|  |
| --- |
| Collège Bois-de-Boulogne |
| Rapport final |
| Dans le cadre du cours Objets Connectés |

|  |
| --- |
| Ali Moussa et Olivier Sirois  29/05/2023 |

# Introduction

Le but du projet est de pouvoir concevoir une porte de serre qui peut se gérer elle-même a l’aide de plusieurs petits modules ou avec une intervention humaine. Pour atteindre notre but, il faut que notre système soit connecté à un service infonuagique, dans notre cas, Microsoft Azure. La raison pour laquelle notre projet entre dans le nuage de l’internet des objets et car nous avons comme tâche de créer un petit cerveau qui devrait être capable de décider par soi-même, sans intervention la prochaine action à faire avec les données qu’il récoltera et sauvegardera dans une base de données à distance. Il restera toujours en communication avec Azure et en conséquence avec un utilisateur à distance si nécessaire. Quoique notre programme communique avec une porte que nous avions créée nous-même, le logiciel pourrait très bien être modifier pour supporter une vraie porte de serre car la logique est présente et fonctionnelle.

# Description du matériel

Le matériel que nous avons utilisé est le suivant :

* Raspberry pi
* Thermistor
* Stepping motor
* Stepping motor driver
* ADC Module
* GPIO Extension Board
* Breadboard
* 40 Pin GPIO Cable
* Ultrasonic Ranging Module
* Batterie 9V
* Câble pour batterie 9V
* Jump wires
* Résistances
* Maquette de serre

# Description de l’interface (côté Python)

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

L’interface permet à l’utilisateur de voir les informations en temps réels sur l’état de la serre et la porte. Elle permet aussi de contrôler cette porte avec des boutons et un champ de texte.

Les informations en temps réels sont affichées par les valeurs appropries sous les sections « Informations courantes de la serre » et « Informations courantes sur le moteur ». Le bouton « Tableau de bord » permet de voir un ensemble de données sauvegarder sur la machine locale dans un diagramme à lignes brisées. La barre de progression est vide lorsque la porte est fermée et pleine lorsque ouverte à sa limite.

Les boutons dans la section « Contrôles » permettent de contrôler la porte de la serre. Seul le bouton « Manuelle » a besoin d’une entrée de l’utilisateur pour décider l’ouverture exacte voulu de la porte.

# Description de l’interface (côté Web)

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Les données ne sont pas représentatives

Cette interface fonctionne exactement comme l’interface du côté Python, à un détail près : les informations ne sont pas affichées en temps. En effet, il faut cliquer sur le bouton « Actualiser » pour afficher les informations les plus récentes.

# Description des modules (côté Python)

## TP1.py

Ce module est le cerveau du programme. Il communique avec tous les autres modules. Il cherche les informations nécessaires pour en envoyer d’autres. C’est avec les informations qu’il reçoit du « UltrasonicSensor.py », « Thermistor.py » et du « GUI.py » qu’il en enverra d’autres à « Motor.py ». Le cerveau s’occupe aussi de décider lorsqu'il est temps de recevoir et d’envoyer des données à Azure ainsi qu’à la base de données locale MySQL avec les modules « ReceivedMessageManager.py » et « MySqlConnector.py » respectivement. Il s’occupe aussi d’envoyer les informations en temps réel au module « GUI.py » pour mettre à jour les informations qui seront afficher à l’utilisateur.

## UltrasonicSensor.py

Ce module s’occupe de calculer la distance entre soi-même et la porte pour l’envoyer au cerveau.

## Thermistor.py

Ce module s’occupe de retourner la température courante au cerveau.

## Motor.py

Ce module permet au cerveau de contrôler le moteur. Le cerveau est capable d’accéder aux fonctions qui contrôler le moteur.

## MySqlConnector.py

Ce module s’occupe simplement de sauvegarder les états courants de la serre et de les retourner au besoin.

## GUI.py

Ce module s’occupe d’afficher l’interface avec laquelle l’utilisateur communiquera. Elle présentera des données en temps réel que le cerveau lui enverra, des données que le cerveau aura reçu des modules « UltrasonicSensor.py », « Thermistor.py » et « Motor.py ». Ce module permet aussi d’envoyer des demandes de changements d’états au cerveau, par exemple: passer du mode manuel à automatique. Ce module cherche directement les informations de « MySqlConnector.py » pour afficher le tableau de bord sans passer par le cerveau.

## ReceivedMessageManager.py

Ce module s’occupe de recevoir des évènements envoyer par le contrôle à distance. Elle enverra ce qu’elle reçoit au cerveau s'il y a une requête.

# Description des modules (côté Web)

## HomeController.cs

Ce module sert fait à faire le lien entre l’interface Web et les actions côté serveur. Chaque méthode correspond à une page qu’on peut accéder ou une action qu’on peut effectuer à l’aide d’un ou plusieurs modules.

## TableauDeBordModel.cs et ModeManuelFormModel.cs

Les « Model » sont des classes permettant de transmettre des données du serveur vers l’interface Web. Par exemple, une instance de TableauDeBordModel va contenir toutes les informations à afficher dans la page Tableau de Bord.

## DALKusto.cs

DALKusto sert à aller chercher les données dans la base de données KQL.

## BLLAzureIotManager.cs

Ce module permet d’envoyer les messages à la serre connectée pour qu’elle puisse effectuer l’action demandée.

# Configuration Azure – Partie 1 – Création des ressources Azure

## Création d’un Cluster

1. Tout d’abord, il a fallu créer un Cluster en allant à la [page principale du portail Azure](https://portal.azure.com/#home) et en cliquant sur « Create a ressource ».

Une image contenant capture d’écran, Police, ligne, Tracé

Description générée automatiquement

1. Ensuite j'ai cherché pour “azure data explorer” dans la barre de recherche. Ensuite j'ai mis mon curseur sur « Create » pour ensuite sélectionner “Azure Data Explorer”.

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

1. J’ai créé un « Resource group » au nom de « resourcegroupserre » pour regrouper toutes les composantes nécessaires au fonctionnement du systèmes de télémétrie.

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

1. J’ai continué avec la création du “Azure Data Explorer Cluster” en lui donnant « clusterserre » comme nom. Je choisis aussi la valeur « East US 2 » pour l’option « Region » et « Dev/test » pour « Workload ».

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

1. Une fois présenté avec la page suivante, j’ai cliqué sur « Go to resource ».

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

## Création d'une base de données

1. Une fois sur la page appartenant au Cluster, j’ai cliqué sur « Create » dans la case « Database creation ».

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

1. J’ai donné « databaseserre » comme nom à ma base de données, 365 jours de rétention de données et une période de 31 jours pour la cache. J’ai terminé en pesant sur « Create ».

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

1. Une fois la base de données créer. Je suis allé dans la section « Query » dans le menu à gauche pour pouvoir entrer les commandes nécessaires pour créer une table, ses colonnes et son mapping.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

1. Une fois les commandes roulées, tous les éléments nécessaires au fonctionnement voulu sont créés dans la base de données. Un fichier \*.kql sera jumeler dans un zip avec ce document contenant les commandes.

Une image contenant texte, capture d’écran

Description générée automatiquement

## Création d’une connexion de données IoT Hub

1. Une fois que toutes les commandes ont été roule avec succès, je suis allé dans la section « Databases » dans le menu à gauche et j’ai cliqué sur ma base de données (databaseserre).

Une image contenant texte, capture d’écran, ligne

Description générée automatiquement

1. Encore une fois je vais dans le menu à gauche pour sélectionner la section « Data connections ». Une fois dans la nouvelle page, je rajoute une connexion a un IoT hub que je vais créer en même temps à l’aide du bouton « Add data connection » et « IoT Hub ».

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

1. Dans la page de création d’un IoT Hub je m’assure encore une fois que je l’associe avec le « Resource group » correct. Je lui donne le nom de « iothubserre » et je sélectionne la même valeur pour « Region » que dans les autres étapes (East US 2). Je choisis aussi la valeur « Free » pour l’option « Tier ».

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

1. Une fois l’IoT Hub créer, il faut créer un « Data connection ».  Je lui donne le nom de « dataconnectionserre » et j'entre le nom de mon IoT Hub que je viens de créer (iothubserre) dans le champ « IoT Hub ». Je choisis aussi la valeur « iothubowner » pour l’option « Shared Access Policy » et « $Default » pour « Consumer group ». Ensuite j’ai activé l’option « Allow routing the data to other databases » en m’assurant que l’état de l’interrupteur associer était à « Allow ». Finalement, j’ai inscrit le nom de ma table (tableserre) dans le champ « Table name », choisis le format « JSON » sous « Data format » et choisis le mapping que j’ai créé avec mes commandes KQL (mappingjsonserre) dans le champ « Mapping name ».

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

## Création d’un appareil

1. Je suis retourné dans la page de mon IoT Hub (iothubserre) pour créer un appareil. Pour faire ceci il a fallu cliquer sur « Devices » dans le menu à gauche et ensuite sur le bouton « Add Device » dans la barre d’options.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

1. Dans l’onglet de création d’appareil j’entre le nom voulu qui est « deviceserre », je m’assure que « Authentication type » est sur « Symmetric key », le case à cocher associé à « Auto-generate kets » est coché et que « Connect this device to an IoT hub » est sur « Enable ».

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

1. Une fois l’appareil créer, j’entre dedans en cliquant sur son nom dans la page des appareil (étape 4.a). Je prends en note le « Primary connection string » pour l’utiliser dans mon code. Au niveau de Azure, le travail est maintenant terminé.

Une image contenant texte, capture d’écran, ligne, nombre

Description générée automatiquement

## Vérifier des données entrantes

1. Pour vérifier si les données entrent et qu’elles sont correct, on peut retourner à la page « Query » de la même manière qu'indiquer à l’étape 2.d et rouler la dernière commande du fichier \*.kql. Cette commande affichera toutes les donnes dans notre table « tableserre ».

# Configuration Azure – Partie 2 – Intégration dans le code

Dans le code, nous implémentons la méthode suivante (cette méthode à changer depuis pour y intégrer la synchronisation des bases de données :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Lorsque cette méthode est appelée, elle exécute une première fois son code, puis lance un thread qui exécutera la méthode à chaque 60 secondes. La base du code de cette méthode est inspirée de la documentation donnée par notre enseignant.

# Configuration Azure – Partie 3 – Example de données

Voici un exemple de données sauvegarder dans la base de données Azure concernant un état de la serre.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

# Étapes du contrôle à distance de la porte

Le contrôle à distance de la porte se fait trois étapes.

1. On sélection une action dans l’interface Web. Cela va utiliser le module BLLAzureIotManager qui enverra un message au iot hub d’Azure.
2. L’iot hub d’Azure reçois le message et transmet le message à la serre connecté.
3. Le module ReceivedMessageManager de la serre reçois le message, le découper, puis active la bonne méthode selon le message envoyé.

# Synchronisation des bases de données

Cette synchronisation dans la méthode « communicateAzure ». Si une erreur empêchant d’envoyer des données se produit, ces données sont dans une table de la base de données locale, qui sera ensuite vidée lorsque que la connexion avec Azure sera de nouveau possible

# Tableau de bord (côté Python)

Le tableau de bord du côté Python fonctionne avec les données qui sont insérer dans la base de données locale MySQL. Voici un exemple donné qui sont présentement sauvegarder sur la machine.

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Ces données ont été insérer avec la fonction « saveDataSerre ». Elles seront ensuite récupérées lorsque le bouton « Tableau de bord » sur l’interface sera cliqué à l’aide de la fonction « loadDataSerre ».

Une image contenant capture d’écran

Description générée automatiquement

Les données retourner sont dans un format d’une liste de tuples. Lorsque cette liste est récupérée par la fonction « graph » dans le module « GUI.py », elle sera filtrée pour garder seulement ce qui est essentiel, la température et la date. Une fois la liste filtrée, j’insère ces données dans un dictionnaire qui sera utiliser pour remplir le tableau.

Une image contenant texte, capture d’écran

Description générée automatiquement

Voici un exemple de tableau de bord correspondant aux données vues dans la base de données MySQL.

Une image contenant texte, capture d’écran, ligne, Tracé

Description générée automatiquement

# Tableau de bord (côté Web)

Une image contenant ligne, texte, Tracé, diagramme

Description générée automatiquement

Le tableau de bord de l’interface Web affiche les mêmes informations que celui du côté Python. Par contre, il va chercher les informations dans Azure, au lieu d’une base de données local.

# Conclusion

En conclusion, le projet a été terminer à 95%. Le dernier 5% reste selon nous une limitation de nos connaissances et des technologies choisis. Tout est fonctionnelle du côté logique. Les imprévus qui apparaissent dans les résultats lors des tests dans le monde réels peuvent être facilement attribuer à l’inconsistance du capteur ultrasonique. Nous sommes contents avec le produit final, qui fut d’une évolution incrémentale tout au long de la session.