lycée professionnel Eugène Decomble

académie Reims
éducation nationale

Présentation de la Ruche Connectée pour le Lycée Eugène Decomble

Professeur : Mr Chaoui Nabil
Élèves : Mr Moussu Esteban et Mr Lamotte Victor

Sommaire

Présentation du projet

Face aux défis croissants de l'apiculture, tels que le changement climatique, les maladies et la diminution des ressources naturelles, … L'intégration des nouvelles technologies offre une opportunité unique pour améliorer la gestion des ruches tout en préservant la santé des abeilles.

Les ruches connectées sont équipées de capteurs intelligents qui permettent aux apiculteurs de surveiller en temps réel plusieurs paramètres essentiels comme la température, l'humidité, le poids de la ruche et d'estimer l'activité des abeilles.

Grâce à ces données, ils peuvent anticiper les risques et intervenir au bon moment, limitant ainsi les pertes et optimisant la production de miel. Ce projet vise à explorer l'impact des ruches connectées sur le travail des apiculteurs, en mettant en lumière leurs avantages en termes de surveillance, de prévention des maladies et d'optimisation des récoltes.

À travers cette approche technologique, l'apiculture peut se transformer en un secteur plus durable et efficace, où la science et la nature avancent main dans la main.

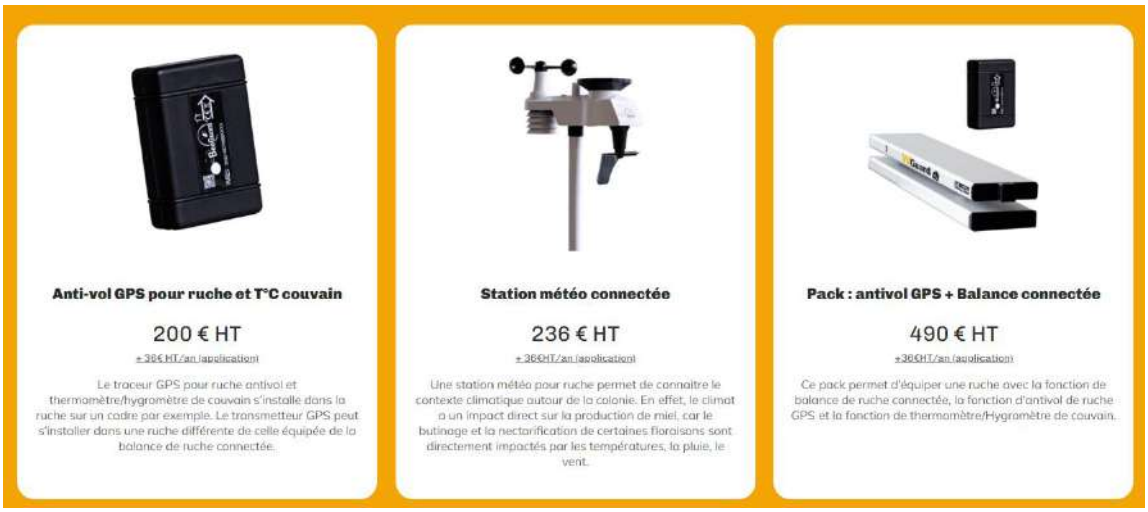
Plus économique pour l’apiculteur ?

Devis de l’électronique d’une ruche :

Composant	Qté	Prix unitaire (€)	Total (€)
Panneau solaire 12 V	1	16.63	16.63
Batterie plomb 12 V	1	15.5	15.5
Contrôleur de charge 12 V (Chine)	1	5.0	5.0
Raspberry Pi 4 B (2 à 4 Go)	1	50.0	50.0
HX711 + cellules de charge (x4)	4	14.0	56.0
Clé 4G LTE D-Link DWM-222	1	59.0	59.0
Caméra pour Raspberry Pi	1	7.99	7.99
Capteur DHT22 (achat en lot)	1	3.0	3.0

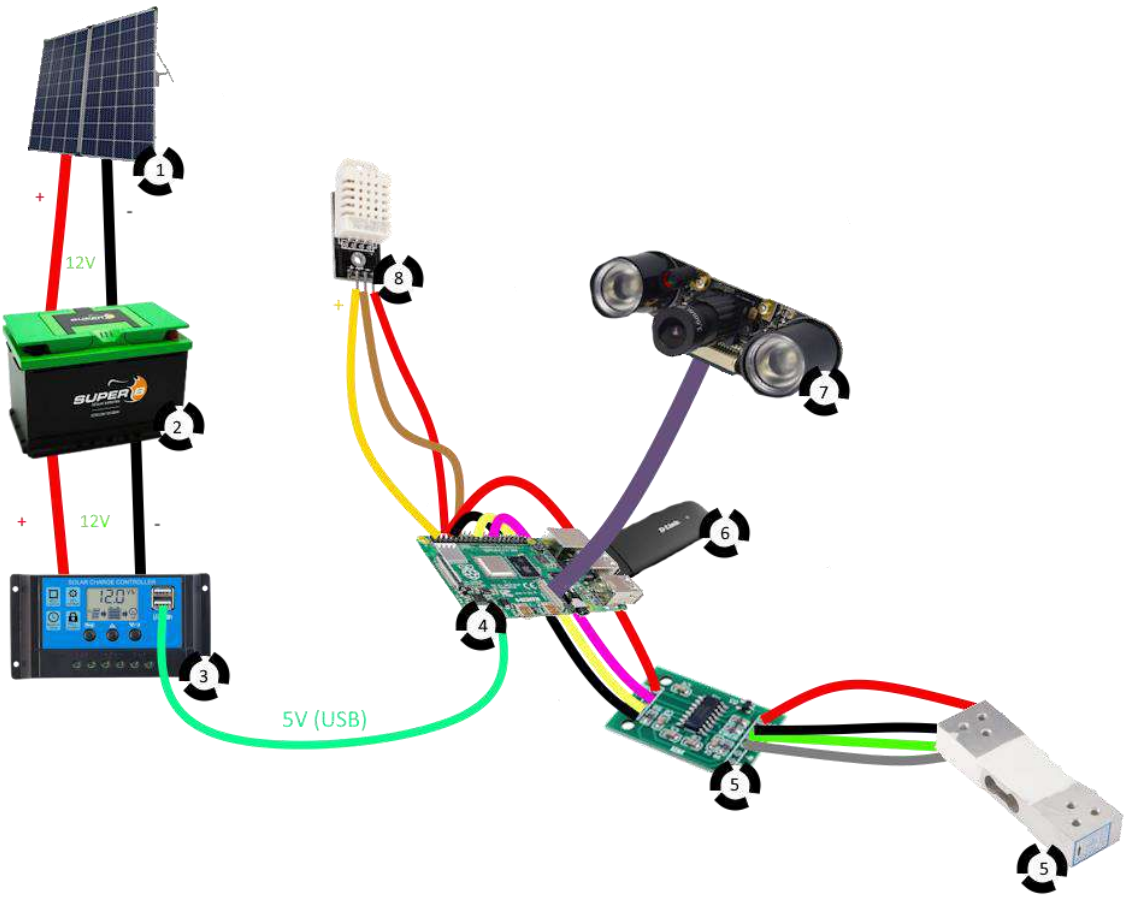
Total final par ruche : 213.12 €

Alors que l’équipement d’une ruche chez d’autres vendeurs pourrait coûter jusqu’à 600€ HTC (sans caméra en plus).



(https://www.beeguard.fr/apiculture)

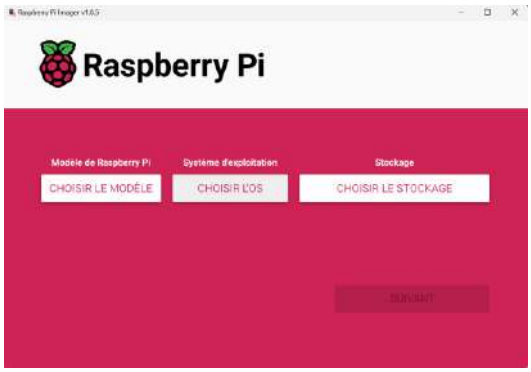
Schéma d’installation de notre ruche connectée



1. Panneau solaire
2. Batterie 12V
3. Contrôleur de batterie
4. Raspberry pi 4b
5. Module + Balance
6. Clé wifi
7. Caméra pour Raspberry pi
8. Capteur DHT22

Déploiement du système d'exploitation sur la ruche

En premier temps, nous avons installé le système d’exploitation de notre Raspberry Pi sur une carte SD grâce au logiciel Raspberry Pi Imager.

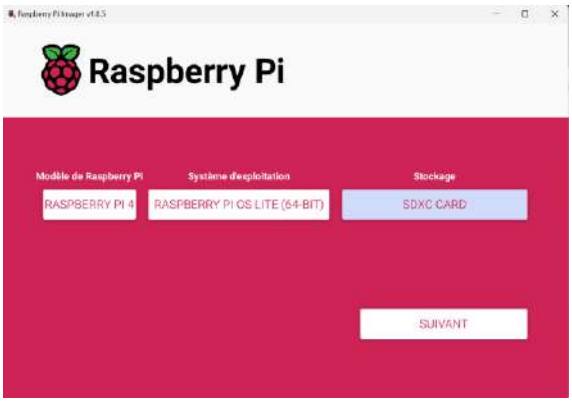


Le système d’exploitation que l’on va installer sera debian (Raspberry Pi OS) qui est un système d’exploitation Linux basé sur Debian.
Nous avons choisi Linux pour sa facilité en programmation et en installation de programme.
Pour ce faire, on doit choisir notre modèle de RaspberryPi parmi les Raspberry zero, 2, 3, 4 et 5.

Pour notre ruche nous avons un Raspberry 4 : C’est un bon compromis pour un ordinateur car il consomme très peu et nous permet de connecter des capteurs très facilement.

Ensuite sur Raspberry Pi Imager nous devons choisir le système d’exploitation, comme dis précédemment nous allons choisir debian Lite (une expérience de système d’exploitation sans bureau et uniquement en ligne de commande).

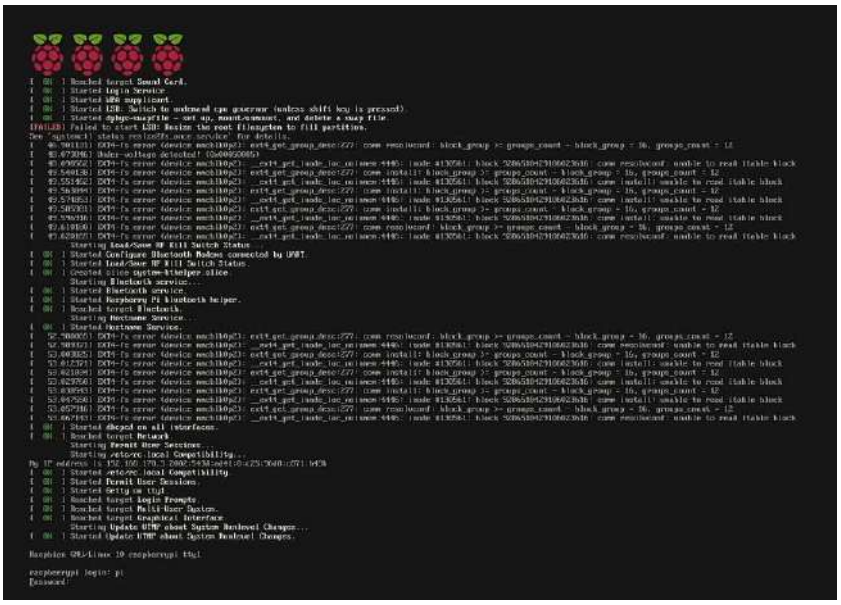
Puis nous allons choisir notre média où nous allons installer debian .



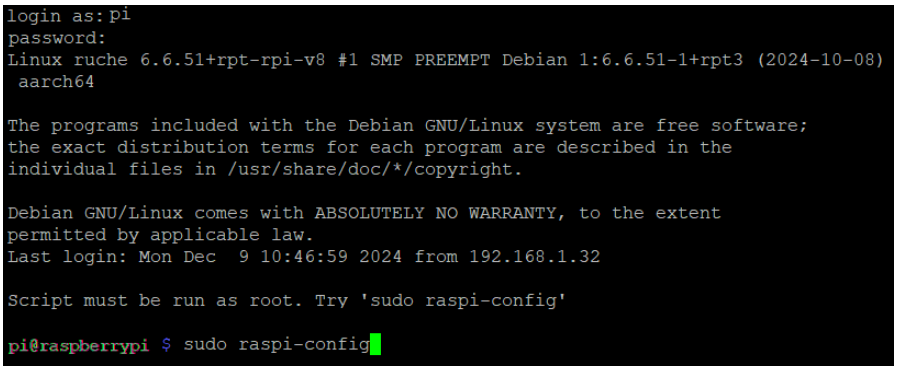
Dès que nous avons appuyé sur Suivant, l'installation du Système d'exploitation va débuter et dès que Raspberry Pi Imager aura fini, nous allons avoir un message de confirmation que le système à bien été installé.

Pour continuer, nous allons mettre la carte SD dans notre Raspberry Pi 4 puis brancher un moniteur en Micro-HDMI puis un câble d'alimentation en USB-C (de préférence l'officiel).
Nous aurons aussi besoin d'un clavier et d'une souris au démarrage.

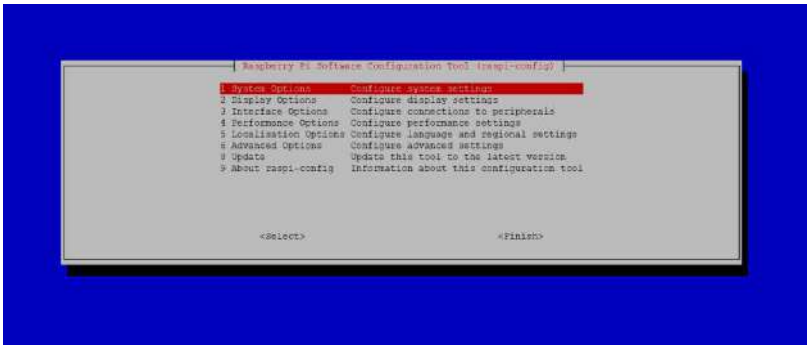
Lors du démarrage, le Raspberry Pi redémarre plusieurs fois avant de réellement démarrer puis va nous demander un nom d'utilisateur et un mot de passe. Le nom d'utilisateur par défaut pour debian est "pi" et le mot de passe est "raspberry" (nous le changerons plus tard), le clavier sera par défaut en Qwerty donc pour le mot de passe il faudra taper "rqspberry".



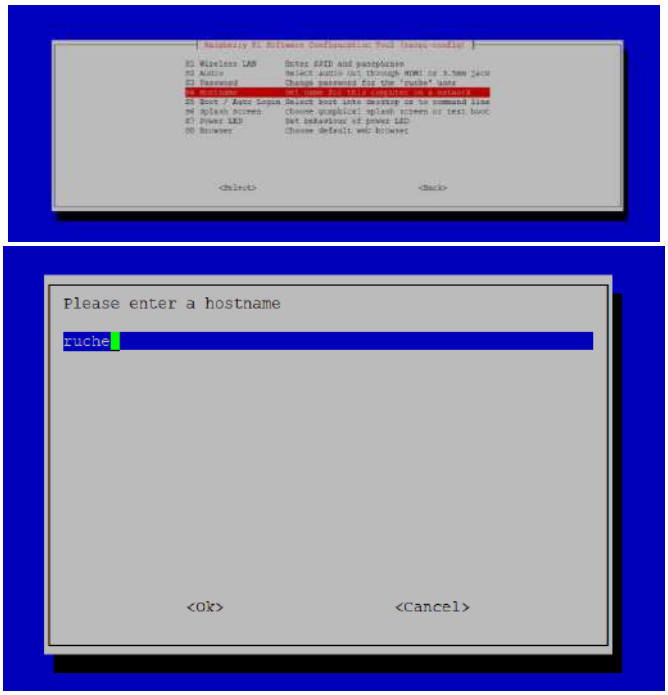
Dès cela fait nous allons aller dans la configuration de notre Raspberry PI grâce à la commande “sudo raspi-config”.



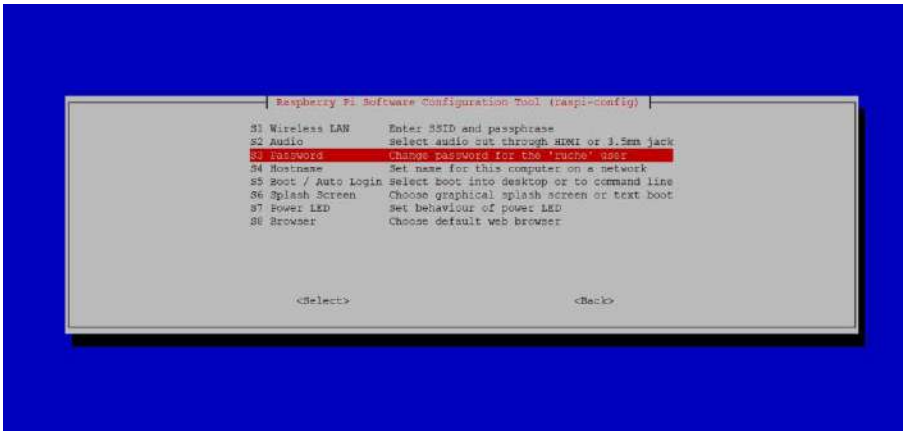
Dès que nous aurons fait cette commande, une page bleue va s’afficher avec plusieurs propositions, nous irons dans “System Options”.



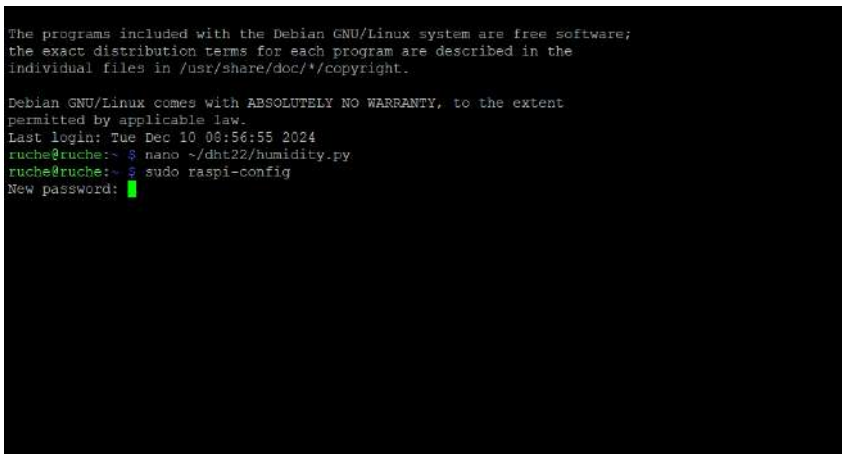
Puis nous irons dans Hostname, puis nous changerons “RaspberryPi” par “ruche”.



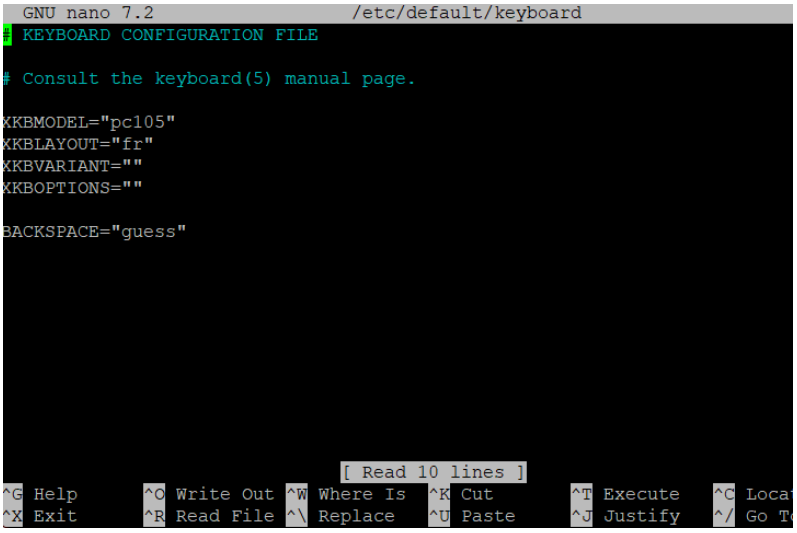
On met Ok puis on va dans Password.



Puis il va nous demander un nouveau mot de passe, nous allons mettre “ruche5250”.

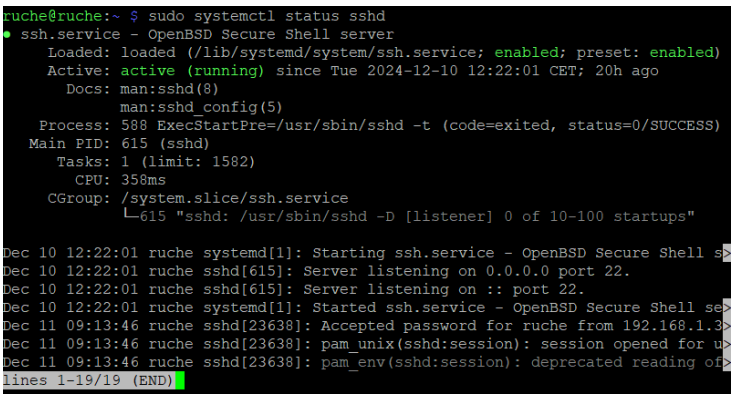


Ensuite il faudra passer le clavier QWERTY vers AZERTY, pour ce faire, on tape la commande “sudo nano /etc/default/keyboard” puis par défaut il y a aura “gb” donc on remplace par “fr” puis on fait CTRL et X pour sauvegarder puis “y”.



Ensuite, nous allons installer OpenSSH- sur notre debian , ce module va nous permettre de prendre le Raspberry Pi sans avoir d’écran à distance : ça nous permettra également de travailler depuis notre poste de travail sans avoir à se déplacer.

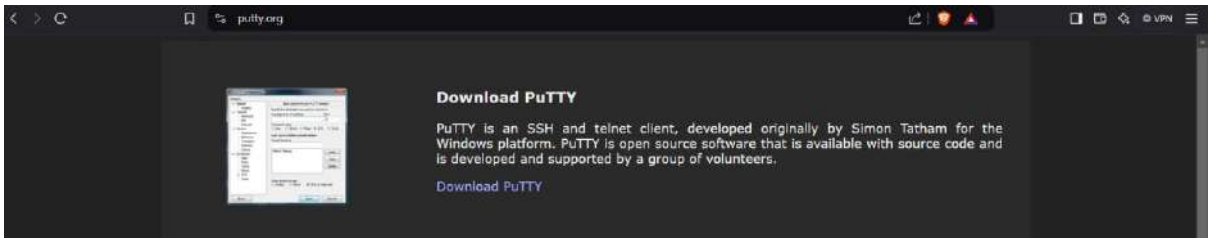
Pour ce faire nous allons faire “sudo apt-get update” puis par la suite “sudo apt-get install openssh-”, dès que celui ci est installé nous pouvons faire “sudo systemctl status sshd” pour voir son bon fonctionnement : S’il n’est pas actif, nous aurions pu faire “sudo systemctl restart sshd”.



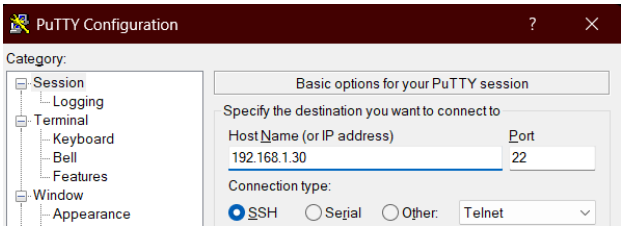
Puis nous pouvons voir l’ip de notre Raspberry grâce à “hostname -I”, notre Raspberry est actuellement en RJ45 donc pour le moment pas besoin de configurer une connexion sans fil.

```
ruche@ruche:~ $ hostname -I
192.168.1.30 172.17.0.1 2a01:cb0d:59b:9800:bcca:bad1:7dde:1b1f
ruche@ruche:~ $
```

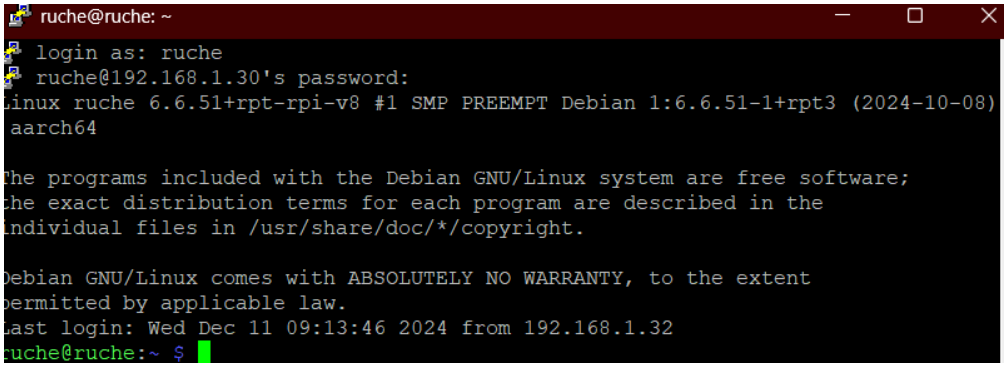
Maintenant que nous avons l'ip du Raspberry PI, nous devons aller sur l'ordinateur nous connecter au même réseau que le Raspberry PI puis nous allons installer Putty (sur putty.org) qui est un logiciel de contrôle à distance SSH en lignes de commandes.



Maintenant que nous avons installé le logiciel, nous devons l'ouvrir et taper l'ip que la commande précédente nous a donné, puis on laisse coché "SSH".



Après avoir fait "Open" nous allons arriver sur une interface en ligne de commande semblable à notre Raspberry Pi relié à un écran, or nous sommes en contrôle à distance (SSH). Nous devons désormais donner nos identifiants, dans notre cas "ruche" et "ruche5250" changé précédemment.

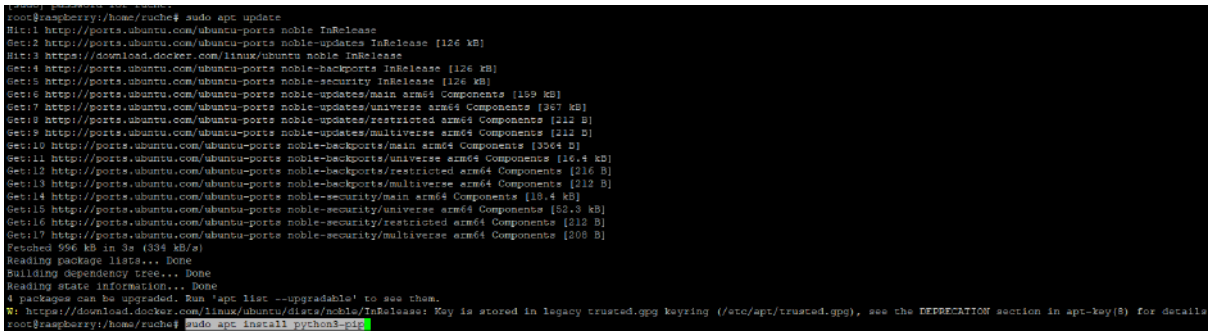


Nous sommes à présent sur notre ruche à partir d'un ordinateur distant.

Maintenant, nous allons passer en “root” c’est à dire en administrateur grâce à la commande “sudo su”, ensuite debian va nous demander le mot de passe administrateur, puis on peut voir que maintenant il y a inscrit “root@raspberry:/home/ruche#”.

```
python3-netplan/noble-updates 1.1.2-2~ubuntu24.04.1 arm64 [upgradable from: 1.1.
1-1~ubuntu24.04.1]
ruche@raspberry:~$ sudo su
[sudo] password for ruche:
root@raspberry:/home/ruche#
```

Désormais, nous allons installer python ainsi que son environnement virtuelle pythonvenv, nous allons commencer par un “sudo apt update” et ensuite pour installer python “sudo apt install python3-pip”



Puis nous allons installer l’environnement python grâce à “sudo apt install python3-venv” dans notre console.

```
root@raspberry:/home/ruche# sudo apt install python3-venv
```

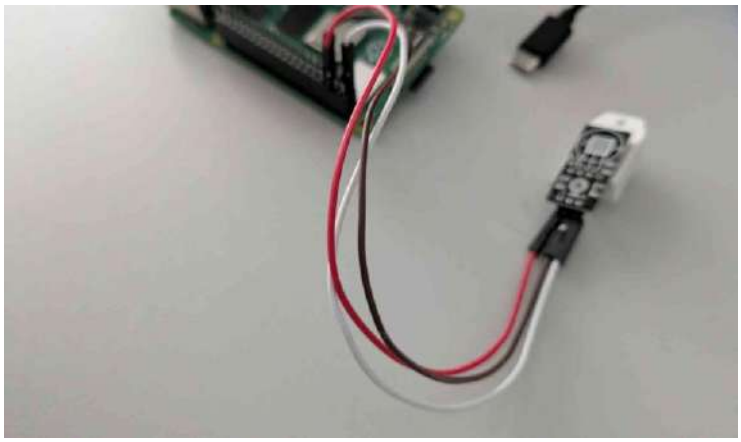
Par la suite on fait “python3 -m venv env” et puis “source env/bin/activate”.

```
root@raspberry:/home/ruche# python3 -m venv env
```

```
root@raspberry:/home/ruche# source env/bin/activate
(env) root@raspberry:/home/ruche#
```

On peut y voir qu’un “(env)” est apparu devant notre “root” ce qui veut donc dire que nous sommes en environnement python.

Nous allons donc commencer notre script python pour la température et l’humidité, il faudra donc connecter le capteur DHT22 aux pins du Raspberry comme ici :



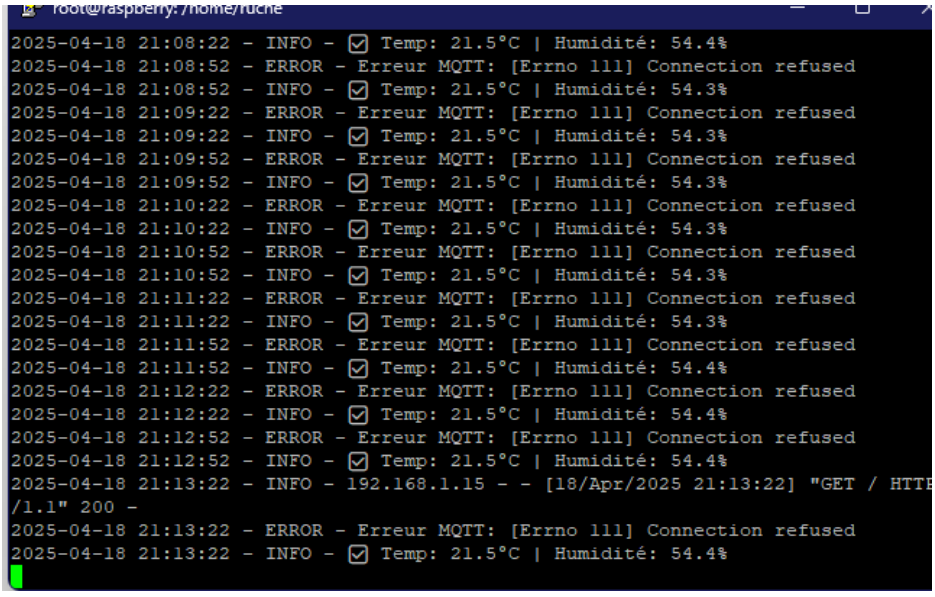
Pour les branchements, nous récupérerons les données sur le GPIO 4.

Puis nous allons commencer notre script “dht22_v1.py”

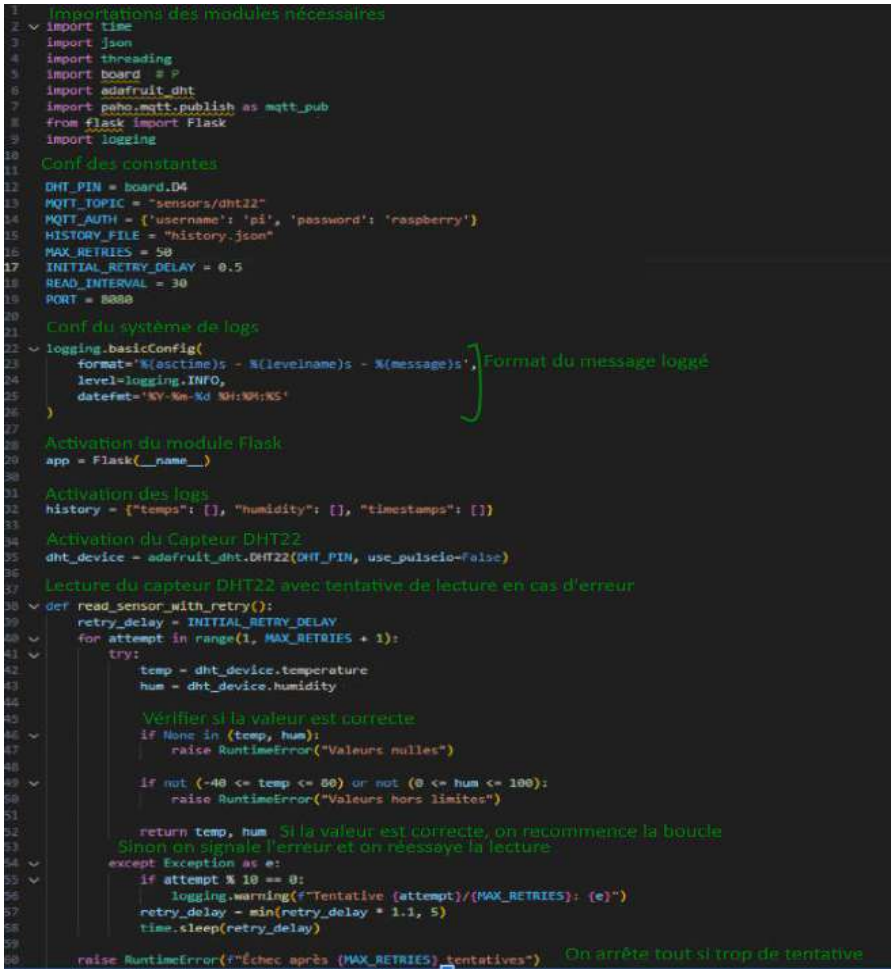
```
import time  Pour gérer les délais et les temporisations
import board  Accès aux broches GPIO (Raspberry Pi)
import adafruit_dht  Pilote pour le capteur DHT22
import logging  Gestion des logs (enregistrement des événements)

DHT_PIN = board.D4  Broche GPIO utilisée (ici GPIO4)
RETRY_DELAY = 2.5  Délai entre les tentatives de lecture (en secondes)
MAX_RETRIES = 5  Nombre max de tentatives en cas d'échec
READ_INTERVAL = 30 Intervalle entre les lectures (en secondes)

logging.basicConfig(  Appel d'une fonction
    format='%(asctime)s - %(levelname)s - %(message)s',  Format des messages
    level=logging.INFO,  Niveau de log (INFO, WARNING, ERROR, etc.)
    datefmt='%Y-%m-%d %H:%M:%S'  Format de la date dans les logs
)

dht_device = adafruit_dht.DHT22(DHT_PIN, use_pulseio=False) Crée un objet pour communiquer avec le capteur puis
                                                            `use_pulseio=False` désactive une fonctionnalité problématique
                                                            sur certains systèmes.
```

```python
def read_sensor():
    for attempt in range(1, MAX_RETRIES + 1):  Boucle de tentative
        try:
            temperature = dht_device.temperature  Lit la temperature
            humidity = dht_device.humidity        et l'humidité

            if None in (temperature, humidity):    Si valeur est nulle
                raise RuntimeError("Valeurs nulles")

            if not (0 <= humidity <= 100) or not (-40 <= temperature <= 80):  Plages de temperature maximale
                raise RuntimeError("Valeurs hors limites")                    du DHT22

            return temperature, humidity   Retourne les valeurs si OK

        except RuntimeError as e:
            logging.warning(f"Tentative {attempt}/{MAX_RETRIES}: {e}")  Gestion des tentatives
            if attempt < MAX_RETRIES:
                time.sleep(RETRY_DELAY)   Attend avant chaque nouvelle tentaative

    raise RuntimeError(f"Échec après {MAX_RETRIES} tentatives")  Echec finale en cas de non réponse du capteur

def main():
    logging.info("Démarrage du moniteur DHT22")  Log de démarrage
    while True:  Boucle infinie
        try:
            start_time = time.time() Collecte l'heure du début de la collecte de donnée

            try:
                temp, hum = read_sensor()
                logging.info(f"✅ Temp: {temp:.1f}°C | Humidité: {hum:.1f}%") Logs des valeurs
            except Exception as e:
                logging.error(f"❌ Échec: {e}")  Echec avec la log de l'erreur

            elapsed = time.time() - start_time    Temps écoulé par tentative
            if elapsed < READ_INTERVAL:     Attend le prochain intervale
                time.sleep(READ_INTERVAL - elapsed)

        except KeyboardInterrupt:   Gestion de 'l'arret manuel avec CTRL+C
            logging.info("Arrêt manuel")
            break

if __name__ == "__main__":
    main()
```

Puis lorsqu'on va lancer notre script nous aurons ça :



Ça fonctionne mais maintenant, nous voulons un site hébergé en local sur le raspberry qui nous ferra des graphiques.
Pour ce faire nous allons créer “dht22_v2.py” :



```python
# Importations des modules nécessaires
import time
import json
import threading
import board  # P
import adafruit_dht
import paho.mqtt.publish as mqtt_pub
from flask import Flask
import logging

# Conf des constantes
DHT_PIN = board.D4
MQTT_TOPIC = "sensors/dht22"
MQTT_AUTH = {'username': 'pi', 'password': 'raspberry'}
HISTORY_FILE = "history.json"
MAX_RETRIES = 50
INITIAL_RETRY_DELAY = 0.5
READ_INTERVAL = 30
PORT = 8888

# Conf du système de logs
logging.basicConfig(
    format='%(asctime)s - %(levelname)s - %(message)s',  # Format du message loggé
    level=logging.INFO,
    datefmt='%Y-%m-%d %H:%M:%S'
)

# Activation du module Flask
app = Flask(__name__)

# Activation des logs
history = {"temps": [], "humidity": [], "timestamps": []}

# Activation du Capteur DHT22
dht_device = adafruit_dht.DHT22(DHT_PIN, use_pulseio=False)

# Lecture du capteur DHT22 avec tentative de lecture en cas d'erreur
def read_sensor_with_retry():
    retry_delay = INITIAL_RETRY_DELAY
    for attempt in range(1, MAX_RETRIES + 1):
        try:
            temp = dht_device.temperature
            hum = dht_device.humidity

            # Vérifier si la valeur est correcte
            if None in (temp, hum):
                raise RuntimeError("Valeurs nulles")

            if not (-40 <= temp <= 80) or not (0 <= hum <= 100):
                raise RuntimeError("Valeurs hors limites")

            return temp, hum  # Si la valeur est correcte, on recommence la boucle
        # Sinon on signale l'erreur et on réessaye la lecture
        except Exception as e:
            if attempt % 10 == 0:
                logging.warning(f"Tentative {attempt}/{MAX_RETRIES}: {e}")
            retry_delay = min(retry_delay * 1.1, 5)
            time.sleep(retry_delay)

    raise RuntimeError(f"Échec après {MAX_RETRIES} tentatives")  # On arrête tout si trop de tentative
```

```python
# Création d'une fonction pour la lecture du capteur DHT22
def sensor_loop():
    consecutive_failures = 0  # Compteur d'echecs
    while True:
        try:
            start_time = time.time()

            try:
                temp, hum = read_sensor_with_retry()
                consecutive_failures = 0

                try:
                    # Envoie des données à MQTT
                    mqtt_pub.single(
                        MQTT_TOPIC,
                        f"{temp},{hum}",
                        hostname="localhost",
                        auth=MQTT_AUTH
                    )
                except Exception as e:                    # Logs des erreurs MQTT
                    logging.error(f"Erreur MQTT: {str(e)}")

                # Mise à jour de l'historique des données grâce au temps
                history['temps'].append(temp)
                history['humidity'].append(hum)
                history['timestamps'].append(time.time())

                # Limite max de l'historique
                if len(history['timestamps']) > 86400 // READ_INTERVAL:
                    for key in history:
                        history[key].pop(0)

                # Sauvegarde de l'historique dans un fichier json
                with open(HISTORY_FILE, 'w') as f:
                    json.dump(history, f)

                logging.info(f"✅ Temp: {temp:.1f}°C | Humidité: {hum:.1f}%")  # Log des données lues
            # Si erreur
            except Exception as e:
                consecutive_failures += 1  # On ajoute +1 au nombre d'erreurs
                logging.error(f"❌ Erreur: {e}")  # Et on log
                if consecutive_failures >= 3:
                    logging.critical("3 échecs consécutifs - Pause de 60s")
                    time.sleep(60)
                    consecutive_failures = 0

            elapsed = time.time() - start_time
            if elapsed < READ_INTERVAL:
                time.sleep(READ_INTERVAL - elapsed)
        # Interruption manuelle = Arrêt de la boucle
        except KeyboardInterrupt:
            break
```

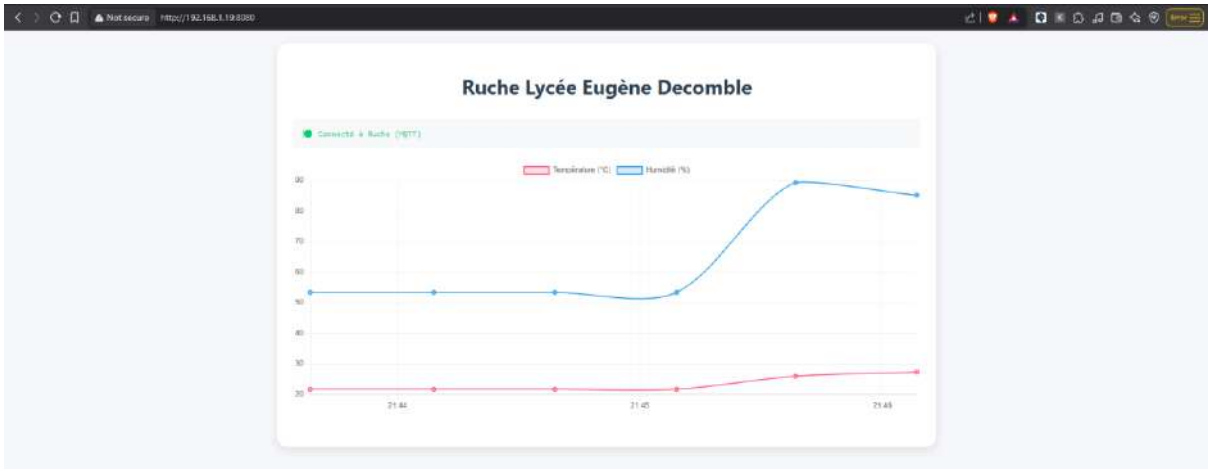
```python
# Le code du site web se déroule entre la ligne 114 et 298, nous allons masquer cette partie car c'est inutile en documents
if __name__ == '__main__':
    sensor_thread = threading.Thread(target=sensor_loop)  # On creer un État pour le capteur DHT22
    sensor_thread.daemon = True  # L'état s'arrête quand le programme s'arrête
    sensor_thread.start()  # Démarrage de l'État
    app.run(host='0.0.0.0', port=PORT)  # On démarre le site web grâce au module Flask
```

Il nous faudra également un serveur MQTT donc nous allons l'héberger localement, pour faire ça, nous devons faire “sudo apt update” puis “sudo apt install mosquitto mosquitto-clients” pour voir si notre serveur MQTT est correctement installé nous devons faire “sudo systemctl start mosquitto” puis “sudo systemctl enable mosquitto” et ensuite “sudo systemctl status mosquitto”.

```
● mosquitto.service - Mosquitto MQTT Broker
     Loaded: loaded (/usr/lib/systemd/system/mosquitto.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-04-28 10:35:46 CEST; 32min ago
       Docs: man:mosquitto.conf(5)
             man:mosquitto(8)
    Process: 817 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
    Process: 823 ExecStartPre=/bin/chown mosquitto:mosquitto /var/log/mosquitto (code=exited, status=0/SUCCESS)
    Process: 836 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/SUCCESS)
    Process: 873 ExecStartPre=/bin/chown mosquitto:mosquitto /run/mosquitto (code=exited, status=0/SUCCESS)
   Main PID: 882 (mosquitto)
      Tasks: 1 (limit: 1519)
     Memory: 3.4M (peak: 3.6M)
        CPU: 2.571s
     CGroup: /system.slice/mosquitto.service
             └─882 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Apr 28 10:35:45 raspberry systemd[1]: Starting mosquitto.service - Mosquitto MQTT Broker...
Apr 28 10:35:45 raspberry mosquitto[882]: 1745829345: Loading config file /etc/mosquitto/conf.d/auth.conf
Apr 28 10:35:45 raspberry mosquitto[882]: 1745829345: Loading config file /etc/mosquitto/conf.d/default.conf
Apr 28 10:35:45 raspberry mosquitto[882]: 1745829345: Loading config file /etc/mosquitto/conf.d/websockets.conf
Apr 28 10:35:46 raspberry systemd[1]: Started mosquitto.service - Mosquitto MQTT Broker.
```

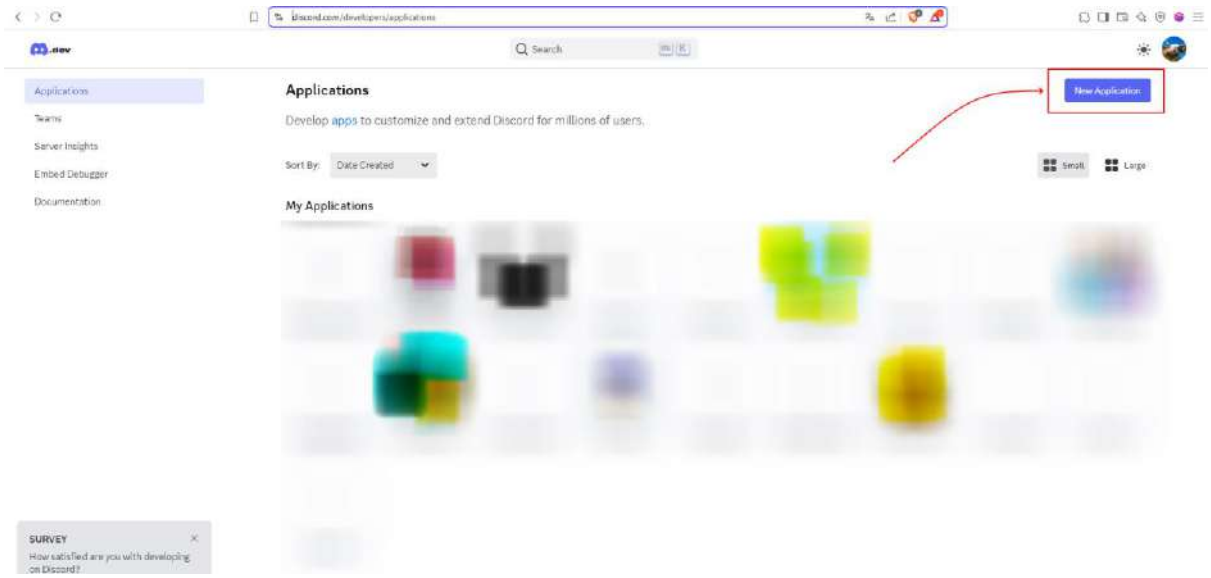
On peut donc voir que MQTT est activé (enabled) et qu'il est en cours de fonctionnement (active [running]).

Finalement nous pouvons donc désormais lancer notre script en environnement python et nous rendre sur 192.168.0.153:8080 et visualiser cela :

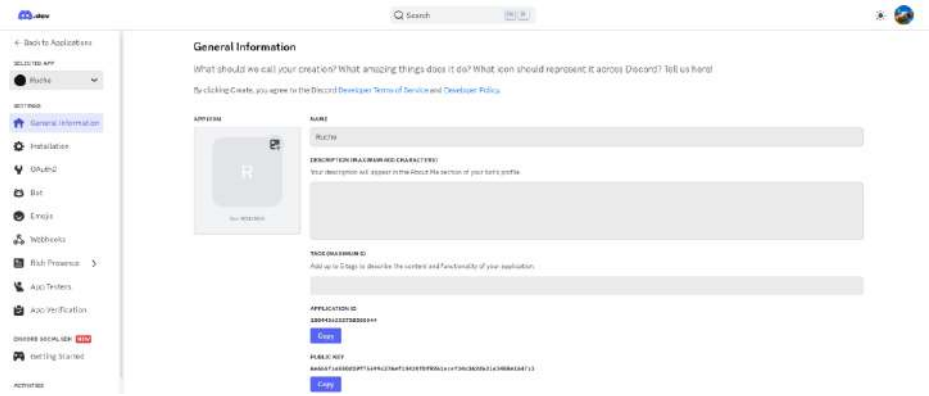


Nous allons également créer un bot discord pour récupérer les données en format tchat ce qui va permettre au lycée d'avoir les données de toutes les ruches en même temps.

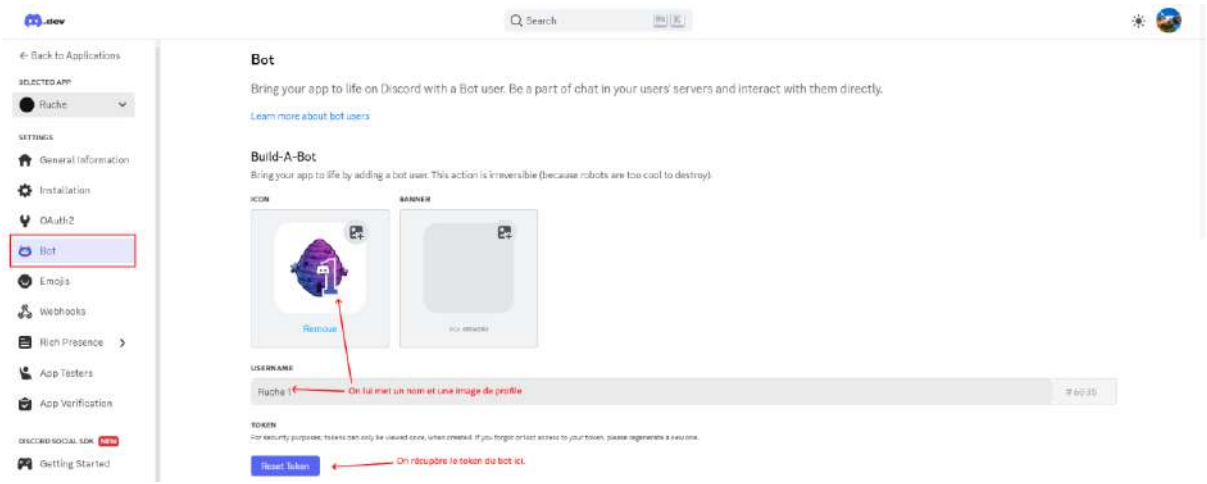
Je dois donc créer une application sur Discord Developers, je me rends donc sur le site "discord.com/developers/applications" puis sur "New Application".



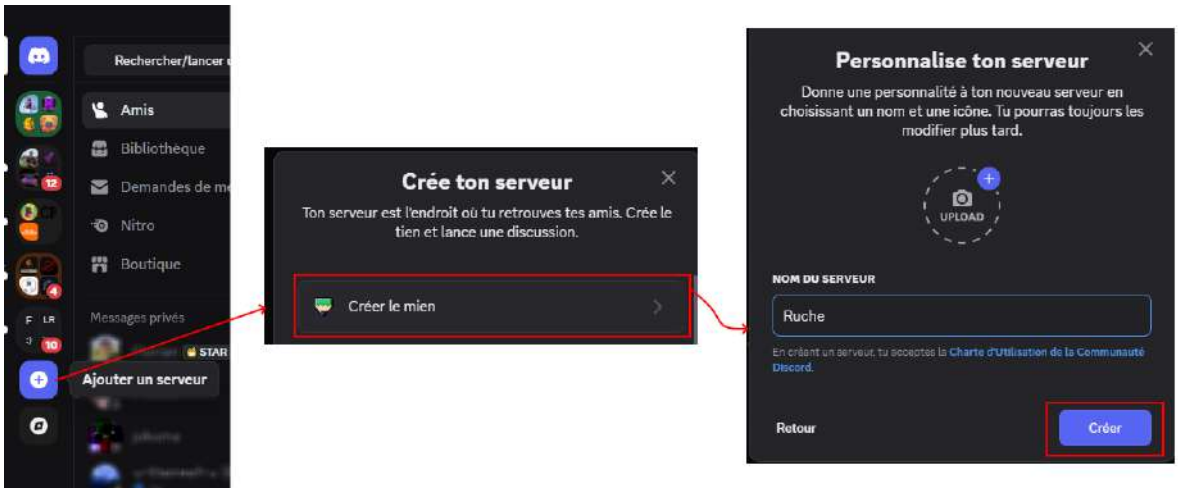
Ensuite après avoir créer notre Application que nous allons nommer “Ruche”.



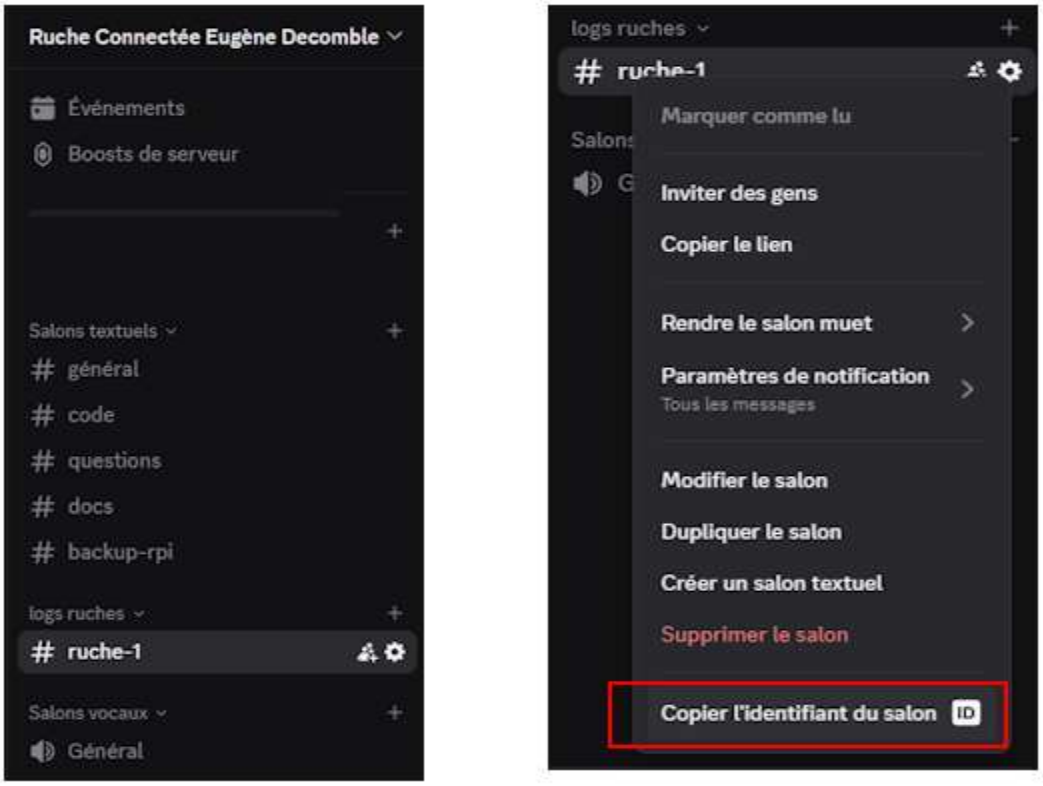
Nous allons nous rendre dans “Bot” et créer notre Bot discord.



Après lui avoir mis un nom, une photo de profil, et avoir récupéré le token du bot, nous allons devoir créer un serveur discord, on se rend donc sur discord, puis sur “+” et “Ajouter un serveur”.



Puis nous allons créer des salons textuels qui vont nous servir pour la ruche ainsi que pour les élèves qui vont reprendre le projet l'année prochaine. On n'oublie pas de récupérer l'identifiant du salon Ruche-1 qui va nous permettre de garder un historique sur discord les données de notre première ruche.



Nous allons maintenant coder en python le serveur du bot discord, le code est trop long pour être inscrit dans ce document, donc nous allons mettre les parties essentielles.

Paramètres généraux :

```
DISCORD_TOKEN = [redacted]
CHANNEL_ID = 1384461989463720058
HISTORY_FILE = "historique_bot.json"
```

Paramètres MQTT :

```
MQTT_BROKER = "localhost"
MQTT_PORT = 1883
MQTT_TOPIC = "ruche/connectee/donnees"
MQTT_USER = "pi"
MQTT_PASS = "raspberry"
```

Connexion à MQTT et récupération des données du capteur :

```
def on_message(client, userdata, msg):
    try:
        payload = msg.payload.decode()
        temp, hum = map(float, payload.split(','))
        timestamp = datetime.now()


        sensor_data["temperature"].append(temp)
        sensor_data["humidite"].append(hum)
        sensor_data["timestamps"].append(timestamp.isoformat())


        if DEBUG_MODE:
            log(f"Reçu: {temp:.1f}°C, {hum:.1f}%")


        if len(sensor_data["timestamps"]) % (SAVE_INTERVAL // 5) == 0:
            save_data()

    except Exception as e:
        log(f"Erreur MQTT: {e}")

def on_connect(client, userdata, flags, rc):
    log(f"Connecté au broker MQTT avec le code {rc}")
    client.subscribe(MQTT_TOPIC)
```

Automatisation des envoies de données sur le discord :

```python
@tasks.loop(seconds=REPORT_INTERVAL)
async def send_periodic_report():
    """Envoi périodique du rapport de moyenne"""
    global last_message_time


    avg_temp, avg_hum, count = calculate_average(AVG_PERIOD)

    if avg_temp is None:
        log("Aucune donnée disponible pour le rapport")
        return


    now = datetime.now()
    start_time = now - timedelta(seconds=AVG_PERIOD)
    time_range = f"{start_time.strftime('%H:%M:%S')} - {now.strftime('%H:%M:%S')}"

    unit = "secondes" if AVG_PERIOD < 60 else "minutes"
    period_value = AVG_PERIOD if AVG_PERIOD < 60 else AVG_PERIOD // 60


    channel = bot.get_channel(CHANNEL_ID)
    if channel:
        try:
            await channel.send(
                f"📊 **Moyenne des {period_value} {unit}** ({time_range})\n"
                f"🌡️ Température: **{avg_temp:.1f}°C**\n"
                f"💧 Humidité: **{avg_hum:.1f}%**\n"
                f"Nombre de relevés: {count}"
            )
            last_message_time = now
            log(f"Rapport envoyé: {avg_temp:.1f}°C, {avg_hum:.1f}%")
        except Exception as e:
            log(f"Erreur d'envoi Discord: {e}")
    else:
        log(f"Canal avec ID {CHANNEL_ID} introuvable")

@tasks.loop(seconds=SAVE_INTERVAL)
async def periodic_save():
    """Sauvegarde périodique des données"""
    save_data()
```

Ajouts des commandes sur le discord tel que “!ruche” pour avoir des informations manuellement, !debug pour savoir si le mode debug est activé, et !stats pour avoir des informations sur une période voulue :

```python
C: > Users > PC ESTEBAN > Documents > Ruche > bot.py > ruche_stats
@bot.command(name='ruche')
async def ruche_stats(ctx):
    """Affiche les dernières mesures"""
    if not sensor_data["timestamps"]:
        await ctx.send("Aucune donnée disponible")
        return

    last_temp = sensor_data["temperature"][-1]
    last_hum = sensor_data["humidite"][-1]
    last_time = datetime.fromisoformat(sensor_data["timestamps"][-1]).strftime("%H:%M:%S")

    report_info = ""
    if last_message_time:
        elapsed = int((datetime.now() - last_message_time).total_seconds())
        report_info = f"\n Dernier rapport: il y a {elapsed}s"

    await ctx.send(
        f" **Statut actuel de la ruche**\n"
        f" Dernière température: **{last_temp:.1f}°C**\n"
        f" Dernière humidité: **{last_hum:.1f}%**\n"
        f" Dernier relevé: {last_time}"
        f"{report_info}"
    )

@bot.command(name='debug')
async def toggle_debug(ctx):
    """Active/désactive le mode debug (nécessite redémarrage)"""
    await ctx.send(
        f" **Mode debug**\n"
        f"Actuellement: {'ACTIVÉ' if DEBUG_MODE else 'DÉSACTIVÉ'}\n"
        f"Pour changer, modifiez la variable DEBUG_MODE dans le code et redémarrez le bot."
    )

@bot.command(name='stats')
async def system_stats(ctx):
    """Affiche les statistiques du système"""
    uptime = int(time.time() - start_time)
    hours, remainder = divmod(uptime, 3600)
    minutes, seconds = divmod(remainder, 60)

    await ctx.send(
        f" **Statistiques du système**\n"
        f" Temps de fonctionnement: {hours}h {minutes}m {seconds}s\n"
        f" Points de données: {len(sensor_data['timestamps'])}\n"
        f" Intervalle rapport: {REPORT_INTERVAL}s\n"
        f" Période moyenne: {AVG_PERIOD}s"
    )
```

Et lorsqu'on lance le bot discord grâce à "python3 bot.py" le bot envoie régulièrement des messages, les commandes fonctionnent, et cela va nous permettre un suivi à distance.

Nous avons également voulu rajouter un flux vidéo, pour ce faire, nous avons ajouté un nouveau script python camera.py qui se constitue de ce code :

```python
from flask import Flask, Response
from picamera2 import Picamera2
import cv2
import time

app = Flask(__name__)

# Activation de la caméra
picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration(main={"format": "XRGB8888", "size": (640, 480)}))
picam2.start()

# Choix du nb d'fps (image par seconde)
DESIRED_FPS = 1  # Puisque notre ruche est principalement en LTE, nous avons choisis 1 FPS pour éviter une saturation réseau
FRAME_DELAY = 1.0 / DESIRED_FPS  # On définit le délai par seconde

@app.route('/camera')
def camera():
    def generate():
        last_frame_time = time.time()

        while True:
            current_time = time.time()
            elapsed = current_time - last_frame_time

            # Si nécessaire, attendre
            if elapsed < FRAME_DELAY:
                time.sleep(FRAME_DELAY - elapsed)

            last_frame_time = time.time()

            frame = picam2.capture_array()
            ret, jpeg = cv2.imencode('.jpg', frame)

            if ret:
                yield (b'--frame\r\n'
                       b'Content-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n')

    return Response(generate(), mimetype='multipart/x-mixed-replace; boundary=frame')

# Lancement de Flask sur le port :5000
@app.route('/')
def index():
    return "Flask fonctionne ! Essayez /camera pour voir le flux."

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Importation des modules nécessaire pour la camera

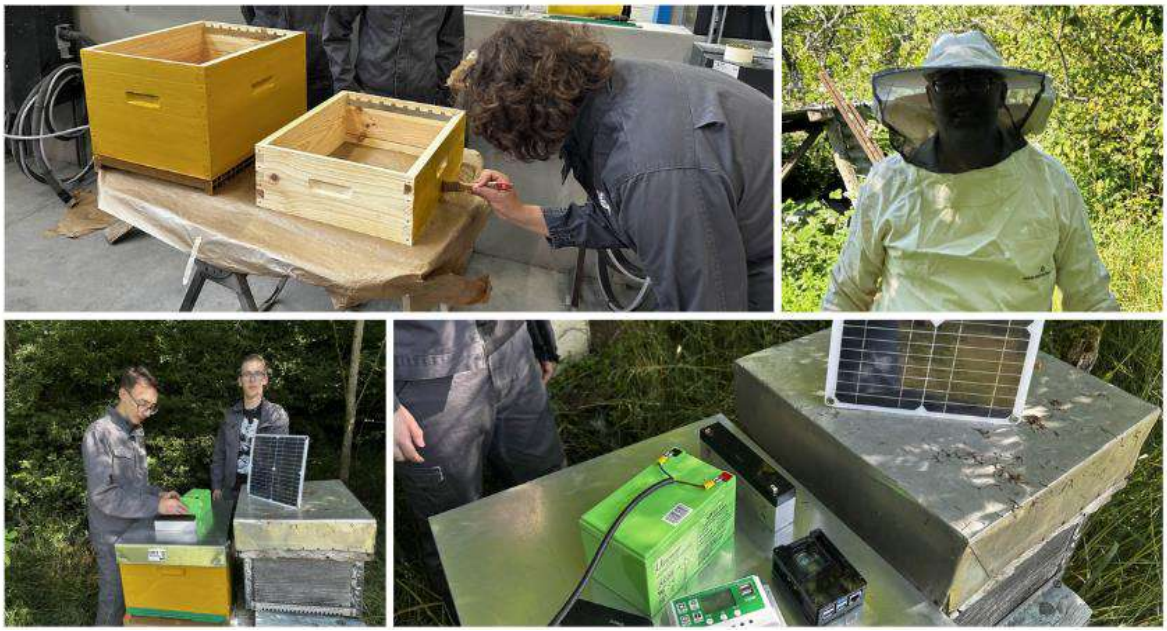
Nous pouvons donc accéder à notre caméra sur l'ip local http://0.0.0.0:5000/camera.
Pour simplifier l'usage, nous pouvons également afficher la caméra directement sur le dashboard de notre ruche connecté en 0.0.0.0:8080.

Est-ce que le projet est fini ?

Non le projet n'est pas fini, nous n'avons pas eu le temps de finaliser le boîtier imprimé en 3D car nous sommes aujourd'hui sur une version prototype.
Nous n'avons également pas obtenu le temps nécessaire pour installer les balances pour mesurer le poids de la ruche connectée.

Mais en état sans balance de poids, oui cette ruche est utilisable.

Nous avons également monté la ruche qui va expérimenter notre ruche connectée, nous l'avons peinte, et nous l'avons emmenée dans un lycée.



Conclusion

Ce projet de ruche connectée nous a permis de découvrir comment on peut vraiment aider le monde de l'apiculture, en combinant des capteurs simples, une caméra, un Raspberry Pi et des outils comme Discord, on a réussi à concevoir une ruche capable de transmettre en temps réel des informations utiles pour l'apiculteur : température, humidité, activité des abeilles, … et bientôt même le poids grâce aux balances.

Le plus impressionnant, c'est qu'on a pu créer tout ça pour un prix bien plus bas que les solutions déjà disponibles sur le marché d'environ 213 € contre plus de 600 € ailleurs, et avec plus de possibilités.

Même si tout n'est pas encore terminé (notamment l'installation des balances et le boîtier 3D), notre prototype fonctionne. Il peut déjà être utilisé pour suivre l'état d'une ruche à distance, ce qui est un vrai plus pour les apiculteurs.

Ce projet nous a aussi permis de mettre en pratique plein de compétences : en électronique, en programmation, en réseau et surtout de travailler en équipe sur un projet concret, utile et motivant.

On espère qu'il pourra inspirer d'autres élèves ou apiculteurs, et peut-être même évoluer dans les années à venir.

Remerciement

Nous tenons à remercier la région Grand Est qui a financé la ruche connecté ainsi que les composants nécessaires à la réalisation et de nous avoir donné l'opportunité de réaliser ce projet.

Nous remercions également M.Chaoui et M.Bazinet qui nous ont permis de réaliser ce projet et de nous avoir soutenu tout au long de ce projet.

Nous remercions aussi Anys Touffoutti pour avoir peint la ruche et Dylan Roux qui nous a aidé à réaliser le boîtier pour le raspberry