Amélioration d'un prototype : ***myVacuum***

Google Sheets Dashboard

Cours d'**Objets connectés – 4EN0303**

myVacuum

Auteur :
Davide DI VENTI - Master 1 Electronique - Immatriculé 60456
60456@etu.he2b.be

Professeure :
Mme A. DEGEEST

Département scolaire :
Haute Ecole Bruxelles-Brabant (*HE²B*)
Institut Supérieur Industriel de Bruxelles (*ISIB*)
Rue royale n°150, 1000 Bruxelles

Publié le 15 janvier 2024

Table des matières

Table des figures

1 Introduction

Ce rapport présente les résultats d'une étude menée dans le cadre du cours d'Objets Connectés du premier quadrimestre du master en ingénierie industrielle, option électronique, à l'Institut Supérieur Industriel de Bruxelles (*ISIB*). L'objectif principal de cette étude est d'optimiser un ancien prototype, nommé *"myVacuum"*, élaboré au cours des bureaux d'études de l'année scolaire 2022-2023. Cette amélioration se concentre sur l'intégration d'une solution connectée.

L'intégration de la connectivité vise à introduire des fonctionnalités avancées, ouvrant ainsi la porte à une expérience utilisateur plus riche et à une intégration intelligente des dernières avancées technologiques dans le domaine des *Dashboard* en ligne.

Actuellement, divers services proposent des solutions allant de la gratuité limitée à des offres payantes et professionnelles. A ce jours, plusieurs services permettraient la création d'un *Dashboard* notamment :

- *Blynk* : une plateforme *IoT* qui propose des solutions pour le développement d'applications mobiles connectées à des microcontrôleurs,
- *Adafruit IO* : une plateforme d'*IoT* d'*Adafruit*, offrant des fonctionnalités pour connecter des périphériques et visualiser des données en temps réel et
- *Home Assistant* : un système *open-source* pour la domotique qui peut être utilisé comme plateforme centrale pour connecter et automatiser différents appareils.

Tous ces services permettent de créer un tableau de bord et d'interagir avec les objets connectés respectifs. *Blynk* et *Adafruit IO* proposent un tarif gratuit, sous réserve d'une limitation du nombre de *widgets* et d'objets connectés. Au-delà de ces limites, les tarifs augmentent. En ce qui concerne *Home Assistant*, son tarif gratuit offre la possibilité quasi illimitée de construire un tableau de bord dans le réseau local. Cependant, si l'idée est d'étendre l'accès au tableau de bord à travers le monde, *Home Assistant* propose un service *cloud* payant permettant un accès en ligne.

L'objectif de ce rapport est de présenter une solution positionnée entre ces deux extrêmes. Malgré un tarif nul, la solution proposée, articulée autour d'un seul microcontrôleur *ESP32*, offre la possibilité d'avoir un tableau de bord aussi personnalisable que possible, accessible en ligne de n'importe où dans le monde.

2 Description du projet

2.1 Aspirateur myVacuum (rappel)

L'ancien projet consiste en un aspirateur doté de fonctionnalités numériques supplémentaires. Comme tout aspirateur domestique, il a pour fonction principale d'aspirer les miettes, la poussière, le sable, etc. Le voici :

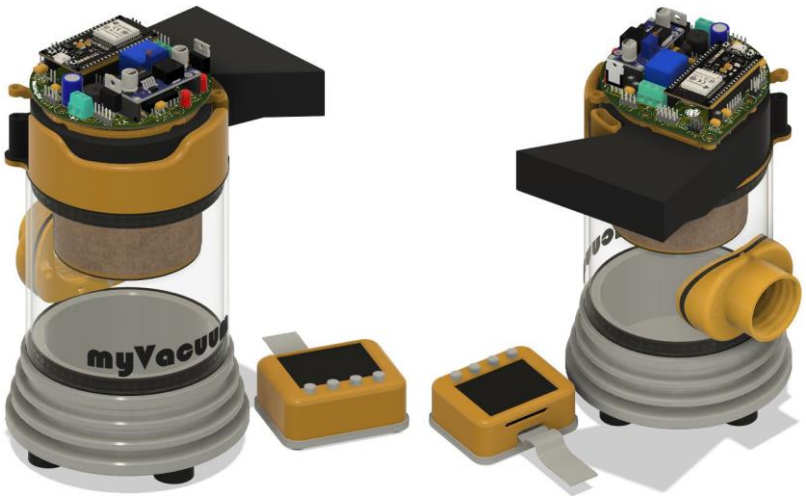

Figure 1 : Visualisation du prototype myVacuum

Toutefois, contrairement à ses homologues, *myVacuum* est capable de fournir à son utilisateur un état de santé en temps réel grâce à un écran. Cet état inclut des informations telles que : la température interne de l'appareil, le niveau de remplissage du réservoir, la qualité de l'air aspiré, l'humidité dans le réservoir, et la vitesse tachymétrique du ventilateur d'aspiration.

L'utilisateur peut également contrôler l'aspirateur à partir de cet écran, une interface utilisateur. Cette dernière prend la forme d'une télécommande filaire équipée d'un petit écran et de 4 boutons poussoirs. L'interface comprend plusieurs menus préprogrammés, notamment une zone permettant de visualiser et de commander l'état de l'appareil, ainsi qu'une autre zone permettant d'acquérir des données et de créer un système de *logs* à partir d'une carte *SD*. Il s'agit d'un aspirateur industriel miniaturisé. Il est équipé de fonctions permettant de régler la puissance d'aspiration graduellement et d'activer un relais embarqué selon les besoins.

Cet aspirateur a été conçu dans le cadre de cours scolaires antérieurs, notamment dans le cadre des bureaux d'études. Il avait été mentionné dans le rapport précédent [1] qu'une amélioration pourrait être envisagée, consistant à apporter une solution connectée à l'objet.

2.2 Solution connectée apportée

Le choix du microcontrôleur pour ce projet s'est porté sur l'*ESP32*. Cette décision découle de considérations techniques, notamment en raison de sa puissance de calcul et de sa polyvalence *GPIO*, permettant une exploitation optimale de ses capacités. Il est intéressant de noter que l'*ESP32* est également capable d'exécuter des tâches *WLAN*, telles que le *Wifi 2,4 GHz* et le *Bluetooth 2,4 GHz*. Cependant, l'idée d'utiliser ces fonctionnalités a été mise de côté initialement, mettant ainsi l'accent sur l'optimisation de l'électronique dans un premier temps.

Actuellement, afin de maximiser les fonctionnalités de l'aspirateur, une solution connectée a été intégrée dans le cadre du cours sur les objets connectés. Un *Dashboard* a été développé dans le but de permettre le contrôle à distance de l'aspirateur en plus de son utilisation avec la télécommande filaire.

Pour garantir la singularité de ce *Dashboard* par rapport aux solutions existantes, certaines contraintes ont été prises en compte, notamment la nécessité d'une personnalisation illimitée, d'une accessibilité à tout endroit et d'une gratuité totale.

2.3 Démonstration

La solution connectée choisie pour ce projet a été intégrée dans un *Google Sheets*. Ce choix a été motivé par la disponibilité d'une *API* permettant d'effectuer des requêtes *HTTP*, notamment des *GET* et *POST*.

Le *Google Sheets* présente l'avantage d'être accessible à tous, sous réserve de disposer d'un compte *Google*. Sa fonction principale consiste à traiter des données dans des tableaux, permettant l'application de fonctions mathématiques, la génération de graphiques, l'analyse de statistiques et de probabilités, tout comme dans *Excel*, mais en ligne.

La nature native en ligne du *Google Sheets* lui confère une portée étendue, offrant la possibilité d'intégrer de nouvelles fonctionnalités via des *plugins* et surtout de profiter pleinement de l'écosystème *Google*. Parmi les nombreux services *Google* disponibles en ligne, l'un d'entre eux a joué un rôle essentiel dans l'automatisation des tâches du *Dashboard* conçu pour *myVacuum* : *Apps Script*. Cette plateforme utilise une version étendue de *JavaScript*, spécifiquement adaptée à l'écosystème *Google*. Elle a facilité l'interaction de l'interface avec des requêtes *HTTP* externes et la liaison de fonctions codées à des *widgets*.

Désormais, les données telles que la température interne de l'appareil, le niveau de remplissage du réservoir, la qualité de l'air aspiré, l'humidité dans le réservoir et la vitesse tachymétrique du ventilateur d'aspiration de l'aspirateur sont accessibles en ligne. De plus, elles peuvent être personnalisées selon les préférences de l'utilisateur et surtout, cette fonctionnalité est gratuite. Ci-dessous, un aperçu de quatre images capturées à différents moments du *Dashboard* :

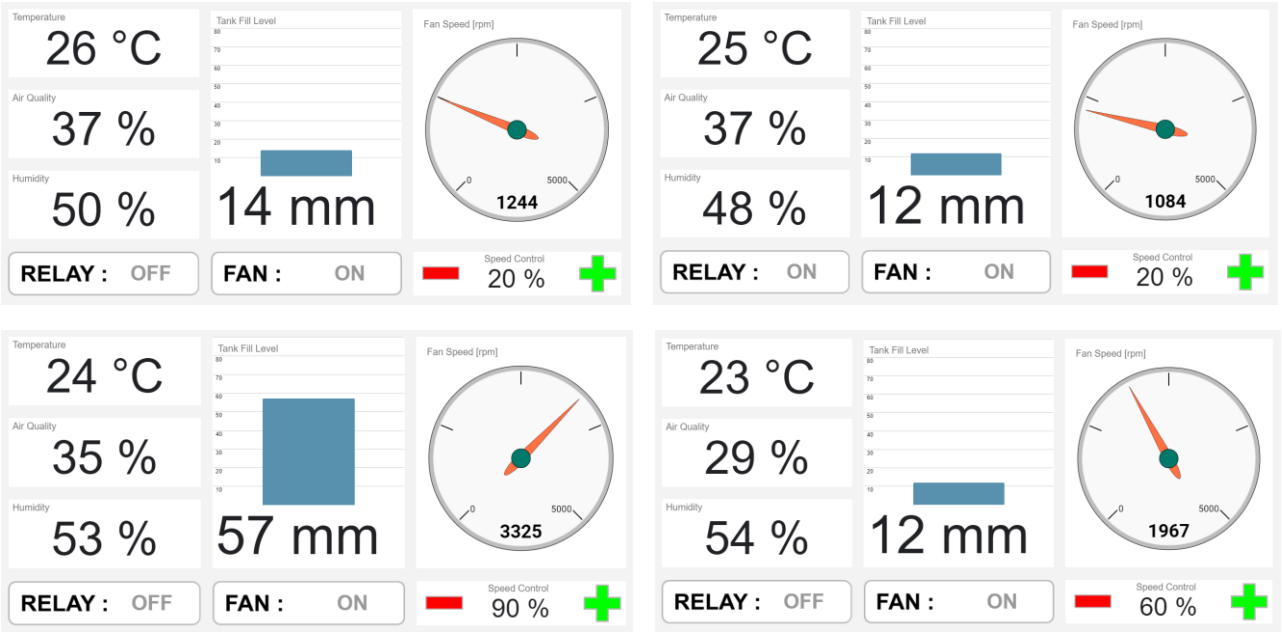

Figure 2 : Images capturées du Dashboard

Le *Dashboard* se compose de quatre boutons, chacun ayant une fonction spécifique. L'un active le relais embarqué (*RELAY*), un autre déclenche la ventilation (*FAN*), tandis que les deux derniers permettent de réguler la vitesse du ventilateur (- et +). Chacun de ces boutons a été personnalisé avec une forme et une logique de code spécifiques, afin de correspondre aux préférences et aux besoins définis.

L'état de remplissage du réservoir est représenté par un diagramme en bâtonnet, qui, bien qu'original, s'intègre parfaitement dans le thème du *Dashboard*. La vitesse du ventilateur est quant à elle affichée à l'aide d'un graphique en jauge, nativement disponible dans l'éditeur *Google Sheets*. La mise en page flexible de *Google Sheets* a joué un rôle essentiel en permettant non seulement de créer une interface de *Dashboard*, mais également de le rendre authentiquement fonctionnel.

3 Google Sheets

A partir de cette section, la méthodologie détaillera la mise en place du système. La première étape consiste à élaborer les différentes mécaniques dans *Google Sheets*.

3.1 Architecture des feuilles de calculs

L'idée est de diviser le système en deux feuilles de calcul. Une première feuille, présentée à la figure 3 et nommée "*Sheet1*", est constituée d'un tableau de 9 lignes sur 3 colonnes. Les trois premières lignes représentent les actions possibles à effectuer sur l'aspirateur : Aspiration, Puissance (vitesse) et Relais. Les lignes suivantes contiennent les données lues depuis l'aspirateur. La dernière ligne indique le nombre de requêtes effectuées depuis l'allumage de l'aspirateur.

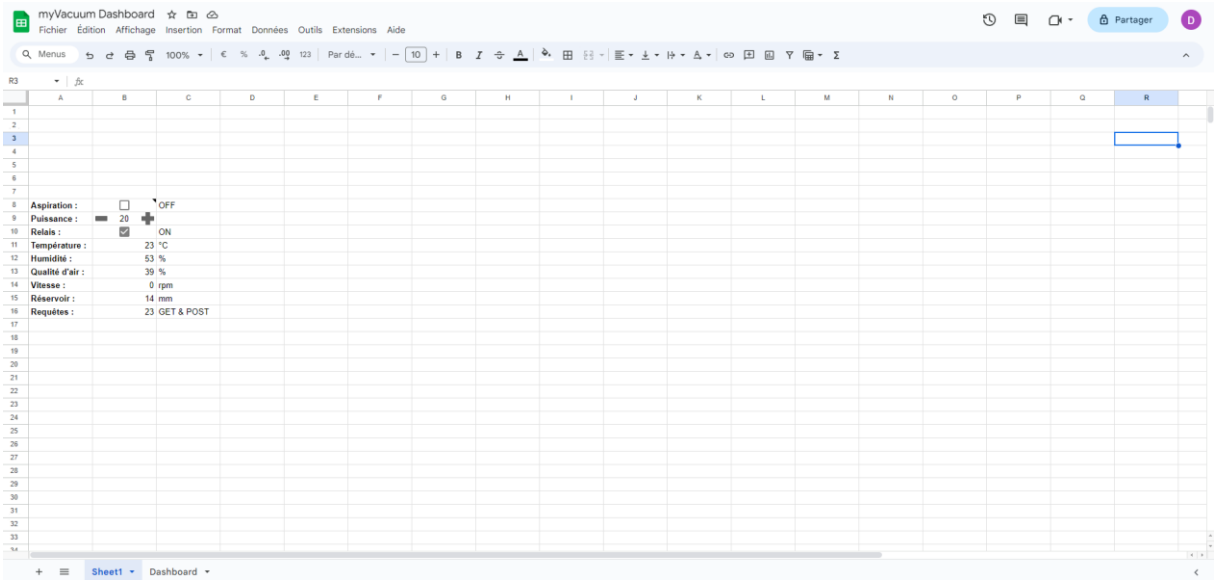

Figure 3 : Feuille de calcul : Sheet1

C'est dans cette feuille que toutes les interactions avec *Apps Scripts* seront exécutées.

Quant à la deuxième feuille, illustrée à la figure 4 et intitulée "*Dashboard*", elle contiendra les différents *widgets* affichant des valeurs utiles. Ce *Dashboard* reprend les informations de la *Sheet1*.

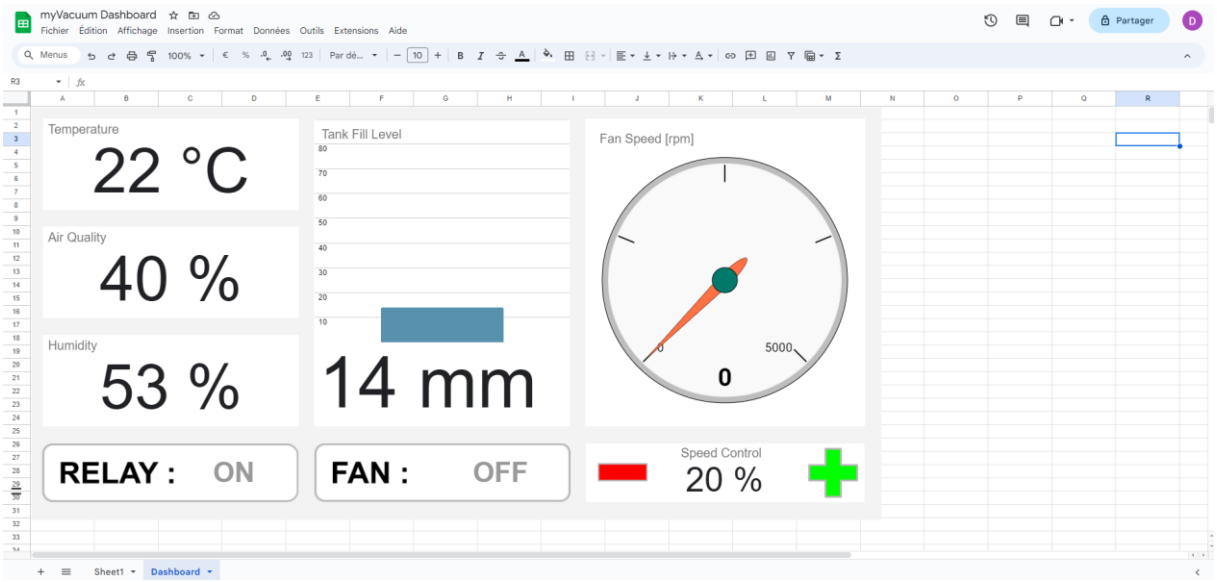

Figure 4 : Feuille de calcul : Dashboard

Dans la section suivante sera présenté comment les *widgets* du Dashboard peuvent etre misent en place.

3.2 Création d'une interface Dashboard

La feuille *Sheet1* seule serait suffisante pour mettre en place la solution connectée, mais se limiter à une interface basée sur des cellules n'est pas considéré comme esthétique. Ainsi, la deuxième feuille a été ajoutée pour dynamiser le contenu sous forme de *Dashboard*.

Widgets entrants (Température, Vitesse, …) :

Dans cette deuxième feuille, pour intégrer, par exemple, le *widget* de température, il suffit d'accéder à l'éditeur de graphiques. Pour ce faire, une fois la page *Google Sheets* ouverte, cliquer sur "*Insérer*", puis sur "*Graphique*", et l'éditeur sera ouvert sur le côté. Voici à quoi ressemblent les cinq différentes fenêtres de l'éditeur :

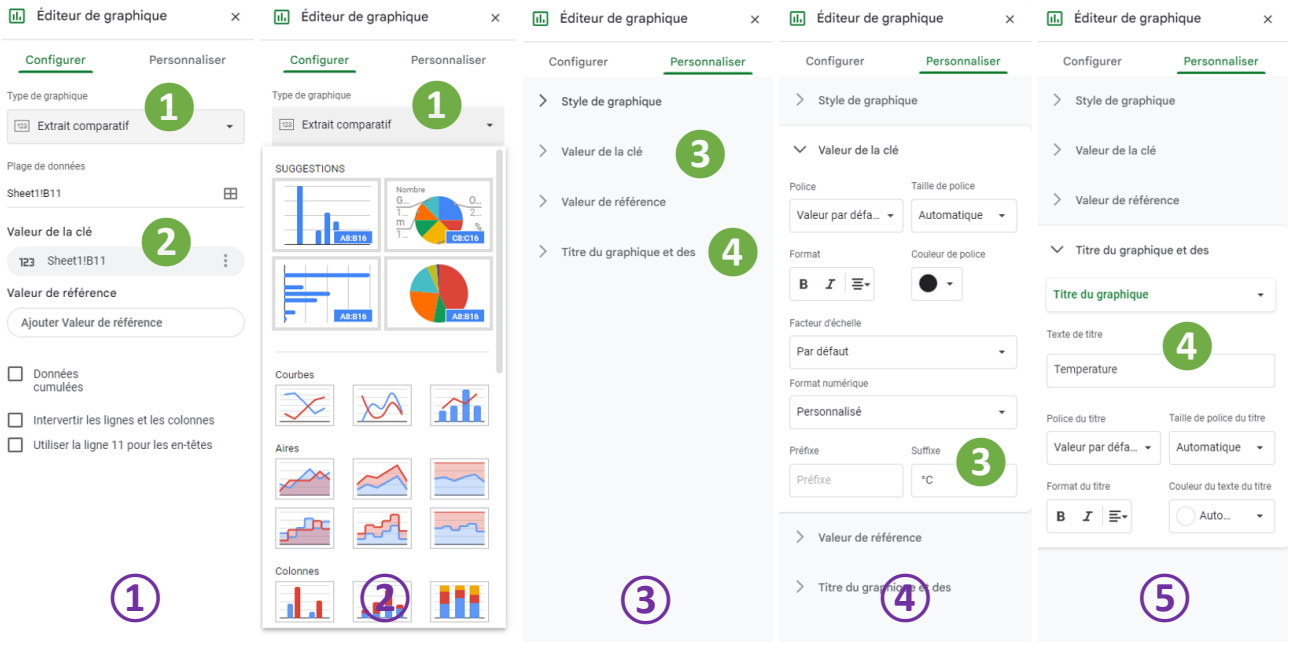

Figure 5 : Fenêtres de l'éditeur de graphique

Les étapes sont indiquées en vert. Dans cet éditeur, la première étape dans l'onglet "*Configurer*" consiste à choisir le type de graphique. Dans le cas du *widget* de température, il s'agit d'un graphique en extrait comparatif. Pour le *widget* du niveau de remplissage du réservoir, c'est un graphique en bâtonnet.

La deuxième étape dans l'onglet "*Configurer*" permet d'assigner au *widget* la valeur de la clé. Cette valeur correspond à la donnée utile à afficher. Dans le cas de la température, cette valeur provient de la *Sheet1* à la cellule B11, qui est par exemple de 22.

La troisième étape consiste à ajouter un suffixe à la donnée. Dans l'onglet "*Personnaliser*", plusieurs options sont disponibles. Cliquer sur "*Valeur de la clé*", puis accéder au suffixe. Ajouter ensuite "*°C*".

Enfin, la quatrième étape consiste à donner un texte au titre. Accéder à l'onglet correspondant et insérer "*Température*".

Widgets sortants (Boutons) :

À ce stade, le *widget* de température a été créé. Il ne reste plus qu'à le redimensionner et à le placer à l'endroit souhaité. Avec un peu d'exercice, les autres *widgets* suivent le même principe, à l'exception des boutons.

Les boutons sont issus d'un autre type d'édition. Il ne s'agit pas d'une insertion de graphique, mais plutôt d'un dessin ajouté. Sur la page *Google Sheets*, cliquer sur "*Insertion*", puis "*Dessin*". Un éditeur s'ouvrira.

Cet éditeur propose divers outils pour dessiner des figures. Dans le cas du bouton *RELAY*, il s'agit d'un rectangle à coins arrondis, avec un rebord gris, un remplissage blanc, et un texte noir. Une fois placé dans la feuille de calcul, il sera possible de lui assigner un script provenant d'*Apps Scripts* (voir la section suivante).

4 Apps Script

À ce stade, le *Dashboard* est mis en place, mais aucune automatisation n'a été ajoutée. L'ajout d'un code dans l'*Apps Script* permettra de mettre en place les protocoles agissant sur la mécanique du *Dashboard* et de l'*ESP32*.

4.1 Code.gs Google Script

Pour accéder à l'interface d'*Apps Script* depuis une feuille de calcul *Google Sheets*, cliquer sur "*Extensions*", puis sur "*Apps Script*". La page s'ouvrira comme suit :

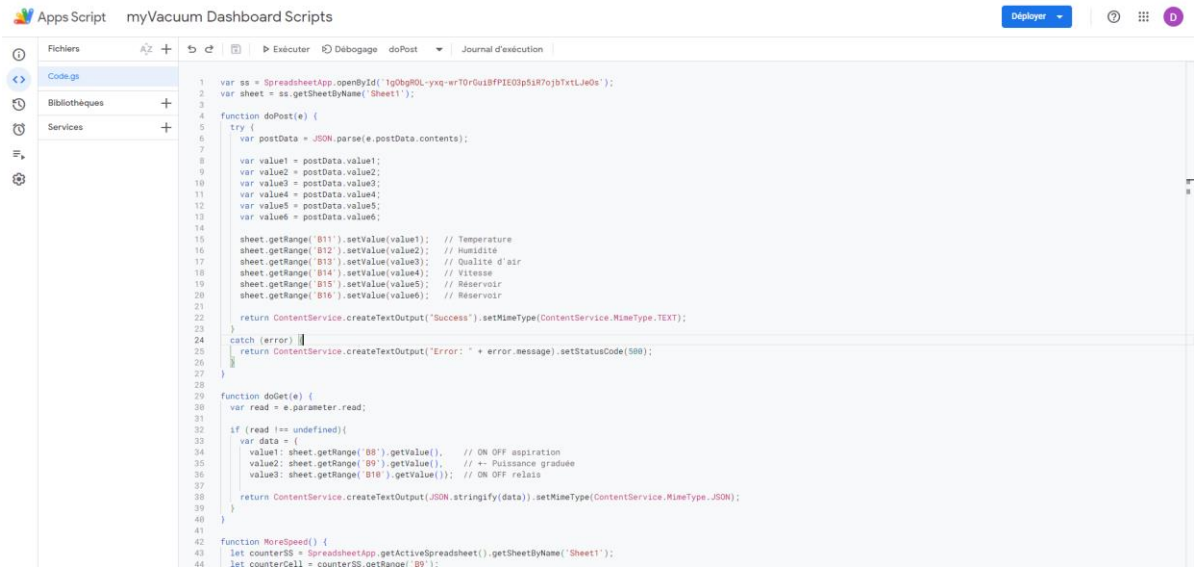

Figure 6 : Page éditrice de Apps Script

Bien entendu, le script doit être rédigé et ne sera pas présent dès l'ouverture. Le code se trouve en annexe 1 et est composé de plusieurs fonctions.

Explication du code :

La première ligne concerne l'*ID* de la page *Google Sheets*. Cette *ID* doit être copiée depuis l'*URL* de la page *Google Sheets*, comme représenté dans la figure suivante :

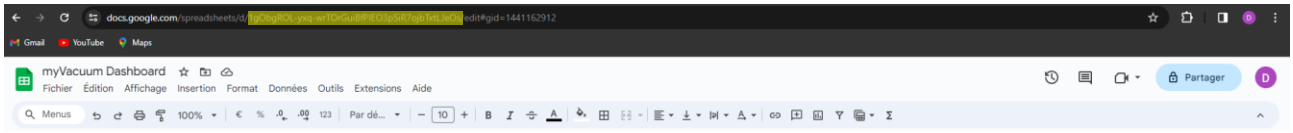

Figure 7 : ID du Google Sheets

À la ligne 4 se trouve la fonction *doPost()*. Elle récupère les données postées par événement depuis l'*ESP32* sous forme de *JSON*, avec des paires de clés et de valeurs. Chaque valeur correspond à une donnée provenant des capteurs de l'aspirateur. Ces valeurs sont ensuite affichées dans leurs cellules respectives de *Sheet1*.

```
1  var ss = SpreadsheetApp.openById('1gObgROL-yxq-wrTOrGuiBfPIEO3p5iR7ojbTxtLJeOs');
2  var sheet = ss.getSheetByName('Sheet1');
3
4  function doPost(e) {
5    try {
6      var postData = JSON.parse(e.postData.contents);
7
8      var value1 = postData.value1;  // Récupération de la valeur1 depuis les données POST
9      var value2 = postData.value2;  // Récupération de la valeur2 depuis les données POST
10     var value3 = postData.value3;  // Récupération de la valeur3 depuis les données POST
11     var value4 = postData.value4;  // Récupération de la valeur4 depuis les données POST
12     var value5 = postData.value5;  // Récupération de la valeur5 depuis les données POST
13     var value6 = postData.value6;  // Récupération de la valeur6 depuis les données POST
14
15     // Mise à jour des données dans la feuille de calcul
16     sheet.getRange('B11').setValue(value1);   // Température
17     sheet.getRange('B12').setValue(value2);   // Humidité
18     sheet.getRange('B13').setValue(value3);   // Qualité d'air
19     sheet.getRange('B14').setValue(value4);   // Vitesse
20     sheet.getRange('B15').setValue(value5);   // Réservoir
21     sheet.getRange('B16').setValue(value6);   // # Requêtes
```

À la ligne 30 se trouve la fonction *doGet()*. Elle répond aux requêtes *GET* effectuées par l'*ESP32*. Cette fonction permet de lire l'état des différents boutons du *Dashboard* afin que l'*ESP32* puisse effectuer l'action souhaitée, à savoir activer le relais, l'aspiration, et ajuster sa puissance. Les données demandées sont envoyées sous forme de *JSON* également.

```
30  function doGet(e) {
31    var read = e.parameter.read;
32
33    if (read !== undefined){
34      // Lecture des données depuis la feuille de calcul et renvoi au format JSON
35      var data = {
36        value1: sheet.getRange('B8').getValue(),    // ON OFF aspiration
37        value2: sheet.getRange('B9').getValue(),    // +- Puissance graduée
38        value3: sheet.getRange('B10').getValue()};  // ON OFF relais
39
40      return ContentService.createTextOutput(JSON.stringify(data)).setMimeType(ContentService.MimeType.JSON);
41    }
42  }
```

Aux lignes 44 et 55 se trouvent les fonctions assignées aux boutons + et – permettant de régler l'aspiration. Ces fonctions se basent sur la cellule B9 de la *Sheet1* afin de la faire varier entre 0 et 100 par pas de 10.

```
44  function MoreSpeed() {
45    // Augmentation de la vitesse de la cellule B9 de 10 si la valeur actuelle est inférieure à 99
46    let counterSS = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Sheet1');
47    let counterCell = counterSS.getRange('B9');
48    let currentValue = counterCell.getValue();
49
50    if (currentValue < 99) {
51      counterCell.setValue(currentValue + 10);
52    }
53  }
54
55  function LessSpeed() {
56    // Diminution de la vitesse de la cellule B9 de 10 si la valeur actuelle est supérieure à 0
57    let counterSS = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Sheet1');
58    let counterCell = counterSS.getRange('B9');
59    let currentValue = counterCell.getValue();
60
61    if (currentValue > 0) {
62      counterCell.setValue(currentValue - 10);
63    }
64  }
```

Aux lignes 66 et 79 se trouvent les fonctions assignées aux boutons *FAN* et *RELAY*. La première fonction permet d'activer ou de désactiver l'aspiration. Elle se base sur la cellule C8 (*ON/OFF*) de la *Sheet1* afin de changer son texte à l'appui et de la cellule B8 afin de synchroniser la case à cocher. La deuxième fonction permet d'activer ou de désactiver le relais. Le principe est le même que celui précédent.

```
66  function EnableVacuum(){
67    // Inversion de la valeur de la cellule B8 et affichage correspondant dans la cellule C8
68    let sheet1 = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Sheet1');
69    let counterCellB8 = sheet1.getRange('B8');
70    let counterCellC8 = sheet1.getRange('C8');
71
72    let currentValue = counterCellB8.getValue();
73    counterCellB8.setValue(!currentValue);
74
75    let onOffText = currentValue ? 'OFF' : 'ON';
76    counterCellC8.setValue(onOffText);
77  }
78
79  function EnableRelay(){
80    // Inversion de la valeur de la cellule B10 et affichage correspondant dans la cellule C10
81    let sheet1 = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Sheet1');
82    let counterCellB10 = sheet1.getRange('B10');
83    let counterCellC10 = sheet1.getRange('C10');
84
85    let currentValue = counterCellB10.getValue();
86    counterCellB10.setValue(!(currentValue));
87
88    let onOffText = currentValue ? 'OFF' : 'ON';
89    counterCellC10.setValue(onOffText);
90  }
```

4.2 Déploiement du script

Une fois le code rédigé, il doit être déployé pour qu'il soit reconnu par le *Google Sheets* et l'*ESP32*. Pour ce faire, sur la page *Apps Script*, cliquer en haut à droite sur "*Déployer*", puis "*Nouveau déploiement*" (valider l'accès à tout le monde), et enfin "*Déployer*". Un *ID* de déploiement unique apparaîtra : *AKfycbzjF98lDuM3QgB7Kz sosUrOBTz47l20ZqSVrmVz3VsRHXONJzTgpJYeMSJy_aiXGlyk*. Il faudra le copier dans le presse-papiers pour l'insérer dans le code de l'*ESP32*.

5 Arduino IDE

Dans un souci de clarté, le code a été simplifié pour ne pas afficher les quelques milliers de lignes du prototype *myVacuum*. Les lignes de code qui seront bientôt expliquées sont celles utilisées dans le prototype, mais elles sont réparties suivant la logique à suivre. Seules les lignes de code nécessaires à la mise en place de la communication avec le *Google Sheets* seront expliquées.

Ce code peut être considéré comme un exemple fonctionnel. Un simple copier-coller de celui-ci sera opérationnel. Il est entièrement compatible avec le *Google Script* du *Dashboard* précédemment expliqué si nécessaire. Les données envoyées ne seront tout simplement pas issues de capteurs, mais plutôt d'un compteur.

5.1 Code.ino ESP32

L'annexe 2 présente le code de l'*ESP32*.

Explication du code :

Le code commence par la construction de quelques variables. À la ligne 7, on retrouve l'*ID* généré lors du déploiement du code Apps Script à coller. Les identifiants du point d'accès *Wi-Fi* sont situés aux lignes 5 et 6. Le compteur initialisé à la ligne 11 sera utilisé pour simuler les données générées par des capteurs.

```
1  #include <WiFi.h>
2  #include <HTTPClient.h>
3  #include <ArduinoJson.h>
4
5  const char * ssid = "GSMDavide";          // Nom du réseau WiFi
6  const char * password = "987654321";   // Mot de passe du réseau WiFi
7  String GOOGLE_SCRIPT_ID = "AKfycbzjF98lDuM3QgB7KzsosUrOBTz47l20ZqSVrmVz3VsRHXONJzTgpJYeMSJy_aiXGlyk";  // ID du script Google associé
8  const int sendInterval = 10000;              // Intervalle entre les envois de données en millisecondes
9  WiFiClientSecure client;                     // Client WiFi sécurisé
10
11 int counter;                                 // Compteur pour générer des données
12 unsigned long startTime;                     // Temps de démarrage pour mesurer le temps écoulé
13
```

Avant de poursuivre avec les lignes suivantes, il est plus pertinent de suivre la logique du code. À la ligne 62, il y a le *setup*. À l'intérieur, la connexion à la borne d'accès *Wi-Fi* est établie. Ainsi, au démarrage, l'*ESP32* commence par se connecter.

```
62 void setup() {
63   Serial.begin(115200);                    // Initialisation de la communication série
64   delay(10);
65   WiFi.mode(WIFI_STA);                     // Mode station WiFi
66   WiFi.begin(ssid, password);              // Connexion au réseau WiFi
67   Serial.print("Connecting");
68   while (WiFi.status() != WL_CONNECTED) {  // Attente de la connexion au WiFi
69     delay(500);
70     Serial.print(".");
71   }
72   Serial.println("\nStarting");
73 }
```

Ensuite, à la ligne 75, la *loop* s'enclenche. Aucun délai ne cadence cette boucle en raison des latences naturellement présentes suite aux émissions et réceptions des requêtes *HTTP*. Ces fonctions sont bloquantes. En premier lieu, l'émission de données de capteurs s'effectue via un *POST*. Un total de 6 données sont envoyées. Ces données seront affichées dans la *Sheet1* du *Google Sheets*. En deuxième lieu, un *GET* est émis, demandant de récupérer les 3 valeurs des boutons dans le *Dashboard* en ligne.

```
75 void loop() {
76   startTime = millis();                    // Enregistrement du temps de début
77   http_post(counter, counter+1, counter-1, counter/2, counter*2, counter%2);  // Appel de la fonction d'envoi POST avec des valeurs générées
78   Serial.println("[" + String(millis() - startTime)+ "ms]");  // Affichage du temps écoulé
79
80   startTime = millis();                    // Enregistrement du temps de début
81   http_get();                              // Appel de la fonction de requête GET
82   Serial.println("[" + String(millis() - startTime)+ "ms]");
83
84   counter ++;
85 }
```

A titre indicatif, en moyenne, la *loop* effectue une boucle toutes les 15 secondes en raison de la puissance de calcul de l'*ESP32* et des latences *HTTP*. Un affichage sur la console informe le temps écoulé entre chaque requête.

Les requêtes *HTTP* se basent sur la création de macros sous forme d'*URL* émises dans le réseau. A la ligne 14 se trouve justement la fonction *http_post()*. Suivant la macro émise, la demande est soit *POST* ou *GET*. Dans le cas de cette fonction, il s'agit d'un *POST*. Elle est appelée dans la *loop*. Dans cette requête se trouvera un *JSON* composé des diverses données des capteurs, à savoir *value1*, *value2*, … et *value6*. La fonction *http.POST(postData)* appelée à la ligne 29 est celle qui se charge de l'émission. Il s'agit de la fonction bloquante du processus. Une fois réussie, elle génère un *payload*, indiquant la réussite ou l'échec de l'émission.

```
14  void http_post(int value1, int value2, int value3, int value4, int value5, int value6) {
15    HTTPClient http;                          // Création d'un objet HTTPClient
16    String url = "https://script.google.com/macros/s/" + GOOGLE_SCRIPT_ID + "/exec";  // Construction de l'URL pour l'envoi POST vers le script Google
17    Serial.print("POST ");                    // Affichage dans la console série
18    http.begin(url.c_str());                  // Initialisation de la connexion HTTP
19    http.addHeader("Content-Type", "application/json");  // Ajout de l'en-tête de contenu JSON
20    DynamicJsonDocument jsonDocument(200);  // Création d'un document JSON dynamique
21    jsonDocument["value1"] = value1;        // Remplissage du document JSON avec les valeurs à envoyer
22    jsonDocument["value2"] = value2;
23    jsonDocument["value3"] = value3;
24    jsonDocument["value4"] = value4;
25    jsonDocument["value5"] = value5;
26    jsonDocument["value6"] = value6;
27    String postData;                          // Création d'une chaîne de caractères pour stocker les données JSON sérialisées
28    serializeJson(jsonDocument, postData);  // Sérialisation du document JSON
29    int httpCode = http.POST(postData);   // Envoi des données au script Google via une requête POST
30    String payload;                           // Réponse HTTP
31    unsigned long elapsedTime = millis() - startTime;  // Calcul du temps écoulé
32    if (httpCode > 0) {                       // Si la requête HTTP a réussi
33      payload = http.getString();             // Récupération de la réponse HTTP
34    } else {
35      Serial.println(http.errorToString(httpCode).c_str());  // Affichage des erreurs en cas d'échec
36    }
37    http.end();                               // Fin de la connexion HTTP
38  }
```

La fonction suivante, à la ligne 40, est *http_get()*. À la suite de la fonction *POST*, celle-ci est appelée dans la *loop*. Il y a création de l'*URL* en question puis émission de celle-ci. La ligne 45 précise au client *HTTP* de suivre strictement les redirections. Cette approche peut améliorer la sécurité en évitant des comportements indésirables, surtout lorsque les redirections sont utilisées dans un contexte sensible. À la ligne 46, l'émission et l'attente de réponse ont lieu. Une fois la demande répondue par le script de *Apps Script*, le *payload* se voit composé des valeurs des boutons du *Dashboard*, à savoir l'activation du relais, du ventilateur et de la puissance d'aspiration souhaitée. Ces valeurs sont stockées sous forme de variables locales dans cette fonction. Elles peuvent ensuite être traitées suivant l'effet souhaité.

```
40  void http_get() {
41    HTTPClient http;                          // Création d'un objet HTTPClient
42    String url = "https://script.google.com/macros/s/" + GOOGLE_SCRIPT_ID + "/exec?read";  // Construction de l'URL pour la requête GET
43    Serial.print("GET ");                     // Affichage dans la console série
44    http.begin(url.c_str());                  // Initialisation de la connexion HTTP
45    http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);  // Suivre strictement les redirections HTTP
46    int httpCode = http.GET();                // Envoi de la requête GET au script Google
47    String payload;                           // Réponse HTTP
48    if (httpCode > 0) {                       // Si la requête HTTP a réussi
49      payload = http.getString();             // Récupération de la réponse HTTP
50      Serial.print(payload);                  // Affichage de la réponse dans la console série
51      DynamicJsonDocument jsonDocument(200);  // Création d'un document JSON dynamique
52      deserializeJson(jsonDocument, payload);       // Désérialisation de la réponse JSON
53      int value1 = jsonDocument["value1"];          // Extraction des valeurs du document JSON
54      int value2 = jsonDocument["value2"];
55      int value3 = jsonDocument["value3"];
56    } else {
57      Serial.println("Error on HTTP GET request");  // Affichage des erreurs en cas d'échec
58    }
59    http.end();                               // Fin de la connexion HTTP
60  }
```

5.2 Interface utilisateur myVacuum

L'intégralité de la solution connectée étant achevée, le système d'objet connecté est désormais opérationnel. Depuis la télécommande filaire de l'aspirateur, il est possible de visualiser le défilement des requêtes envoyées.

Figure 8 : Console sur la télécommande filaire de myVacuum

6 Améliorations

Diverses améliorations potentielles peuvent être implémentées dans le système. Par exemple, l'intégration de graphiques alimentés par un historique de données pourrait constituer une addition notable au *Dashboard*. Cette fonctionnalité offrirait une perspective plus approfondie, complémentaire aux données affichées en temps réel. L'incorporation de telles améliorations conférerait une dimension plus immersive au *Dashboard*. Par ailleurs, des améliorations plus significatives peuvent être envisagées, et deux exemples concrets sont détaillés dans cette section.

6.1 WebSocket

Un inconvénient réside dans le fait d'utiliser une requête *GET* pour récupérer l'état des boutons du *Dashboard*. Ce mécanisme représente une limitation pour le système, car lorsqu'un bouton est pressé sur le *Dashboard*, il est nécessaire d'attendre que l'*ESP32* effectue une requête *GET* avant que l'action correspondante ne soit effectivement déclenchée à distance sur l'aspirateur (entre 8 et 12 secondes).

Une solution pour surmonter le délai lié à l'utilisation de requêtes *GET* pour identifier l'état des boutons du *Dashboard* pourrait être d'adopter une approche asynchrone. Établir une connexion *WebSocket* entre le *Dashboard* et l'*ESP32* plutôt qu'effectuer des *GET* changeraient radicalement les choses. Les *WebSocket* permettent une communication bidirectionnelle en temps réel, ce qui signifie que l'*ESP32* peut recevoir des mises à jour instantanées dès qu'un bouton est pressé sur le *Dashboard*.

Contrairement à *HTTP*, qui est basé sur une communication unidirectionnelle, le protocole *WebSocket* permet une communication bidirectionnelle continue et basée sur des messages entre le client et le serveur. *Apps Script* ne prend pas en charge *WebSocket* de manière native, mais il existe des bibliothèques tierces qu'il est possible à intégrer.

6.2 DualCore

Les latences du système, engendrées par les fonctions bloquantes, pourraient bénéficier d'une amélioration. Une approche envisageable consisterait à exploiter les deux cœurs de l'*ESP32*, permettant ainsi de déclencher des tâches en arrière-plan sans attendre la réponse du serveur lorsqu'une action est en cours. Cette démarche pourrait significativement améliorer la réactivité globale du système, car le microcontrôleur serait en mesure de poursuivre l'exécution d'autres tâches pendant que des opérations plus gourmandes sont en cours.

7 Conclusion

Pour investiguer le phénomène des latences, une expérience a été menée en démarrant le système à 16h le vendredi 22.12.23. Durant les 7 premières heures, 1000 requêtes ont été effectuées. Il a été observé que certaines requêtes se perdent, maintenant un temps d'envoi constant de 120 000 ms. À partir de 23h, aucune requête n'a dépassé les 10 secondes de latence.

Il est à noter qu'à deux reprises (à 17h40 et à 1h15), les requêtes *GET* et *POST* se sont stabilisées exactement à 7 000 ms. Bien que la console ait affiché plusieurs *GET* et *POST* consécutifs de 7000 ms, aucune information ne s'est actualisée sur le *Dashboard* pendant ces périodes, rendant ces requêtes inexploitables.

Le test a été clôturé à 2h du matin, totalisant ainsi une expérience de 10 heures avec un total de 1500 requêtes. À partir de 1h, toutes les requêtes *GET* ont mis plus de 10 secondes. Ces observations soulignent des aspects importants de la stabilité et de la latence du système lorsqu'il est soumis à une charge prolongée.

Ces résultats mettent en évidence l'instabilité du système lorsqu'il est soumis à une charge prolongée. La fiabilité d'un objet connecté réside sur la stabilité justement. Pour remédier à cela, les étapes suivantes du projet consisteraient à implémenter les améliorations préalablement mentionnées, à savoir l'intégration d'un *WebSocket* et la mise en place structurée de tâches réparties sur les deux cœurs de l'*ESP32*.

Bibliographie

[1] Rapport des bureaux d'études de myVacuum, publié le 16 mai 2023, par DavideDiVenti, en ligne, https://github.com/DavideDiVenti/myVacuum/blob/main/Documents/Rapport.pdf, consulté le 16 mai 2023.

Annexe 1 : Code.gs Apps Scripts

```
var ss = SpreadsheetApp.openById('1gObgROL-yxq-wrTOrGuiBfPIEO3p5iR7ojbTxtLJeOs');
var sheet = ss.getSheetByName('Sheet1');

function doPost(e) {
  try {
    var postData = JSON.parse(e.postData.contents);

    var value1 = postData.value1;  // Récupération de la valeur1 depuis les données POST
    var value2 = postData.value2;  // Récupération de la valeur2 depuis les données POST
    var value3 = postData.value3;  // Récupération de la valeur3 depuis les données POST
    var value4 = postData.value4;  // Récupération de la valeur4 depuis les données POST
    var value5 = postData.value5;  // Récupération de la valeur5 depuis les données POST
    var value6 = postData.value6;  // Récupération de la valeur6 depuis les données POST

    // Mise à jour des données dans la feuille de calcul
    sheet.getRange('B11').setValue(value1);   // Température
    sheet.getRange('B12').setValue(value2);   // Humidité
    sheet.getRange('B13').setValue(value3);   // Qualité d'air
    sheet.getRange('B14').setValue(value4);   // Vitesse
    sheet.getRange('B15').setValue(value5);   // Réservoir
    sheet.getRange('B16').setValue(value6);   // # Requêtes

    return ContentService.createTextOutput("Success").setMimeType(ContentService.MimeType.TEXT);
  }
  catch (error) {
    return ContentService.createTextOutput("Error: " + error.message).setStatusCode(500);
  }
}

function doGet(e) {
  var read = e.parameter.read;

  if (read !== undefined){
    // Lecture des données depuis la feuille de calcul et renvoi au format JSON
    var data = {
      value1: sheet.getRange('B8').getValue(),    // ON OFF aspiration
      value2: sheet.getRange('B9').getValue(),    // +- Puissance graduée
      value3: sheet.getRange('B10').getValue()};  // ON OFF relais

    return ContentService.createTextOutput(JSON.stringify(data)).setMimeType(ContentService.MimeType.JSON);
  }
}

function MoreSpeed() {
  // Augmentation de la vitesse de la cellule B9 de 10 si la valeur actuelle est inférieure à 99
  let counterSS = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Sheet1');
  let counterCell = counterSS.getRange('B9');
  let currentValue = counterCell.getValue();

  if (currentValue < 99) {
    counterCell.setValue(currentValue + 10);
  }
}

function LessSpeed() {
  // Diminution de la vitesse de la cellule B9 de 10 si la valeur actuelle est supérieure à 0
  let counterSS = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Sheet1');
  let counterCell = counterSS.getRange('B9');
  let currentValue = counterCell.getValue();

  if (currentValue > 0) {
    counterCell.setValue(currentValue - 10);
  }
}

function EnableVacuum(){
  // Inversion de la valeur de la cellule B8 et affichage correspondant dans la cellule C8
  let sheet1 = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Sheet1');
  let counterCellB8 = sheet1.getRange('B8');
  let counterCellC8 = sheet1.getRange('C8');

  let currentValue = counterCellB8.getValue();
  counterCellB8.setValue(!currentValue);

  let onOffText = currentValue ? 'OFF' : 'ON';
  counterCellC8.setValue(onOffText);
}

function EnableRelay(){
  // Inversion de la valeur de la cellule B10 et affichage correspondant dans la cellule C10
  let sheet1 = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Sheet1');
  let counterCellB10 = sheet1.getRange('B10');
  let counterCellC10 = sheet1.getRange('C10');

  let currentValue = counterCellB10.getValue();
  counterCellB10.setValue(!(currentValue));

  let onOffText = currentValue ? 'OFF' : 'ON';
  counterCellC10.setValue(onOffText);
}
```

Annexe 2 : Code.ino ESP32

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

const char * ssid = "GSMDavide";          // Nom du réseau WiFi
const char * password = "987654321";   // Mot de passe du réseau WiFi
String GOOGLE_SCRIPT_ID = "AKfycbzjF98lDuM3QgB7KzsosUrOBTz47l20ZqSVrmVz3VsRHXONJzTgpJYeMSJy_aiXGlyk";  // ID du script Google associé
const int sendInterval = 10000;            // Intervalle entre les envois de données en millisecondes
WiFiClientSecure client;                   // Client WiFi sécurisé

int counter;                               // Compteur pour générer des données
unsigned long startTime;                   // Temps de démarrage pour mesurer le temps écoulé

void http_post(int value1, int value2, int value3, int value4, int value5, int value6) {
  HTTPClient http;                         // Création d'un objet HTTPClient
  String url = "https://script.google.com/macros/s/" + GOOGLE_SCRIPT_ID + "/exec";  // Construction de l'URL pour l'envoi POST vers le script Google
  Serial.print("POST ");                   // Affichage dans la console série
  http.begin(url.c_str());                 // Initialisation de la connexion HTTP
  http.addHeader("Content-Type", "application/json");  // Ajout de l'en-tête de contenu JSON
  DynamicJsonDocument jsonDocument(200);  // Création d'un document JSON dynamique
  jsonDocument["value1"] = value1;        // Remplissage du document JSON avec les valeurs à envoyer
  jsonDocument["value2"] = value2;
  jsonDocument["value3"] = value3;
  jsonDocument["value4"] = value4;
  jsonDocument["value5"] = value5;
  jsonDocument["value6"] = value6;
  String postData;                         // Création d'une chaîne de caractères pour stocker les données JSON sérialisées
  serializeJson(jsonDocument, postData);  // Sérialisation du document JSON
  int httpCode = http.POST(postData);   // Envoi des données au script Google via une requête POST
  String payload;                          // Réponse HTTP
  unsigned long elapsedTime = millis() - startTime;  // Calcul du temps écoulé
  if (httpCode > 0) {                      // Si la requête HTTP a réussi
    payload = http.getString();            // Récupération de la réponse HTTP
  } else {
    Serial.println(http.errorToString(httpCode).c_str());  // Affichage des erreurs en cas d'échec
  }
  http.end();                              // Fin de la connexion HTTP
}

void http_get() {
  HTTPClient http;                         // Création d'un objet HTTPClient
  String url = "https://script.google.com/macros/s/" + GOOGLE_SCRIPT_ID + "/exec?read";  // Construction de l'URL pour la requête GET
  Serial.print("GET ");                    // Affichage dans la console série
  http.begin(url.c_str());                 // Initialisation de la connexion HTTP
  http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);  // Suivre strictement les redirections HTTP
  int httpCode = http.GET();             // Envoi de la requête GET au script Google
  String payload;                          // Réponse HTTP
  if (httpCode > 0) {                      // Si la requête HTTP a réussi
    payload = http.getString();            // Récupération de la réponse HTTP
    Serial.print(payload);                // Affichage de la réponse dans la console série
    DynamicJsonDocument jsonDocument(200);  // Création d'un document JSON dynamique
    deserializeJson(jsonDocument, payload);      // Désérialisation de la réponse JSON
    int value1 = jsonDocument["value1"];          // Extraction des valeurs du document JSON
    int value2 = jsonDocument["value2"];
    int value3 = jsonDocument["value3"];
  } else {
    Serial.println("Error on HTTP GET request");  // Affichage des erreurs en cas d'échec
  }
  http.end();                              // Fin de la connexion HTTP
}

void setup() {
  Serial.begin(115200);                    // Initialisation de la communication série
  delay(10);
  WiFi.mode(WIFI_STA);                  // Mode station WiFi
  WiFi.begin(ssid, password);           // Connexion au réseau WiFi
  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED) {  // Attente de la connexion au WiFi
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nStarting");
}

void loop() {
  startTime = millis();                    // Enregistrement du temps de début
  http_post(counter, counter+1, counter-1, counter/2, counter*2, counter%2);  // Appel de la fonction d'envoi POST avec des valeurs générées
  Serial.println("[" + String(millis() - startTime)+ "ms]");  // Affichage du temps écoulé

  startTime = millis();                    // Enregistrement du temps de début
  http_get();                              // Appel de la fonction de requête GET
  Serial.println("[" + String(millis() - startTime)+ "ms]");

  counter ++;
}
```