HermesEnlighteNed

Net and Light

2023 - 2024

Résumé

Cette documentation décrit l'application web Flask intitulée "Flask Web Application for PLC and GPIO Control". Elle comprend une vue d'ensemble, une description des fonctionnalités, une architecture, et des instructions d'installation et d'utilisation.

Table des matières

1 Introduction

Cette documentation présente le script "HermesEnlighteNed.py", qui utilise Flask pour gérer la communication avec des automates programmables logiques (PLC) Siemens et les GPIO d'un Raspberry Pi. Le script permet de lire et d'écrire des données mémoire sur les PLC, ainsi que de contrôler les pins GPIO.

2 Vue d'ensemble du Projet

Le projet "Flask Web Application for PLC and GPIO Control" permet de se connecter à des PLC Siemens, de lire et d'écrire des données mémoire à l'aide de la bibliothèque Snap7. Il utilise également des routes pour contrôler les pins GPIO du Raspberry Pi.

3 Architecture du Système

3.1 Description Globale

L'application est composée des composants suivants :
— **Flask** : Pour créer l'application web.
— **Snap7** : Pour la communication avec les PLC Siemens.
— **RPi.GPIO** : Pour contrôler les pins GPIO du Raspberry Pi.
— **MySQL Connector** : Pour la connexion à la base de données MySQL.

3.2 Composants

Le script est structuré autour de plusieurs routes clés permettant de lire et d'écrire des données sur les PLC et de contrôler les pins GPIO, en plus de gérer la connexion aux PLC.

4 Fonctionnalités

4.1 Configuration

Le script charge la configuration à partir d'un fichier JSON situé à `/home/pi/Desktop/configHermes.json`.

```
# Loading configuration from a JSON file
with open('/home/pi/Desktop/configHermes.json', 'r') as config_file:
    config = json.load(config_file)
```

Listing 1 – Chargement de la configuration

4.2 Initialisation

Le script configure les pins GPIO, le logging, et la connexion à la base de données.

```
# GPIO configuration
GPIO.setmode(GPIO.BCM)
gpio_pins_to_free = [2, 3, 14, 15, 18, 8, 7, 1, 12, 20, 21,
   13, 19, 10, 9, 11]

# Free specific GPIO pins
for pin in gpio_pins_to_free:
    os.system(f"sudo raspi-gpio set {pin} ip")

# Logging configuration
logging.basicConfig(filename=config['logging']['file_path'],
   level=logging.getLevelName(config['logging']['log_level'])
   )

# Database connection setup
conn = connect(
      user=config['Login_Turtle']['user'],
      password=config['Login_Turtle']['password'],
      host=config['Login_Turtle']['host'],
      database=config['Login_Turtle']['database'])
```

Listing 2 – Configuration des GPIO

4.3 Configuration de l'application Flask

Le script initialise l'application Flask et charge des configurations supplémentaires pour la gestion des variables PLC et des GPIO.

```
# Flask app initialization
app = Flask(__name__)

# Loading additional configurations
var_api_out = config['Var_API_Out']
var_api_in = config['Var_API_In']
id_chassis = config['id_chassis']
possibles_modes = config['possibles_modes']
```

Listing 3 – Initialisation de l'application Flask

4.4 Routes

L'application définit plusieurs routes pour gérer et interagir avec les PLC et les pins GPIO.

4.4.1 /connected

Vérifie si le châssis (PLC) est connecté et met à jour son statut.

```python
@app.route('/connected', methods=['GET'])
@synchronized
def connected():
    global CONNECTED_PLC
    try:
        commS7.writeMemory(CONNECTED_PLC,"Mx_master_connecte", "True")
        return jsonify(message="Chassis connecte")
    except Exception as e:
        logging.error("Error chassis non connecte")
        return jsonify(error=str(e)), 500
```

Listing 4 – Route /connected

4.4.2 /check_possible/<string :ip_api>

Essaie de se connecter à un PLC en utilisant l'adresse IP fournie.

```python
@app.route('/check_possible/<string:ip_api>', methods=['GET'])
@synchronized
def check_possible(ip_api):
    global CONNECTED_PLC
    try:
        plc = snap7.client.Client()
        plc.connect(ip_api, 0, 1)
        CONNECTED_PLC = plc
        return jsonify(message="API connecté"), 200
    except Exception as e:
        return jsonify(error=str(e)), 500
```

Listing 5 – Route /check_possible/<string:ip_api>

4.4.3 /multi/<string :BOOL>

Définit si le mode multi est possible sur le PLC.

```python
@app.route('/multi/<string:BOOL>', methods=['GET'])
def multi(BOOL):
    with plc_lock:
        global CONNECTED_PLC
        try:
            commS7.writeMemory(CONNECTED_PLC, "Mx_master_multi_possible", BOOL == "True")
            return jsonify(message="Mode multi activé"), 200
        except Exception as e:
            logging.error(f"Error in /multi: {str(e)}")
            return jsonify(error=str(e)), 500
```

Listing 6 – Route /multi/<string:BOOL>

4.4.4 /multinbr/<string :BOOL>

Définit si le mode multi avec au moins trois châssis est possible.

```
@app.route('/multinbr/<string:BOOL>', methods=['GET'])
def multinbr(BOOL):
    with plc_lock:
        global CONNECTED_PLC
        try:
            commS7.writeMemory(CONNECTED_PLC, "
               Mx_master_multi+de3", BOOL == "True")
            return jsonify(message="Mode multi activé"), 200
        except Exception as e:
            logging.error(f"Error in /multinbr: {str(e)}")
            return jsonify(error=str(e)), 500
```

Listing 7 – Route /multinbr/<string:BOOL>

4.4.5 /trigger/<string :request>

Lit une variable spécifique à partir du PLC.

```
@app.route('/trigger/<string:request>', methods=['GET'])
def trigger(request):
    with plc_lock:
        global CONNECTED_PLC
        try:
            valeur = commS7.readMemory(CONNECTED_PLC, request
               )
            return jsonify(valeur=valeur), 200
        except Exception as e:
            logging.error(f"Error in /trigger: {str(e)}")
            return jsonify(error=str(e)), 500
```

Listing 8 – Route /trigger/<string:request>

4.4.6 /reset_trigger/<string :request>

Réinitialise la variable du PLC spécifiée.

```
@app.route('/reset_trigger/<string:request>', methods=['GET'
   ])
def reset_trigger(request):
    with plc_lock:
        global CONNECTED_PLC
        try:
            commS7.writeMemory(CONNECTED_PLC, request, "False
               ")
            return jsonify(message="Trigger réinitialisé"),
               200
        except Exception as e:
```

```
            logging.error(f"Error in /reset_trigger: {str(e)}
                ")
            return jsonify(error=str(e)), 500
```

Listing 9 – Route /reset_trigger/<string:request>

4.4.7 /pin/<string :position>/<string :couleur>/high

Active une LED spécifique à une position et une couleur données.

```
@app.route('/pin/<string:position>/<string:couleur>/high',
    methods=['GET'])
def pin_HIGH(position, couleur):
    try:
        if position in valid_positions:
            if couleur in color_combinations:
                for color in color_combinations[couleur]:
                    setup_and_activate(position, color, GPIO.
                        HIGH)
                return jsonify(message="Cellule à la position
                    {} et couleur {} a été activée".format(
                    position, couleur)), 200
            else:
                return jsonify(error="Couleur non valide"),
                    400
        else:
            return jsonify(error="Position non valide"), 400
    except Exception as e:
        logging.error(f"Error in /pin: {str(e)}")
        return jsonify(error=str(e)), 500
```

Listing 10 – Route /pin/<string:position>/<string:couleur>/high

4.4.8 /pin/<string :position>/<string :couleur>/low

Désactive une LED spécifique à une position et une couleur données.

```
@app.route('/pin/<string:position>/<string:couleur>/low',
    methods=['GET'])
def pin_LOW(position, couleur):
    try:
        if position in valid_positions:
            if couleur in color_combinations:
                for color in color_combinations[couleur]:
                    setup_and_activate(position, color, GPIO.
                        LOW)
                return jsonify(message="Cellule à la position
                    {} et couleur {} a été désactivée".format
                    (position, couleur)), 200
            else:
```

```
                return jsonify(error="Couleur non valide"), 400
        else:
            return jsonify(error="Position non valide"), 400
    except Exception as e:
        logging.error(f"Error in /pin: {str(e)}")
        return jsonify(error=str(e)), 500
```

Listing 11 – Route /pin/<string:position>/<string:couleur>/low

4.4.9 /reset

Réinitialise tous les pins GPIO à l'état bas.

```
@app.route('/reset', methods=['GET'])
def reset_all():
    try:
        for pos in valid_positions:
            for color in color_list:
                pin_LOW(pos, color)
        return jsonify(message="Tous les GPIOs ont été ré initialisés"), 200
    except Exception as e:
        logging.error(f"Error in /reset: {str(e)}")
        return jsonify(error=str(e)), 500
```

Listing 12 – Route /reset

4.4.10 /reset/<string :position>

Réinitialise les pins GPIO à une position spécifique à l'état bas.

```
@app.route('/reset/<string:position>', methods=['GET'])
def reset(position):
    try:
        if position in valid_positions:
            for color in color_list:
                pin_LOW(position, color)
            return jsonify(message="GPIOs à la position {} ont été réinitialisés".format(position)), 200
        else:
            return jsonify(error="Position non valide"), 400
    except Exception as e:
        logging.error(f"Error in /reset: {str(e)}")
        return jsonify(error=str(e)), 500
```

Listing 13 – Route /reset/<string:position>

4.4.11 /rainbow/<string :position>

Active un effet "arc-en-ciel", en changeant cycliquement les couleurs à une position spécifique.

```
@app.route('/rainbow/<string:position>', methods=['GET'])
def rainbow(position):
    try:
        for color in color_list:
            pin_HIGH(position, color)
            time.sleep(0.5)
        return jsonify(message="Effet arc-en-ciel activé à la
            position {}".format(position)), 200
    except Exception as e:
        logging.error(f"Error in /rainbow: {str(e)}")
        return jsonify(error=str(e)), 500
```

Listing 14 – Route /rainbow/<string:position>

4.4.12 /all/<string :couleur>

Active toutes les positions à une couleur spécifiée.

```
@app.route('/all/<string:couleur>', methods=['GET'])
def all_high(couleur):
    try:
        for pos in valid_positions:
            pin_HIGH(pos, couleur)
        return jsonify(message="Toutes les positions ont été
           activées à la couleur {}".format(couleur)), 200
    except Exception as e:
        logging.error(f"Error in /all: {str(e)}")
        return jsonify(error=str(e)), 500
```

Listing 15 – Route /all/<string:couleur>

4.5 Routes Utilitaires

4.5.1 /kill

Arrête le script et quitte l'application.

```
@app.route('/kill', methods=['GET'])
def kill():
    os.system("sudo kill $(pgrep -f run.py)")
    return jsonify(message="Script arrêté"), 200
```

Listing 16 – Route /kill

5 Gestion des Erreurs

L'application inclut des gestionnaires personnalisés pour les statuts 404 (Not Found) et 500 (Internal Server Error). Ces gestionnaires journalisent les erreurs et retournent une réponse JSON.

```
@app.errorhandler(404)
def not_found(error):
    logging.error(f"404 Error: {error}")
    return jsonify(error="Not found"), 404

@app.errorhandler(500)
def internal_error(error):
    logging.error(f"500 Error: {error}")
    return jsonify(error="Internal server error"), 500
```

Listing 17 – Gestion des erreurs

6 Point d'entrée Principal

Le point d'entrée principal démarre l'application Flask en utilisant l'hôte et le port spécifiés dans le fichier de configuration.

```
if __name__ == '__main__':
    app.run(host=config['host'], port=config['port'])
```

Listing 18 – Point d'entrée principal