

Sérialiser et désérialiser en/de JSON

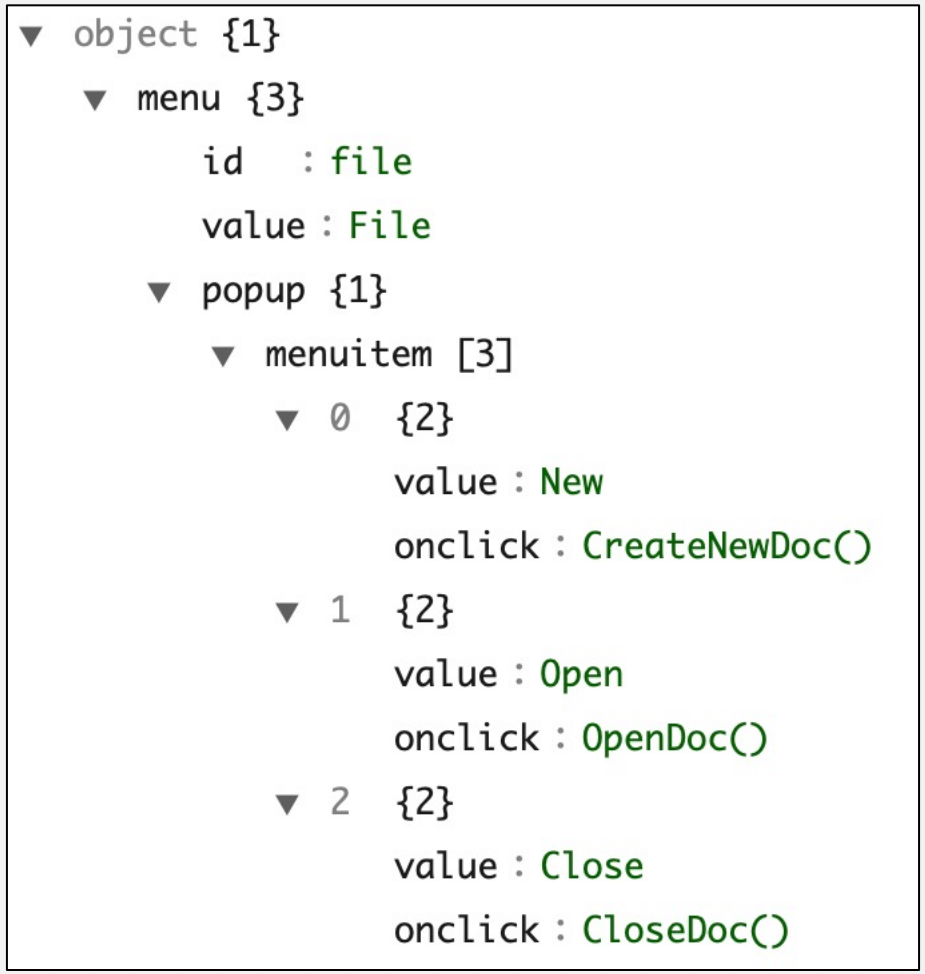
- Pour sérialiser et désérialiser des données en JSON ou à partir de JSON, nous vous demandons d'utiliser la bibliothèque « ArduinoJSON » https://arduinojson.org/
- Dans « platformio.ini », ajoutez la dépendance à cette bibliothèque

```
lib_deps =
        bblanchon/ArduinoJson @ ^6.18.0
```

- Vous pouvez utiliser l'assistant présent sur le site afin de vous aider à écrire vos méthodes de sérialisation / désérialisation

Désérialiser des documents JSON

Structure du document JSON à traiter




```
const char* json =
"{"
" \"menu\": {"
" \"id\": \"file\","
" \"value\": \"File\","
" \"popup\": {"
" \"menuitem\": ["
" {"
" \"value\": \"New\","
" \"onclick\": \"CreateNewDoc()\""
" },"
" {"
" \"value\": \"Open\","
" \"onclick\": \"OpenDoc()\""
" },"
" {"
" \"value\": \"Close\","
" \"onclick\": \"CloseDoc()\""
" }"
" ]"
" }"
" }"
"}";
```

https://json.org/example.html

Désérialiser des documents JSON

```
const char* json = "[...]"; // Voir code précédent
DynamicJsonDocument doc(1024);
DeserializationError error = deserializeJson(doc, json);

if (error) {
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.f_str());
    return;
}

JsonObject menu = doc["menu"];
const char* menu_id = menu["id"]; // "file"
const char* menu_value = menu["value"]; // "File"
Serial.println(String("id: ") + menu_id);
Serial.println(String("value: ") + menu_value);

for (JsonObject elem : menu["popup"]["menuitem"].as<JsonArray>()) {
    const char* value = elem["value"]; // "New", "Open", "Close"
    const char* onclick =
    elem["onclick"]; // "CreateNewDoc()", "OpenDoc()", "CloseDoc()"

    Serial.println(String("menu item: ") + value + " - " + onclick);
}
```

```
▼ object {1}
  ▼ menu {3}
      id    : file
      value : File
    ▼ popup {1}
      ▼ menuitem [3]
        ▼ 0  {2}
            value : New
            onclick : CreateNewDoc()
        ▼ 1  {2}
            value : Open
            onclick : OpenDoc()
        ▼ 2  {2}
            value : Close
            onclick : CloseDoc()
```

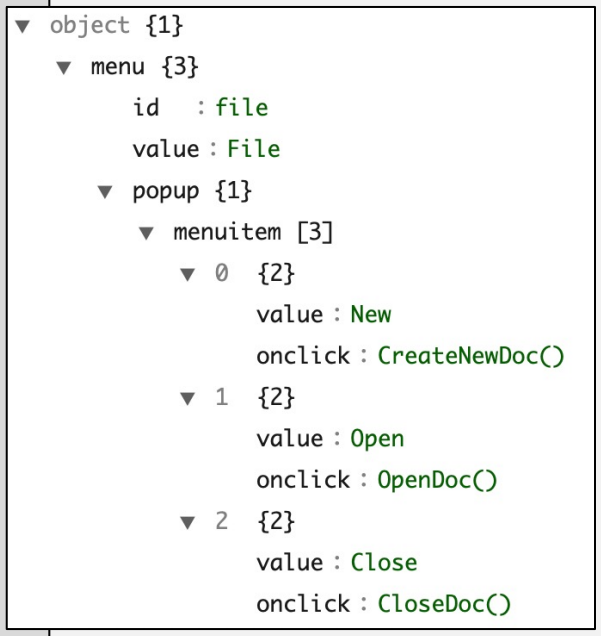
```
id: file
value: File
menu item: New - CreateNewDoc()
menu item: Open - OpenDoc()
menu item: Close - CloseDoc()
```

Majorité du code généré par : https://arduinojson.org/v6/assistant/

Sérialiser des documents JSON

```
DynamicJsonDocument doc(1024);

JsonObject menu = doc.createNestedObject("menu");
menu["id"] = "file";
menu["value"] = "File";

JsonArray menu_popup_menuitem = menu["popup"].createNestedArray("menuitem");

JsonObject menu_popup_menuitem_0 = menu_popup_menuitem.createNestedObject();
menu_popup_menuitem_0["value"] = "New";
menu_popup_menuitem_0["onclick"] = "CreateNewDoc()";

JsonObject menu_popup_menuitem_1 = menu_popup_menuitem.createNestedObject();
// ...
JsonObject menu_popup_menuitem_2 = menu_popup_menuitem.createNestedObject();
// ...

// File file = xyz.open(filename, FILE_WRITE);
// serializeJson(doc, file);
// file.close();
// ou serializeJson(doc, Serial);
char chaineTmp[1024];
serializeJson(doc, chaineTmp);
Serial.println(chaineTmp);
```

Structure du document JSON à créer




```
{"menu":{"id":"file","value":"File","popup":{"menuitem":[{"value":"New","onclick":"CreateNewDoc()"},{"value":"Open","onclick":"OpenDoc()"},{"value":"Close","onclick":"CloseDoc()"}]}}}
```


Majorité du code généré par : https://arduinojson.org/v6/assistant/

Références

- https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf : Datasheet
- https://fr.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol : DHCP
- https://fr.wikipedia.org/wiki/Network_address_translation : NAT
- https://arduinojson.org/v6/assistant/ : Assistant de création de code pour ArduinoJSON
- https://os.mbed.com/blog/entry/littlefs-high-integrity-embedded-fs/ : LittleFS
- https://json.org/example.html : Exemples JSON utilisés dans le cours