

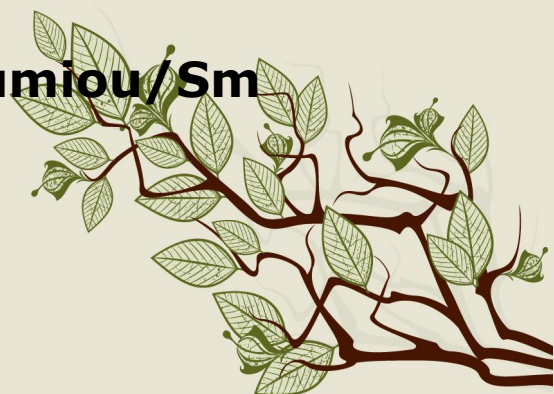


# Smart Bud

Smart Bud est un projet IoT de petite ampleur dont l'objectif principal est de pouvoir monitorer une ou plusieurs serres à distance. Le principe est simple : une serre possède plusieurs capteurs qui permettent d'enregistrer certains paramètres importants au bon développement de plantes et ces paramètres sont affichés sur un affichage web qui permet de suivre à tout moment le bon état des pousses. Une automatisation de l'arrosage est aussi en place afin d'arroser les plantes lorsque les paramètres sont trop faibles (par exemple l'humidité).

Notre Git :

**<https://github.com/Namioumiou/Smart-Bud>**





<b>PRÉSENTATION DE L'ÉQUIPE</b>	<b>3</b>
<b>COMMUNICATIONS DES DIFFÉRENTS ÉLÉMENTS</b>	<b>3</b>
<b>PARTIE 1 – LE NODE LORAWAN</b>	<b>4</b>
LES CAPTEURS	4
L'ESP32	4
RASPBERRY PI 5 ET HAT SX1262	6
BROKER MQTT	6
BASE DE DONNÉES SQLITE	7
SERVICE MQTT	7
SERVICE LoRa	8
<b>PARTIE 2 – LA PASSERELLE LORAWAN</b>	<b>9</b>
<b>PARTIE 3 – LE TRAITEMENT SUR THE THINGS NETWORK</b>	<b>10</b>
HANDLE DE LA PASSERELLE	10
APPLICATION SMART BUD	10
DISPOSITIF DE NOTRE NODE	11
PAYLOAD FORMATTER	11
BASE INTERNE À THE THINGS NETWORK	12
WEBHOOK HTTP	13
<b>PARTIE 4 – L’AFFICHAGE SUR THINGS BOARD</b>	<b>15</b>
RÈGLE DE TRAITEMENT DES UPLINKS ET DE CRÉATION DES TAGS	15
DISPOSITIF	16
DASHBOARD	16
LIENS DES TAGS SUR LE DASHBOARD	17





## Présentation de l'équipe

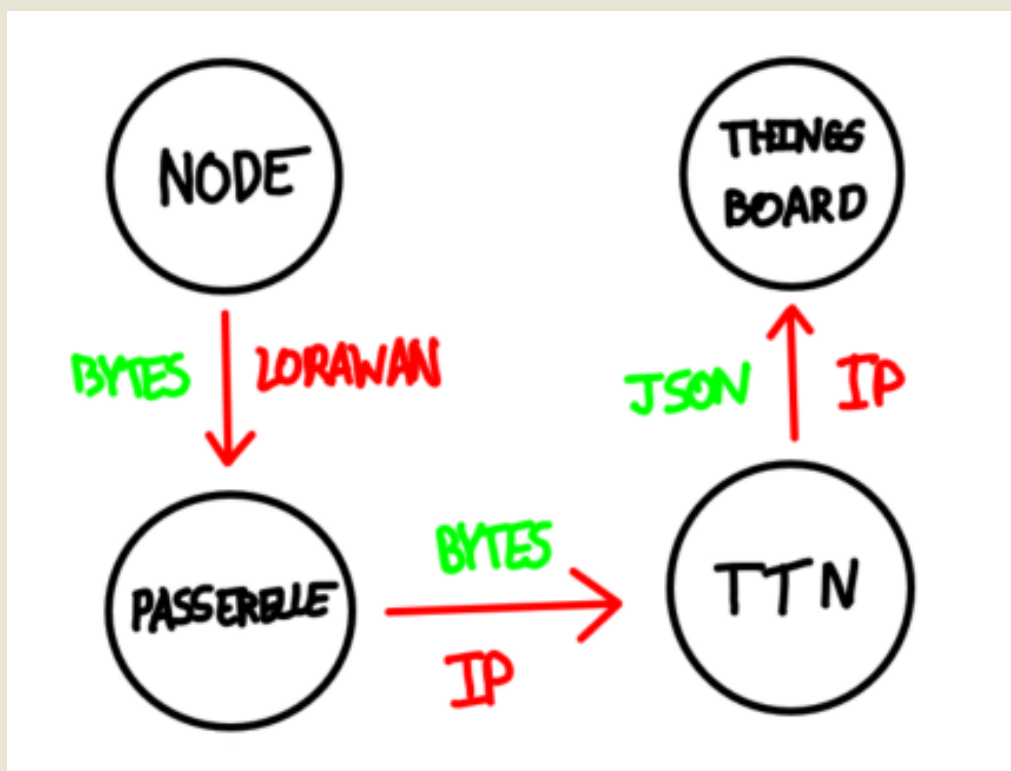
Quatre personnes ont contribué à la mise en place de ce projet :

- Ambre ROUILLON : Cheffe de projet et responsable de la partie data
- Enzo BOURGEOIS : Responsable de la partie passerelle Raspberry
- Guillaume VERN : Responsable de la partie web
- Alexy MANSUY : Responsable de la partie capteurs

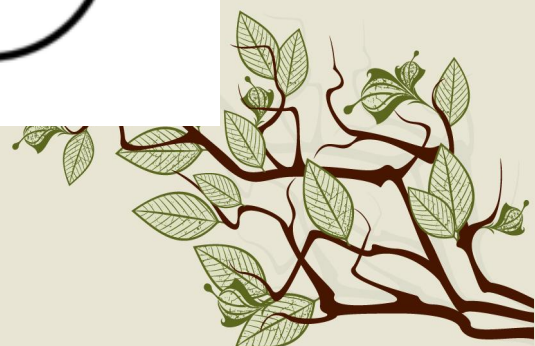
Tout au long de ce dossier, nous nous efforcerons de suivre le flux de données comme ordre de présentation. Nous allons donc passer sur quatre éléments principaux :

1. Le Node LoRaWAN
2. La passerelle LoRaWAN
3. Le traitement sur The Things Network
4. L'affichage sur ThingsBoard

## Communications des différents éléments



**Figure 1 :** Différents protocoles de communications ainsi que leur format entre chaque grande partie du système





## Partie 1 – Le Node LoRaWAN

Le système complet repose en majorité sur cet élément. Il s'agit du point de collecte des données. Il se compose des équipements suivants :

- Des capteurs et un actionneur :
  - o Un capteur BME280
  - o Un capteur BH1750
  - o Un premier capteur capacitif
  - o Un second capteur capacitif (volontairement séparé du premier)
  - o Un actionneur de type pompe à eau
- Un ESP32
- Un Raspberry Pi 5
- Un HAT SX1262

### Les capteurs

Commençons par présenter les fonctions des capteurs et actionneur :

- Le capteur BME280 : mesures de la température et de l'humidité ambiant
- Le capteur BH1750 : mesures de la luminosité
- Le premier capteur capacitif : mesures de l'humidité du sol
- Le second capteur capacitif : mesures du niveau d'eau dans le bac d'arrosage
- La pompe à eau : s'active lorsque les paramètres d'humidité sont insuffisants

### L'ESP32

Ces capteurs sont connectés à un ESP32 qui sert à trois fonctions :

- Lire les mesures des capteurs à intervalles réguliers (arbitrairement paramétré à 3 minutes)
- Effectuer un pré-traitement des données ainsi qu'une mise en forme du payload
- Publier les différents payload résultants via MQTT à la passerelle Raspberry Pi 5 sur un topic dédié et nommé « sensor data ».







Afin de rendre le Node fonctionnel sans électricité ménager, celui-ci est fonctionnel par piles ou accumulateurs grâce à l'aide d'un convertisseur de tension. Le convertisseur est réglé de manière à assurer une tension d'alimentation constante de 5V du moment que la batterie est suffisamment chargée. Seul l'ESP32 est alimentée en 5V. La sortie 3V3 est utilisée pour alimenter les capteurs afin d'obtenir des mesures plus prévisibles sur les capteurs analogiques tels que les capteurs capacitifs.

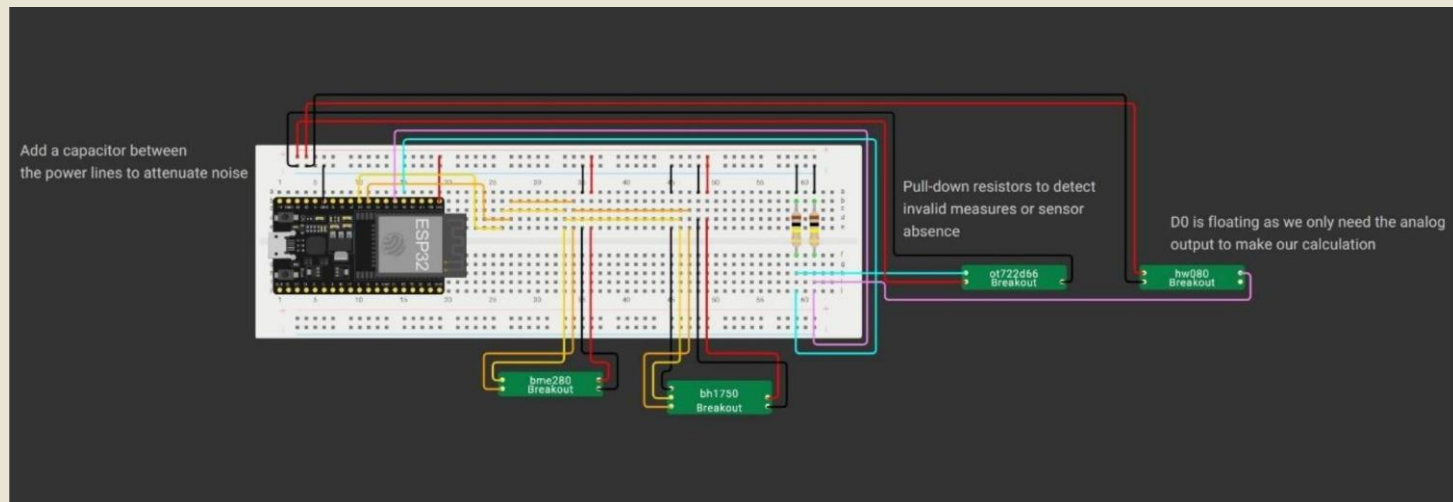


Figure 2 : Schéma de l'ESP32 et des capteurs connectés

Afin d'assurer la bonne mise en fonction de l'ESP32, plusieurs scripts en MicroPython ont été transversés dans sa mémoire flash :

- **bh1750.py** : Fournit des fonctions pour la réception des données, sur le bus I2C de l'ESP32, du capteur BH1750 ainsi que des fonctions de conversion
- **bme280.py** : Fournit des fonctions pour la réception des données, sur le bus I2C de l'ESP32, du capteur BME280 ainsi que des fonctions de conversion
- **ot722d66.py** : Fournit des fonctions pour la réception des données, via un port DAC de l'ESP32, du capteur capacitif de niveau d'eau ainsi que des fonctions de conversion
- **hw080.py** : Fournit des fonctions pour la réception des données, via un port DAC de l'ESP32, du capteur capacitif d'humidité du sol ainsi que des fonctions de conversions
- **utils.py** : Fournit des fonctions utilitaires comme la conversion d'un nombre à virgule flottante en nombre à virgule fixe (avec taille binaire paramétrable)
- **data\_transmitter.py** : Se connecte à Internet et crée un gestionnaire un client MQTT puis utilise les fonctions des autres scripts pour générer le payload final afin de publie les erreurs obtenues et le payload lui-même via MQTT sur le topic « sensor\_data » au broker installé sur la passerelle Raspberry
- **webrepl\_connect.py** : Permet la connexion à un client WebREPL qui permet le transfert Over The-Air (OTA via le Wi-Fi) de mise à jour de scripts.

À des fins pédagogiques, ces scripts, ainsi que les protocoles utilisés, ont tous été recodés afin d'en comprendre le fonctionnement.





Le payload en question se compose des bytes suivants (leur représentation en bits est marquée entre parenthèses et sera utilisée pour représenter le payload plus bas) :

- 1 byte d'erreur (E)
- 2 bytes de température de l'air ambiant (T)
- 2 bytes d'humidité de l'air ambiant (H)
- 2 bytes de luminosité (L)
- 1.5 byte de niveau d'eau (W)
- 1.5 byte d'humidité du sol (S)

Ce qui correspond à ceci en binaire :

EEEEEEEE TTTTTTTT TTTTTTTT HHHHHHHH HHHHHHHH LLLLLLLL LLLLLLLL WWWWWWWW  
WWWWSSSS SSSSSSSS

## Raspberry Pi 5 et HAT SX1262

La Raspberry Pi 5 est la partie la plus conséquente du Node. Elle se compose de quatre parties distinctes :

- Un broker MQTT
- Une base de données SQLite
- Un service MQTT
- Un service LoRa

## Broker MQTT

Le broker MQTT à un usage simple et est basé sur le package mosquitto-server. La Raspberry s'y connecte et souscrit au topic « sensor\_data » tandis que l'ESP32 s'y connecte et y publie les payload enregistrés. Des utilisateurs ont été créés afin de tester les multiples connexions et d'éviter les connexions anonymes.





## Base de données SQLite

Une simple base de données SQLite a été créée pour réceptionner les payloads reçus, depuis la souscription du client MQTT dans le service MQTT sur la Raspberry au topic « sensor\_data ». L'entrée la plus récente dans la table sera ensuite récupérée toutes les 20 minutes par le service LoRa afin d'être envoyée à The Things Network d'abord via le HAT SX1262 puis forwardée par la passerelle Dragino LSP8.

La durée de 20 minutes est arbitraire et a été choisie de manière à envoyer un payload assez régulièrement dans la journée tout en respectant la politique d'usage des fréquences libres (ici 868MHz) qui est limitée à 30 secondes de airtime par device par jour (pour TTN en tout cas). Cela permet d'avoir un historique de données relativement complet sur l'entièreté de la journée tout en restant dans la légalité. Les 20 minutes ont été calculées avec un payload de 10 bytes sur la fréquence 868MHz (fréquence européenne), avec un facteur d'étalement à SF10 (spreading factor 10) et une bande passante de 125KHz. Ces données nous donnent un airtime de 370ms par payload :

Input Bytes	Spreading Factor	Region	Bandwidth
10	SF10	EU868	125 kHz

**Result**  
**370.7 ms**  
Time on air

Figure 3 : Airtime autorisé par jour par device

## Service MQTT

Le service MQTT est un simple service créé pour systemd qui exécute un code développé en C++ avec non seulement l'aide de la bibliothèque libmosquitto et de la bibliothèque libsqlite3. Ce service tourne dans une boucle infinie. Tout d'abord le code, initialise le gestionnaire de connexions pour MQTT et pour SQLite. Puis le service utilise le client MQTT pour souscrire au topic « sensor\_data ». Enfin, à chaque fois que l'ESP32 publie un payload sur le topic « sensor\_data », la Raspberry y étant souscrite, elle récupère les bytes envoyées et les enregistre dans la base de données SQLite via le gestionnaire de connexion créé plutôt.



## Service LoRa

Enfin, le service LoRa est un service créé pour systemd qui exécute un code développé en C++ avec l'aide d'une bibliothèque nommée RadioLib. Cette bibliothèque permet de gérer les communications radio incluant le LoRaWAN. Nous l'avons utilisé en raison du fait qu'il s'agissait d'une des bibliothèques recommandées pour l'usage avec le C++. L'exécutable utilisé tourne dans une boucle infinie.

Le code commence par initialiser les différentes instances utilisées pour la communication LoRaWAN telles que l'instance du SX1262 pour le HAT SX1262 ainsi que les constantes d'accès au dispositif créé sur The Things Network puis initialise la connexion OTAA (Over The Air Activation) du dispositif.

Une fois la connexion établie, le programme utilise la bibliothèque libsqlite3 pour requêter une table de la base de données SQLite sur le Raspberry afin d'y récupérer le dernier payload reçu que le service MQTT y a stocké. Une fois récupérés, ils sont enfin envoyés sur le réseau LoRaWAN afin d'être forwardés par la passerelle LoRaWAN Dragino LSP8 sur The Things Network.

Au final, le Node ressemble à ceci :

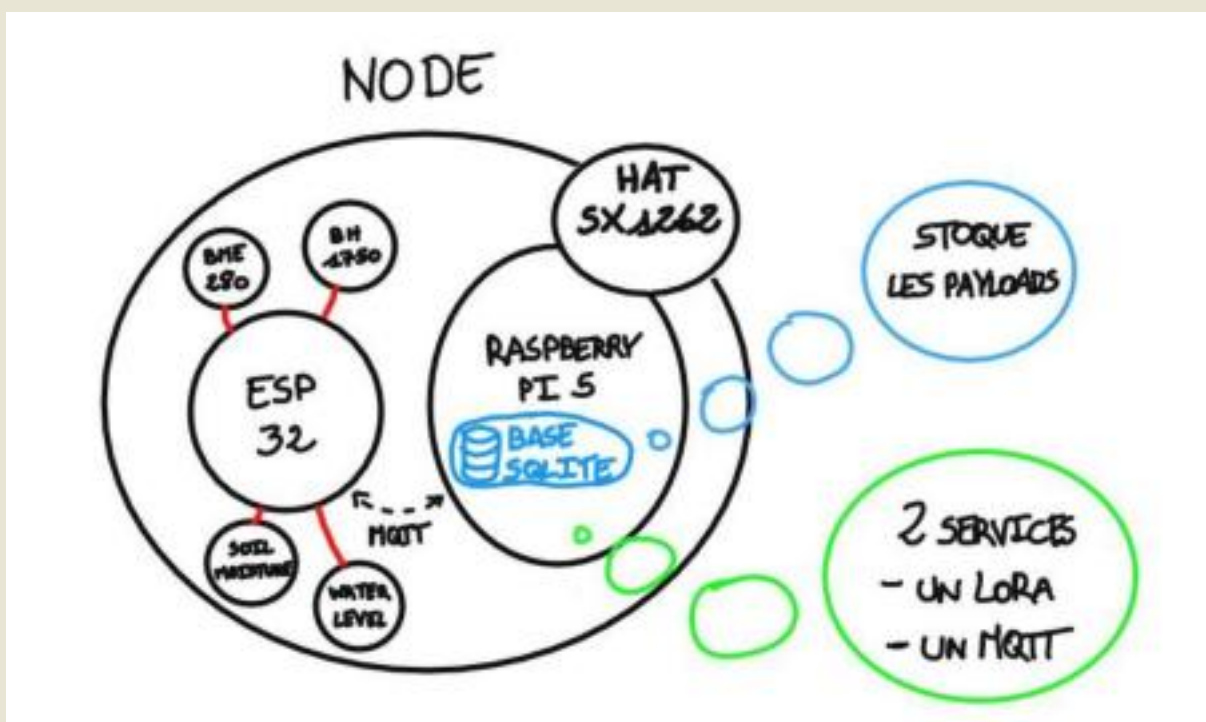


Figure 4 : L'allure du Node dans le schéma d'architecture





## Partie 2 – La passerelle LoRaWAN

La passerelle LoRaWAN n'était pas vraiment prévu pour ce projet mais a finalement dû être commandée afin de permettre la communication LoRaWAN dans les zones où il n'y a pas de passerelles publiques accessibles (ce qui a été notre cas lors de l'avancement du projet depuis chez nous). La passerelle en question est une passerelle Dragino LSP8 qui est une passerelle très simple d'utilisation avec une interface web utilisable lors de sa connexion qui permet sa configuration via une interface graphique compréhensible.

La passerelle a dû être configurée de manière à permettre le transfert des paquets décodés sur la fréquence 868MHz à The Things Network. Cela nécessite de passer non seulement l'URL de The Things Network mais aussi quelques informations de configuration supplémentaires, telle qu'une clé API (à générer sur The Things Network au niveau de l'application hôte). Les paquets décodés sont transférés via IP car la passerelle est connectée soit via le Wi-Fi (qui est géré) soit via un câble Ethernet avec accès à Internet.

Le schéma est simpliste étant donné que la passerelle ne fait que transférer les paquets



*Figure 5 : Simple passerelle qui récupère les bytes en LoRaWAN depuis la Raspberry et les transfère via IP vers The Things Network*





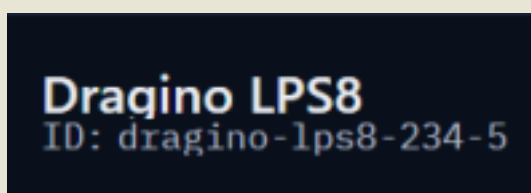
## Partie 3 – Le traitement sur The Things Network

Cette partie est la plus conséquente dû en partie à l'ajout de la passerelle. Elle se découpe en six parties :

- le handle de la passerelle Dragino LSP8 ;
- l'application Smart Bud ;
- le dispositif de notre Node ;
- le payload formatter ;
- la base interne à The Things Network ;
- le webhook http

### Handle de la passerelle

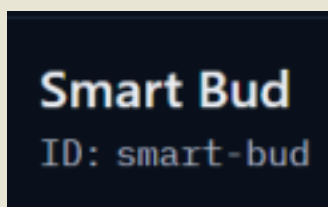
Le handle de la passerelle est juste une représentation de la passerelle dans The Things Network. Cependant, il permet aussi de journaliser les messages transférés par la réelle passerelle Dragino LSP8. Elle sert aussi à transférer les messages aux applications. Cela va avoir son intérêt dans la partie payload formatter. Voici le handle de la passerelle dans The Things Network :



*Figure 6 : Handle de la passerelle dans The Things Network*

### Application Smart Bud

Une application permet d'enregistrer des dispositifs tel que notre Node afin de pouvoir transférer les données reçues à une application tierces qui pourra en faire usage (dans notre cas ThingsBoard). Voici l'application dans The Things Network :



*Figure 7: Application Smart Bud sur The Things Network*



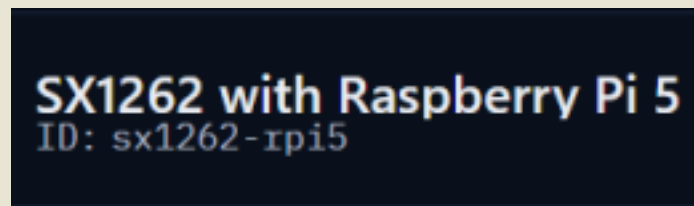


## Dispositif de notre Node

Le dispositif est un handle qui gère un Node. Dans notre cas, il permet de gérer l'ensemble des données mesurées par notre ESP32 et transférées jusqu'à The Things Network. Il s'agit du dernier point à accéder pour pouvoir envoyer les données dans un format voulu à une application tierce donnée (dans notre cas du JSON à envoyer à Things Board). Le dispositif a été créé d'une manière bien précise aux vues de la spécificité de notre Node. Notre Node utilise un HAT SX1262 et sa connexion est moins simple que pour un dispositif industriel (un Node tout en un dans un petit boîtier). Nous avons donc dû utiliser une configuration bien précise qui est la suivante :

- connexion OTAA (Over The Air Activation)
- dispositif classe A uniquement (The Things Network n'envoie les downlink que lorsque l'application reçoit un uplink ce qui est particulièrement adapté à notre cas d'usage d'accumulateurs car cela permet d'économiser l'énergie)
- spécification LoRaWAN 1.1.0
- EUID générés aléatoirement par The Things Network
- réinitialisation des join (extrêmement dangereux dans un cas réel mais obligatoire dans notre cas pour permettre le rejoin de notre dispositif dans le cas d'une déconnexion de celui-ci)

Voici le dispositif dans The Things Network :



*Figure 8 : Dispositif de notre Node dans The Things Network*

## Payload formatter

Dans The Things Network, il est possible de créer un script JavaScript permettant de faire sens des bytes arrivant en entrée d'une application. Notre application peut donc utiliser ce fameux script pour donner sens aux données transférées depuis notre handle de passerelle à notre application. Il peut être créé au niveau de l'application comme au niveau du dispositif. Le nôtre est un script qui prend ces bytes en entrée et qui forme un JSON à partir du code d'erreur envoyé dans les données. Si les données sont en erreur, elles ne feront pas partie du JSON de sortie. Sinon elles sont décodées puis ajoutées au JSON. C'est la seule opération que fait notre script. Notre script a été créé au niveau du dispositif. Le voici dans notre application :



### Formatter type \*

Custom Javascript formatter



### Formatter code \*

```
108 }
109
110 if (!(errorState & OT722D66_BIT)) {
111     waterLevel = decodeWaterLevel((input.bytes[7] << 4) | (input.bytes[8] >> 4));
112     result.waterLevel = waterLevel;
113
114     warnings.push("Device notified an error on water level retrieving. Ignoring.");
115 }
116
117
118 if (!(errorState & HW080_BIT)) {
119     soilMoisture = decodeSoilMoisture(((input.bytes[8] & 0x0f) << 8) | (input.bytes[9] << 0));
120     result.soilMoisture = soilMoisture;
121
122     warnings.push("Device notified an error on soil moisture retrieving. Ignoring.");
123 }
124
125
126 return {
127     data: result,
128     warnings: warnings,
129     errors: [],
130 };
131 }
132 }
```

Figure 9: Payload formatter dans The Things Network

## Base interne à The Things Network

La base interne est un simple stockage des données décodées par le payload formatter. Il s'agit d'une simple configuration à faire sur l'application. C'est la partie la plus simple à mettre en place étant donné qu'il suffit simplement de l'activer. Les uplinks sont donc d'abord décodés, les résultats sont immédiatement stockés en base de données mais aussi transférés à l'application tierce. Voici quelques exemples de stockage dans The Things Network :

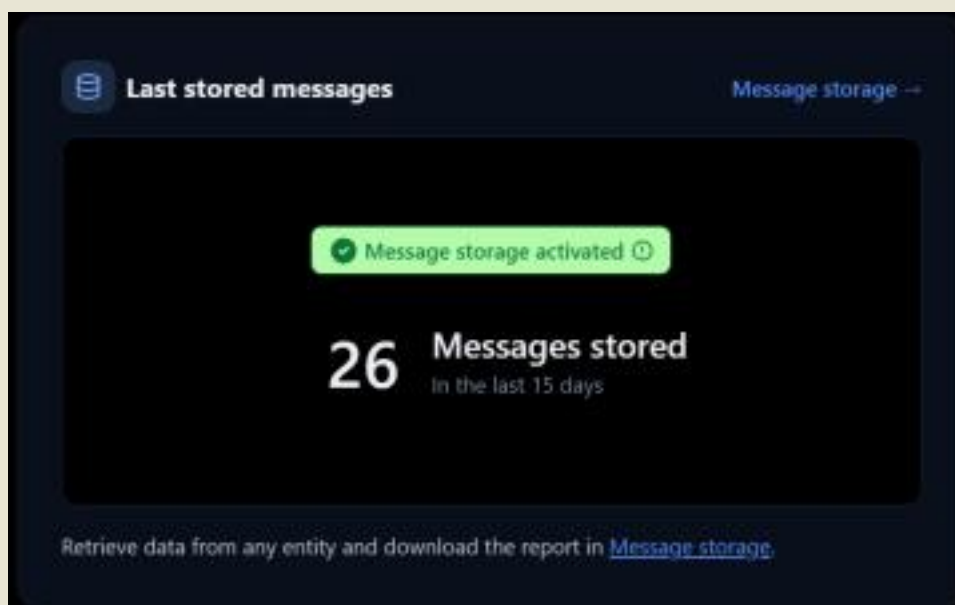




Figure 10 : Notifications de stockage des message décodés dans The Things Network



 All devices stored messages

 Last 7 days

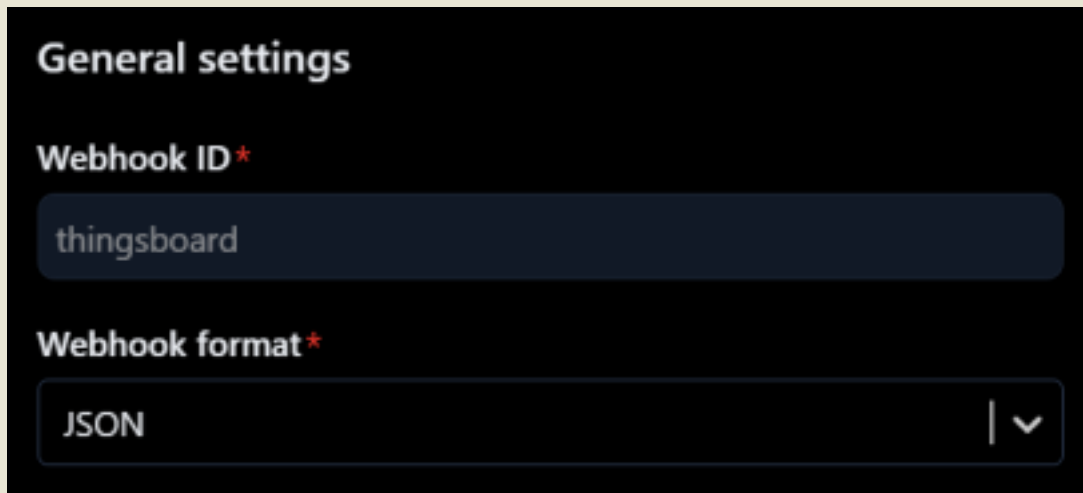
END DEVICE ID	RECEIVED	FPORT	FCNT	SEEN BY GATEWAYS	DECODED PAYLOAD
sx1262-rpi5	Apr 15, 2025 07:46:02	1		1	{ humidity: 44.140625, light: 16, soilMoisture: 46.48556876061121, temperature: 19.3125 }
sx1262-rpi5	Apr 15, 2025 08:06:08	1	1	1	{ humidity: 44.140625, light: 16, soilMoisture: 46.48556876061121, temperature: 19.3125 }
sx1262-rpi5	Apr 15, 2025 08:14:37	1		1	{ humidity: 44.140625, light: 16, soilMoisture: 46.48556876061121, temperature: 19.3125 }
sx1262-rpi5	Apr 15, 2025 08:34:42	1	1	1	{ humidity: 44.140625, light: 16, soilMoisture: 46.48556876061121, temperature: 19.3125 }
sx1262-rpi5	Apr 15, 2025 08:54:47	1	2	1	{ humidity: 44.140625, light: 16, soilMoisture: 46.48556876061121, temperature: 19.3125 }
sx1262-rpi5	Apr 15, 2025 09:14:54	1	3	1	{ humidity: 44.140625, light: 16, soilMoisture: 46.48556876061121, temperature: 19.3125 }

Figure 11 : Exemples de messages stockés dans la base interne à The Things Network

## Webhook HTTP

Lorsqu'un uplink est reçu, après la mise en base de données, il est possible de spécifier un moyen de pousser les données sur une application tierce (dans notre cas, Things Board). Ici, nous avons utilisé un webhook HTTP qui enverra une requête vers l'API de l'application tierce lorsque l'uplink est décodé. Ici, nous avons fourni l'URL d'un serveur mis en place pour Things Board. Nous utilisons donc, dans notre cas, l'API telemetry de Things Board pour envoyer les données à afficher. The Things Network a besoin d'une IP publique pour pouvoir requêter. Nous avons donc dû créer un conteneur sur un serveur appartenant à l'un de nous pour l'obtenir. Cela nous permet donc de requêter Things Board. Voici la configuration du webhook :

- URL de base : `https://<domain>/api/v1/<token du device>/telemetry`
- format du webhook : JSON
- événements activés : uplinks et downlinks



General settings

Webhook ID \*

thingsboard

Webhook format \*

JSON

Figure 12: Webhook HTTP vers Things Board sur The Things Network

Nous nous retrouvons donc avec ce morceau de schéma d'architecture pour The Things Network :

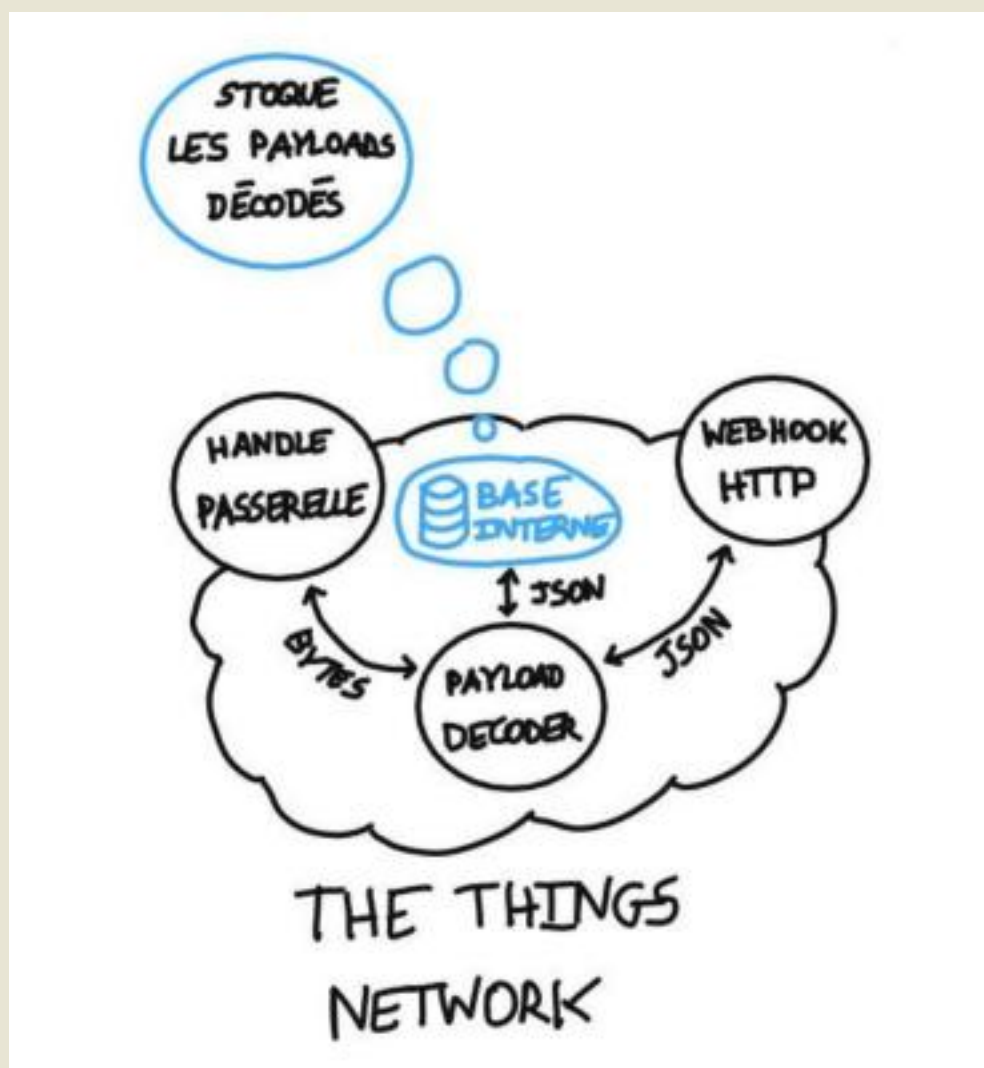


Figure 13 : Schéma d'architecture de la partie The Things Network



## Partie 4 – L’affichage sur Things Board

Things Board est une application de monitoring web permettant l’affichage de données dans plusieurs formats différents (graphes, valeurs immédiates, heat map, etc...). Dans notre cas, il s’agit d’un simple affichage des données immédiates. Il y’a quatre parties ici :

- la règle de traitement des uplinks et de création des tags
- le dispositif
- le dashboard
- les liens des tags sur le dashboard

### Règle de traitement des uplinks et de création des tags

La règle de traitement permet, comme son nom l’indique, de faire un pré-traitement des données avant d’être stockées dans le handle de dispositif sur Things Board. Il s’agit d’un système assez simple à prendre en main et relativement flexible sous format no-code. Dans notre cas, elle ne fait que stocker chaque donnée dans un tag différent pour chacune. Voici un exemple de règle créée pour le dashboard :

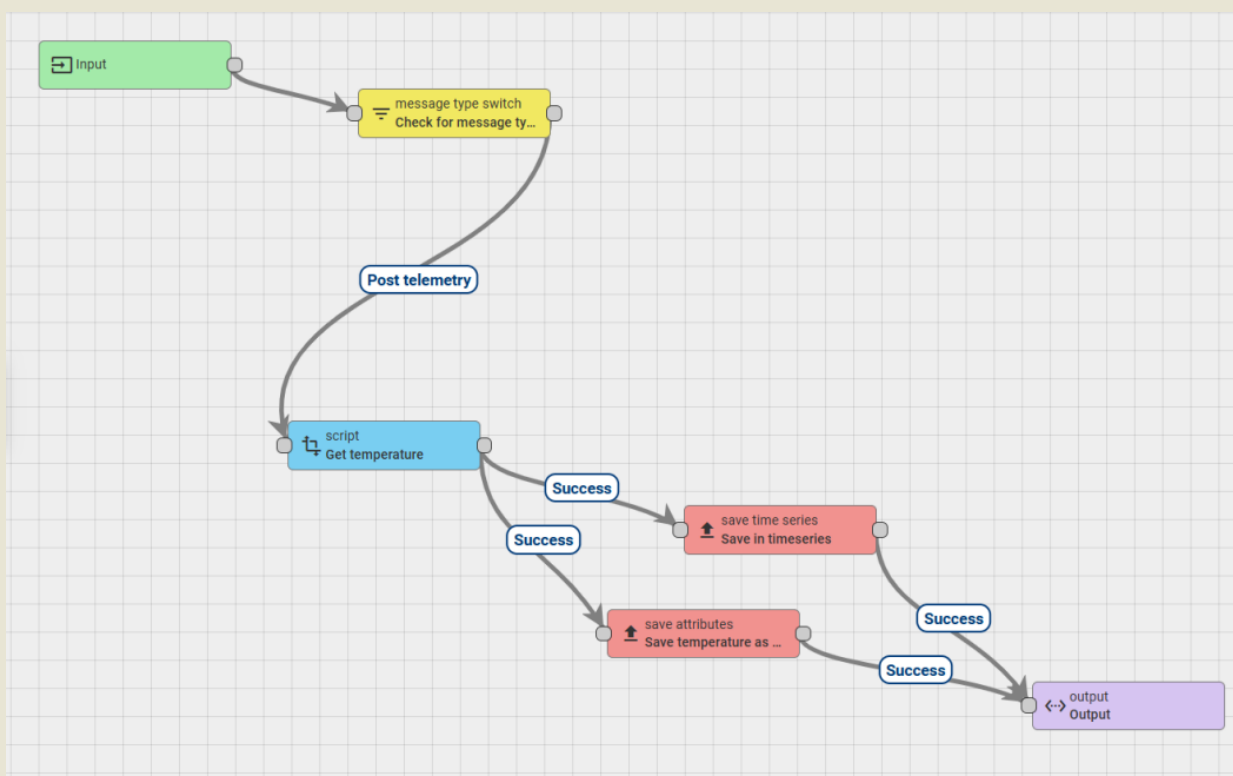


Figure 14 : Exemple de règle de traitement des données en entrées





## Dispositif

Le dispositif réceptionne les données provenant de The Things Network via le webhook HTTP entre les deux applications. Pour réceptionner des données, à la création, une route API avec un token représentant le dispositif est générée. Cette URL doit être passée dans la configuration du webhook dans The Things Network. Le dispositif sur Things Boards doit réceptionner les données afin de s'en servir dans le dashboard. Le rule engine envoie les données sous forme tags après son traitement (il garde aussi des données temps réel dans une base de données time series mais ce n'est pas important dans notre cas). Le dispositif stocke ces tags. Ces tags seront donc utilisés pour créer les affichages sur le dashboard, il s'agira donc de créer un lien vers ces tags afin d'afficher les données sous-jacentes sur l'affichage. Voici un exemple de dispositif sur Things Board :

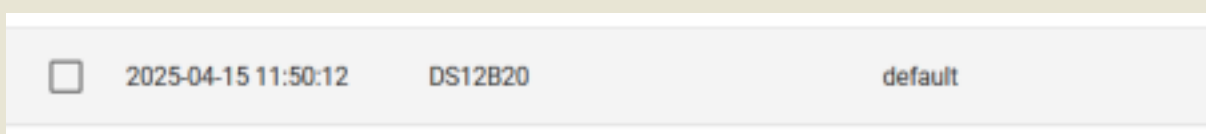


Figure 12 : Exemple de dispositif créé sur Things Board

## Dashboard

Le dashboard est un simple affichage avec beaucoup de possibilités de customisations. Dans notre cas, il s'agit d'un affichage simple, mais il est entièrement possible de créer des dashboards complexes avec une certaine interactivité. Étant donné qu'il s'agit d'un simple affichage à créer, cette partie est donc très courte. Voici un exemple léger de dashboard sur Things Board :



Figure 15 : Exemple de dashboard sur Things Board







## Liens des tags sur le dashboard

Afin d'afficher les données sur le dashboard, il suffit de cliquer sur un des widgets et de lier un tag dessus. Voici un lien de tag sur un affichage de température :

Thermometer scale

Thermometer scale

Datasource

Device\*

DS12B20

Data key\*

temperature

*Figure 16 : Exemple de lien de tag de température pour un widget*





Au final, voici le morceau du schéma d'architecture de Things Board :

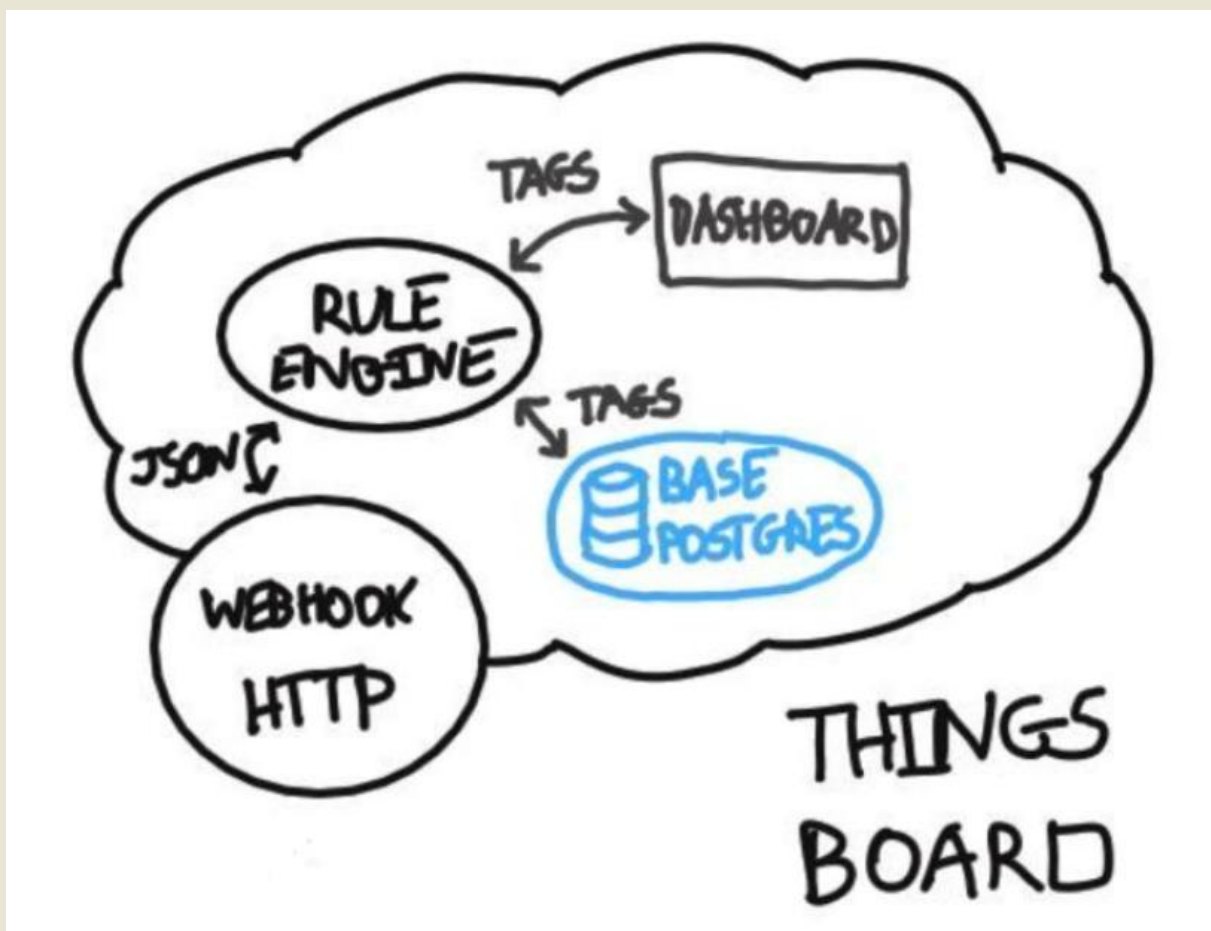


Figure 17 : Schéma d'architecture de Things Board dans le système

Ceci conclut l'explication complète du système Smart Bud. Nous avons pu apprendre beaucoup de nouvelles choses durant la période de mise en place du système et avons compris un bon grand nombre d'autres. C'était un projet très intéressant à produire. Voici le schéma complet d'architecture du système :

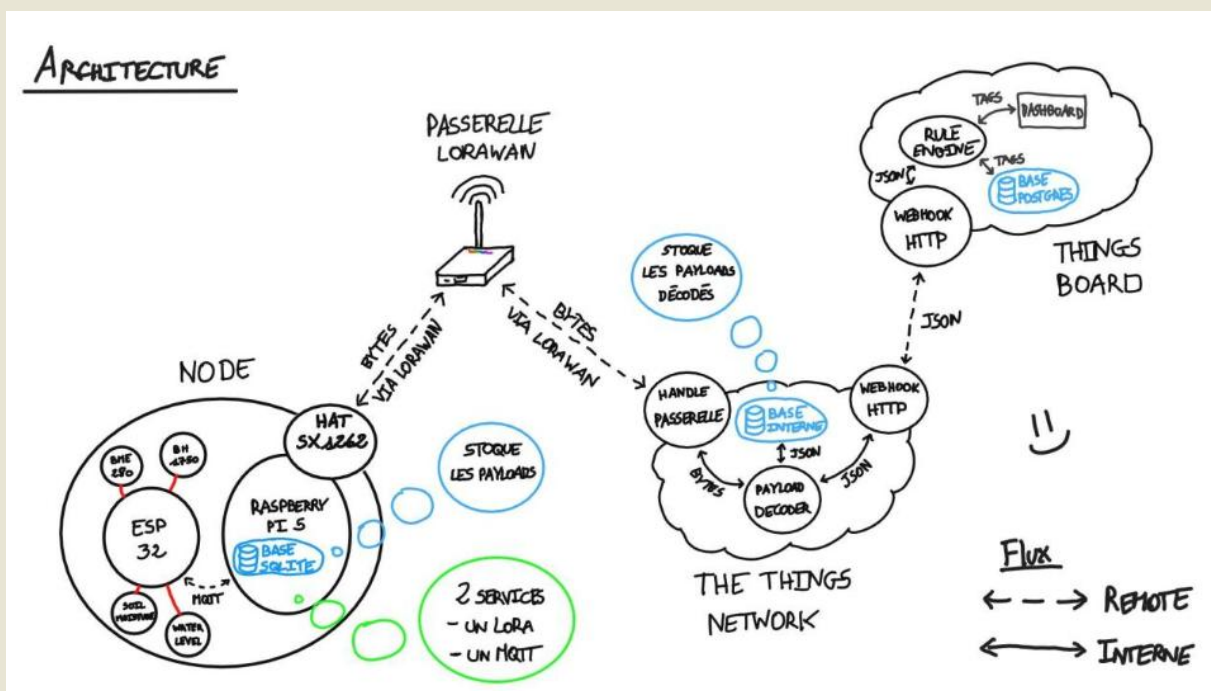
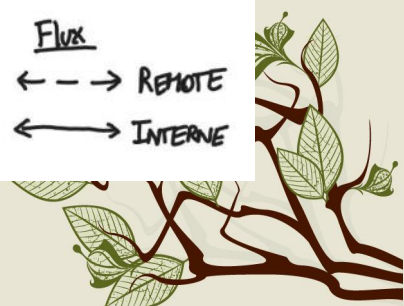


Figure 18 : Schéma complet du système mis en place sur la totalité du projet





## Description de la conception de smart-bud

Nous avons conçu notre serre de façon à ce qu'elle ne forme qu'un seul bloc afin que ce soit plus simple pour l'utilisateur de le mettre chez lui n'importe où. La partie haute est là où les plantes poussent. Il y a environ 15 cm entre la terre et le haut du plexiglas, afin de permettre une bonne pousse des plantes, et notamment celles aromatiques qui font une dizaine de centimètres en moyenne.

Ensuite, on trouve la couche de terre qui a une hauteur de 10 cm environ. Là aussi, c'est pour permettre un bon développement de la plante et lui permettre d'avoir de bonnes racines. Juste en dessous, il y a le système de drainage qui va permettre d'éviter l'accumulation d'eau au fond du bac et donc de ne pas noyer les racines. Celui-ci est en 3 parties. Tout d'abord, en contact direct avec la terre, on trouve une couche de géotextile. C'est un textile synthétique qui va laisser passer l'eau mais pas la terre. Juste en dessous, on trouve des billes d'argile qui, elles aussi, permettent le drainage de l'eau. Elles sont aussi présentes afin d'éviter que, si le géotextile casse ou si l'utilisateur l'endommage et ne s'en rend pas compte, la terre ne tombe pas directement sur la grille en dessous, où il est possible que des morceaux tombent dans le bac à eau. La grille constitue la troisième partie. Elle possède des mailles très fines afin de tout de même laisser passer l'eau, mais le moins possible la terre.

Ce surplus d'eau va se déverser dans le bac à eau et ainsi pouvoir être réutilisé, car elle ne contiendra pas de terre. Ce système est mis en place afin que l'utilisateur n'ait pas à remettre de l'eau quotidiennement dans le bac, pour être encore plus autonome. Dans ce bac, on trouve une pompe qui va permettre d'arroser les plantes en prenant l'eau qui s'y trouve. De plus, le sol de cette partie est en pente vers la pompe afin, là aussi, d'éviter que l'eau ne stagne. Enfin, tout au fond, on trouve notre électronique.

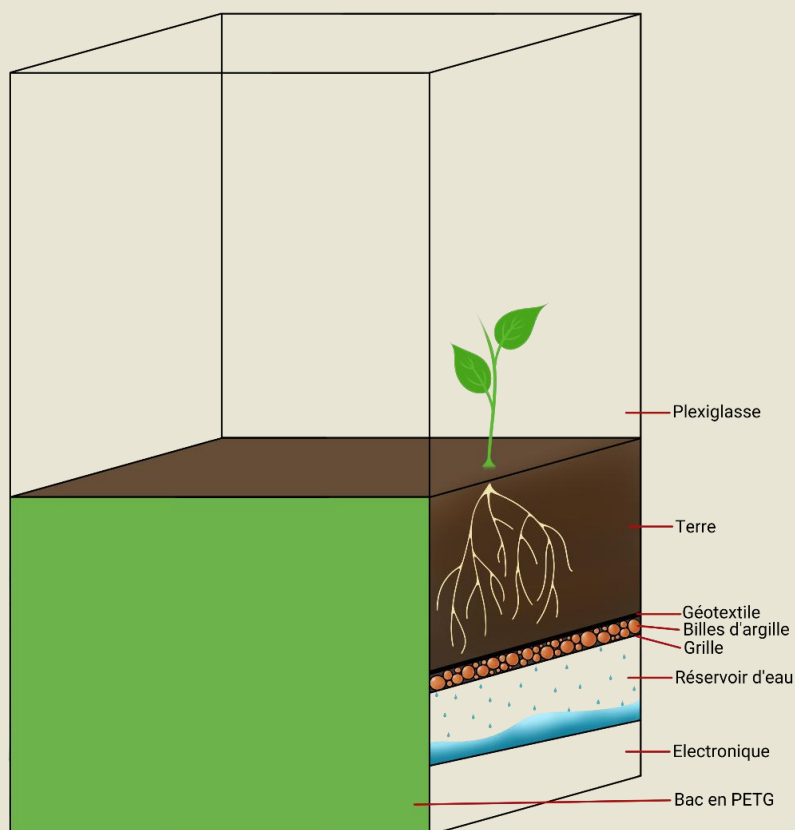


Figure 19 : Schéma du bac simplifier

