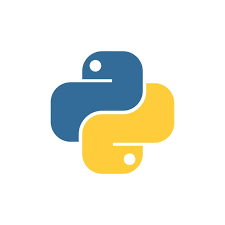
BUT3 - GEII - ESE

Thomas Gomes

IUT NICE CöTE d’AZUR

ROS

PROJET DE MAISON CONNECTE



# Introduction

Dans le cadre du module ROS du semestre 5 du BUT GEII, nous avons réalisé le projet de développer d’un système de maison connecté à partir du moyen ROS 2 (Robot Operating System) présent sur la distribution Linux Ubuntu22.04. Le but recherché est de concevoir et d’implémenter et de réaliser une architecture d’acquisition, de traitement et d’exploitation en temps réel de données diverses.

En effet, lors de ce projet j’ai réalisé un système permettant à une maison connectée de gérer ses lumières, son chauffage, ses volets et sa sécurité en passant par l’exploitation de données tel que la luminosité, la température, le temps, et la détection de personne. De plus, ce système devait être accessible et contrôler depuis une page web.

Ainsi, ce rapport présente tout d’abord l’architecture générale du projet, puis les différentes ressources permettant de réalisés via ROS 2, les tests et validations de son fonctionnement, ainsi que les résultats obtenus sur le web.

Table des matières

[Introduction 1](#_Toc190987982)

[Objectif du projet 3](#_Toc190987983)

[Fonctionnalité 4](#_Toc190987984)

[Architecture globale 5](#_Toc190987985)

[Publishers 7](#_Toc190987986)

[Publisher de Luminosité 7](#_Toc190987987)

[Publisher de Température 7](#_Toc190987988)

[Publisher de Temps 7](#_Toc190987989)

[Publisher de Présence Intérieur 8](#_Toc190987990)

[Publisher de Présence Extérieur 8](#_Toc190987991)

[Suscribeurs 9](#_Toc190987992)

[Souscription Luminosité 9](#_Toc190987993)

[Souscription Températures 9](#_Toc190987994)

[Souscription Temps 9](#_Toc190987995)

[Souscription Présence Intérieur 9](#_Toc190987996)

[Souscription Présence Extérieur 9](#_Toc190987997)

[Client-Serveur 10](#_Toc190987998)

[Test unitaire 10](#_Toc190987999)

[Interface Web 11](#_Toc190988000)

[Exécution du projet 13](#_Toc190988001)

[GITHUB 14](#_Toc190988002)

# Objectif du projet

L’objectif de ce projet est le suivant :

* Régulation de la Lumière : Automatiser l'allumage et l'extinction des lumières en fonction de la luminosité (Données reçue : en lux)
* Régulation de la Température : Automatiser le maintien d’une plage de température à l’aide d'un système de régulation par chauffage (Données reçue : en °C)
* Pilotage des volets: Automatiser l’ouverture et la fermeture des volets en fonction de l’heure (Données reçue : HH-MM).
* Détection de présence : Automatiser la sécurité par l’activation et désactivation d’un buzzer sonore lorsqu’une présence intérieur et extérieur est détecté.

# Fonctionnalité

* **Régulation de la Lumière (Capteur de luminosité) :**
  + Si la luminosité est basse, le système allume les lumières.
  + Si la luminosité est haute, le système éteint les lumières.
* **Régulation de la Température (Capteur de température) :**
  + Si la température est basse, le chauffage s'allume.
  + Si la température est haute, le chauffage s'éteint.
* **Contrôle des volets (Capteur de temps) :**
  + Si certaine heure, le volet s’ouvre.
  + Si certaine heure, le volet se ferme
* **Système d'Alarme (Bouton poussoir)**
* **Si alarme désactivée** → Appuie BP → **Alors alarme activée  :** 
  + - Fermeture des volets + Arrêt mesure de temps
    - Extinction des lumières + Arrêt mesure de luminosité
    - Coupure du chauffage + Arrêt mesure de température
    - Activation du capteur de présence intérieur et extérieur.
      * **Capteur de présence Intérieur et Extérieur :**

Si mouvement intérieur ou extérieur détecté, l’alarme sonore est activée.

Si pas de mouvement intérieur ou extérieur détecté, l’alarme sonore est désactivée.

* + **Si alarme activée** → Appuie BP → **Alors alarme désactivée  :** 
    - * Activation mesure de temps
      * Activation mesure de luminosité
      * Activation mesure de température
      * Coupure buzzer + Arrêt mesure détection de personne

Bouton d’activation et de désactivation du système d’alarme contrôler via la page web

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

Le contenu généré par l’IA peut être incorrect.

# Architecture globale

Le projet repose sur plusieurs nœuds d’échange d’informations ROS 2 assurant la collecte de données environnementales ( température, luminosité, temps et présence), la gestion d’actions automatisées (contrôle d’un volet, d’un chauffage, de lumières ou d’un buzzer), et l’interaction avec une interface web permettant une supervision à distance via un package nommer *projet\_pkg*. Ce système permet une communication facile et efficace grâce à la publication de donnée via des Publishers sur des topics spécifiques. De même pour la souscription de données par des souscripteurs qui s’abonne à ces topics pour récupérer les données de mesures.

Ainsi pour notre maison connectée nous retrouvons différents Publishers et topics :

1. **Capteur de luminosité**

Publisher **/luminosity\_publisher** → Publie sur le topic **/capteur/luminosity**

1. **Capteur de température**

Publisher **/temperature\_publisher** → Publie sur le topic **/capteur/temperature**

1. **Capteur de temps**

Publisher **/time\_publisher** → Publie sur le topic **/capteur/time**

1. **Détection de présence en intérieur**

Publisher **/presence\_Int\_publisher** → Publie sur le topic **/capteur/presence/interieur**

1. **Détection de présence en extérieur**

Publisher **/presence\_Out\_publisher** → Publie sur le topic **/capteur/presence/exterieur**

Toutes les valeurs de capteurs sont mit à jours toutes les 5 secondes.

Par la suite, l’ensemble de ces topics sont ensuite abonnés par **/capteur\_subscriber**, qui collecte les informations de tous les capteurs en temps réel afin de les traiter et de les exploiter pour effectuer une action.

En parallèle de l’émission et la réception de données en continu via les Publishers et souscripteurs, nous retrouvons dans notre projet la présence d’un système de client/serveur. Ce dernier est utilisé lorsqu’un nœud doit demander une action spécifique à un autre nœud et attendre une réponse. Ce système évite donc un échange continue d’informations mais d’en avoir seulement lorsque cela est nécessaire.

* Ainsi nous retrouvons un serveur nommé **/alarme\_service** et un client nommé **/alarme\_client** qui permettre la gestion d’un mode d’alarme pour la maison connectée. Le but étant pour ce système d’activer et de désactiver l’alarme en fonction du besoin de l’utilisateur.

L’ensemble de cette architecture du système de maison connecté peut se représenter via le schéma suivant :

**Une image contenant texte, diagramme, ligne, Police

Le contenu généré par l’IA peut être incorrect.**

## Publishers

### Publisher de Luminosité

Script : publisher\_luminosity.py  
Type de message : Float32

Fonctionnement :

* Génère une valeur aléatoire entre 0 et 1000 lux
* Publie cette valeur sur le topic ROS **/capteur/luminosity** toutes les 5 secondes.
* Affiche la valeur publiée dans le terminal.

### Publisher de Température

Script : publisher\_temperature.py  
Type de message : Float32

Fonctionnement :

* Génère une température aléatoire entre -10°C et 45°C.
* Publie cette valeur sur le topic ROS **/capteur/temperature** toutes les 5 secondes.
* Affiche la valeur publiée dans le terminal.

### Publisher de Temps

Script : publisher\_time.py  
Type de message : String

Fonctionnement :

* Génère une heure aléatoire pour 24h (ex : "14:30").
* Publie cette valeur sur le topic ROS **/capteur/time** toutes les 5 secondes.
* Affiche la valeur publiée dans le terminal.

### Publisher de Présence Intérieur

Script : publisher\_presenceInt.py  
Type de message : Bool

Fonctionnement :

* Génère un état aléatoire (True pour détection, False pour vide).
* Publie cette valeur sur le topic ROS **/capteur/presence/interieur** toutes les 5 secondes.
* Affiche la valeur publiée dans le terminal.

### Publisher de Présence Extérieur

Script : publisher\_presenceOut.py  
Type de message : Bool

Fonctionnement :

* Génère un état aléatoire (True pour détection, False pour vide).
* Publie cette valeur sur le topic ROS **/capteur/presence/interieur** toutes les 5 secondes.
* Affiche la valeur publiée dans le terminal.

## Suscribeurs

Tous les souscribeurs sont dans le script python : **info\_web.py**

Ce fichier contient un nœud **Capteur\_Subscriber**, qui écoute plusieurs topics ROS simultanément, et qui traite les informations reçues selon les capteurs de la manière suivante :

### Souscription Luminosité

Topic : **/capteur/luminosity/**

Traitement de l’information : Lumières ON si < 400 lux, sinon OFF

### Souscription Températures

Topic : **/capteur/temperature/**

Traitement de l’information : Chauffage ON si < 18°C, sinon OFF

### Souscription Temps

Topic : **/capteur/time/**

Traitement de l’information : Volets ouverts entre 07:00 et 19:00, sinon fermés

### Souscription Présence Intérieur

Topic : **/capteur/presence/intérieur**

Traitement de l’information : Si présence détectée et Alarme ON → alors Buzzer ON

### Souscription Présence Extérieur

Topic : **/capteur/presence/extérieur**

Traitement de l’information  : Si présence détectée et Alarme ON → alors Buzzer ON

## Client-Serveur

Client : Classe **Alarme\_Client** contenu dans script python info\_web.py

Serveur : Classe **Alarme\_Service** contenu dans script python : service\_bp.py

L’utilisateur déclenche l’activation ou la désactivation de l’alarme via la page web :

Le client Alarme\_Client envoie une requête via le service etat\_alarme.

Le service Alarme\_Service reçoit la requête, vérifie l’état actuel de l’alarme et répond :

* + Si l’alarme est déjà dans l’état attendu alors message d’erreur.
  + Sinon, changement d’état de l’alarme et envoie confirmation.
* Le client reçoit réponse et met à jour page web via WebSocket

## Test unitaire

Chaque élément du projet possède un script de test unitaire pour vérifier le bon fonctionnement d’un publisheur ou du serveur :

En effet le package projet\_pkg possède des scripts avec un suscribeurs unitaire et un client manuel permettant de tester chaque fonctionnalité tel que :

* Temperature : controle\_temperature.py (test de publisheur\_temperature.py)
* Luminosité : controle\_luminosity.py (test de publisheur\_luminosity.py)
* Temperature : controle\_time.py (test de publisheur\_time.py)
* PrésenceInt  : controle\_presenceInt.py (test de publisheur\_presenceInt.py)
* PrésenceOut  : controle\_presenceOut.py (test de publisheur\_presenceOut.py)
* Serveur : client\_bp.py (test de serveur\_bp.py)

*Une image contenant texte, Police, capture d’écran, diagramme

Le contenu généré par l’IA peut être incorrect.Exemple de test unitaire pour le capteur de température*

# Interface Web

Pour afficher et contrôler notre système de maison connecté nous utilisons les moyens de **Flask** et **Flask-SocketIO** pour le serveur web et les communications en temps réel.

En effet leur utilisation va permettre de lire les valeurs des capteurs et piloter l’alarme en temps réel.

Cela fonctionnement par les étapes suivantes :

* Initialisation d’un serveur Flask qui écoute sur http://0.0.0.0:5000.
* Les données des capteurs sont mises à jour dans un dictionnaire global **capteur\_data** et envoyées en temps réel à l’interface web via socketio.emit().
* Emploie du client Alarme\_Client pour activer/désactiver l’alarme via Alarme\_Service.
* Utilisation d’un thread pour exécuter le système ROS2 en même temps que Flask.

L’interface web est créer via :

un fichier index.html contenu dans le dossier /templates (requis pour Flask)

Fichier index.css et index.js contenu dans le dossier /static (requis pour Flask)

Ainsi la page web permet d’affiche les valeurs des capteurs ainsi que le traitement qui convient

(Fermeture volet, allumage chauffage …). Et de piloter le bouton permettant de switch entre le mode alarme à ON ou à OFF

Lorsque Alarme OFF : Mode détection et buzzer figés pendant que les autres capteurs sont actifs

Une image contenant texte, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect.

Lorsque Alarme ON : Mode détection et buzzer actifs pendant que les autres capteurs sont figés

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

Le contenu généré par l’IA peut être incorrect.

# Exécution du projet

Le projet de maison connecté réaliser peut-être mis en route de plusieurs manières :

**Méthode 1 : Pas à Pas**

Pour exécuter le projet vous pouvez lancer un par un tous les nœuds du package

En effecutant les commandes bash suivantes :

>>>     cd ~/ros2\_ws

 >>>   colcon build

 >>>   source install/setup.bash

>>> ros2 un projet\_pkg <nom du nœud> *Exemple : run projet\_pkg publisher\_temperature*

*>>> … (exécution des autres nœuds) -> voir script setup.py pour voir autre noeud*

*>>> … (exécution des autres nœuds)*

*>>> python3 src/projet\_pkg/projet\_pkg/info\_web.py*

**Méthode 2 : Script bash**

Vous pouvez également lancer le projet dans son intégralité via le script bash : **start\_projet\_pkg.sh**

Pour cela, exécuter dans le dossier *ros2\_ws/src/projet\_pkg, la commande suivante*

>>> bash start\_projet\_pkg.sh

Puis sélectionner Y pour reconfigurer si modification ou sinon N

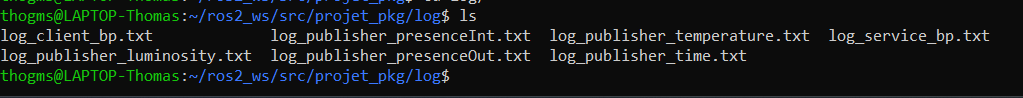
* + - Projet lancer : récupération de l’adresse IP (ex : http://127.0.0.1:5000)

/ ! \ Avec cette méthode, les affichages de chaques scripts sont redirigé vers des fichiers textes de logs (ex : log/log\_publisheur\_luminosty.txt)

*Exécution bash :*

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.

Fichier log :

# GITHUB

Pour récupérer mon Projet ROS MAISON, RDV sur GITHUB

>>> git clone https://github.com/thoxgms/ROS\_ProjetMaison.git