# Architecture Logicielle

Notre architecture logicielle est relativement simple, nous avons dans le dossier principal « api », quatre sous-dossiers : tests, data, scripts et notes.

Pour ce qui est du dossier « tests », nous concentrons ici tous nos développements de tests. Pour chacun de ces développements, on pourra trouver des scripts de la forme :

* gateway\_\*\*\*.py : contient le code principal, ce fichier permet d’implémenter la collecte de données, leur traitement, et les sortie (« output ») mises en place en fonction des données captées.
* main\_\*\*\*.py : Il s’agit du fichier principal d’exécution, il va appeler le code développé dans le « gateway » pour faire ensuite tourné le système en temps réel.
* branchements\_\*\*\*.txt : précise l’architecture physique de notre système (capteurs & ports utilisés).

Dans ce dossier tests, il est possible de trouver d’autres scripts supplémentaires, hors de l’architecture classique détaillée, qui sont des scripts utilisés en compléments pour ajouter des fonctionnalités particulière (traitement des données ou écriture dans un fichier annexe par exemple).

Le dossier « scripts » contient notre travail final. Il est possible d’y retrouver un fichier « main.py » d’exécution, un fichier « gateway.py » qui contient le code principal du système (récupération des valeurs captées (« input »), leur traitement, et la gestion des outputs). Le dernier script présent dans le fichier « scripts » est intitulé « LastDataDf.py ». Il s’agit d’une classe Python appelée dans le fichier « gtaeway.py » afin de décentraliser et factoriser le code pour le rendre plus compréhensible et plus facile à maintenir en cas d’utilisation sur le long terme. Chacun de scripts et leurs interactions seront ensuite détaillés. Enfin, il est possible de trouver le dossier « scripts », le fichier « all\_data.csv » qui est un fichier de données contenant les données agrégées captées par le système au cours de l’exécution des scripts. Ce fichier aurait dû se situer dans le dossier « data » de notre architecture (cf. pistes d’améliorations).

**Fonctionnement des scripts :**

Nous allons ici détailler comment les scripts du dossier « scripts » fonctionnent et interagissent entre eux. Cette logique est bien sur applicables aux scripts trouvable dans le dossier « tests » dont l’architecture est similaire.

* **gateway.py** : C’est le script principal du système. Il contient le code de la classe Python : Gateway. Cette classe permet, quand elle est instanciée, de se connecter avec du code aux capteurs connectés physiquement au RasberryPI. En faisant cela, il est possible d’accéder aux valeurs captées dans la méthode de classe « inputUpdate». Cette méthode permet d’accéder à l’état du système au moment de l’appel de la méthode. Ensuite, il est possible de traiter les données récupérées avec la méthode de classe « inputProcessing ». Ces traitements peuvent être des calculs ou l’écriture dans un fichier annexe par exemple. Dans notre cas, cette méthode de classe nous permet de calculer des moyennes sur les valeurs captées sur les 20 dernières secondes. Ce traitement va nous servir ensuite pour gérer les outputs. Vient ensuite une troisième méthode classe, intitulée « graph » qui permet de gérer le graphe d’état. Il s’agit ici de faire le point sur les états du système. Un état peut être assimilé à un stade du système, ces stade vont, dans notre code, modifier le fonctionnement du RasberryPI et plus particulièrement du traitement des inputs et de leurs rendus. Dans notre code, il y a deux états possibles pour le système :

- état 1 - Fonctionnement habituel : lorsque l’IUT (atelier) est ouvert, l’état du système est l’état 1. On va récupérer les données captées, puis calculer une moyenne sur les données des 20 dernières secondes avant de redéfinir la couleur des LED en fonctions des moyennes calculée (voir méthode « outputProcessing » & « outputUpdate »). De plus, si les moyennes calculées sont au-dessus de seuil sanitaires ou que les valeurs enregistrées sont significatives d’un danger, une alarme se met en place avec l’activation du clignotement rouge de 2 des LED et d’un bip continu du buzzer lors de l’alerte.

- état 2 – Fonctionnement nocturne de la station : Quand l’IUT est fermé, c’est dans cet état qu’est le système. Dans cet état, la station sert de détecteur d’intrusion, d’alerte incendie, et d’alerte de fuite de gaz (présence de gaz dangereux pour l’Homme). Il s’agit plus de collecter des données sur les horaires des différentes alertes et de collecter des données plutôt que de véritablement avertir (là où des alarmes spécialisées seront beaucoup plus efficaces – et réglementaires accessoirement). Ce deuxième état est activable par l’utilisateur via un bouton de la station, s’il n’est pas déclenché manuellement, il se met en place en fonction des horaires de fermeture de l’IUT. De même, il est possible de rebasculer sur le premier état quand ce bouton est appuyé.

On a ensuite les méthodes de classe « outputProcessing » et « outputUpdate » qui permettent de mettre à jours les outputs de la station (LEDstick et buzzer) en fonction des traitements et des inputs. La première méthode concerne l’aspect logique et calculatoire, et la seconde permet la mise à jour technique des composants. C’est dans la première méthode que nous écrivons les données calculées dans un fichier annexe avec une granularité de 3 minutes

Une méthode de classe nommée « alerte » est aussi implémentée. Elle permet de concrétiser les alertes avec les clignotements et les bips du buzzer.

* **main.py** : Ce script est celui qui est exécuté lorsque la station est en marche. Il permet d’instancier la classe Gateway, et d’appeler dans une boucle infinie, de manière successive, les méthodes : inputUpdate, inputProcessing, graph, outputProcessing, outputUpdate. Ce script permet la répétition de la mise à jour de l’état du système et la collecte de données en temps réel.
* **LastDataDf.py** : Cette classe permet de faciliter le traitement. Il s’agit ici d’utiliser la bibliothèque Python nommée Pandas. Elle permet d’utiliser l’objet ‘dataframe’ (tableau de données) au sein de python et de faciliter les calculs réalisés sur ces objets. Le fait de créer une classe permet de réutiliser le code : pour le calcul des moyennes des 20 dernières seconde – afin de mettre à jour les LED - ou bien sur les 3 dernière minutes – pour l’écriture dans le fichier csv annexe.

Note : chacun de ces scripts est lui-même documenté avec de nombreux commentaires afin de faciliter la compréhension et la maintenance du système.

**Pistes d’améliorations :**

Bien que les dossiers « notes » et « data » soient vide, il aurait été judicieux de s’en servir pour expliquer nos intention et document de manière hexogène notre code (dans le dossier « notes », pour donner des consignes d’utilisation et des précisions quant à l’utilité du système mis en place). Le dossier « data » aurait quant à lui pu être utilisé pour le recueil de fichier de données CSV (code non implémenté par manque de temps).

# Evolution du Système