



Master 1 Technologies de l'Internet

---

# **Rapport de Projet Tutoré**

## **Amélioration de l'interface utilisateur d'une application web embarquée sur Raspberry PI**

---

Tristan Taupiac

Arnaud Chaubet

**Encadrant :**  
M. Congduc Pham

Année 2022/2023

## Table des matières

Rapport de Projet Tutoré.....	1
Introduction.....	4
Etat Initial de l'application.....	6
Le dashboard.....	6
Le Device Manager.....	8
Le Configurator.....	11
Travail réalisé.....	13
Affichage de plusieurs appareils.....	13
Bouton pour configurer.....	17

## SOMMAIRE DES FIGURES

# Introduction

Ce rapport de projet a pour but de présenter les résultats de nos travaux pour l'amélioration du projet se nommant PRIMA INTEL-IRRIS. L'objectif initial de ce projet est d'optimiser l'irrigation chez les petits exploitants agricoles grâce à une solution technologique à faible coût.

Dans ce contexte, une application web embarquée nommée IIWA a été développée en utilisant le langage de programmation Python ainsi qu'un framework nommé Flask. Cette application utilise des données de capteurs qui sont plantés dans le sol pour fournir à l'utilisateur des données sur l'état de l'irrigation de ses plantes.

Notre travail consistait à améliorer l'interface utilisateur de l'application IIWA afin qu'elle soit plus facile à utiliser pour les agriculteurs et qu'elle puisse synthétiser les informations reçues des capteurs de manière simple et efficace. Après discussions avec notre encadrant nous n'avons pas de cahier des charges définie mais juste quelque piste et les données utiliser ne proviendront pas de vrais capteur mais uniquement de données virtuelle qui émule de vrais capteur.

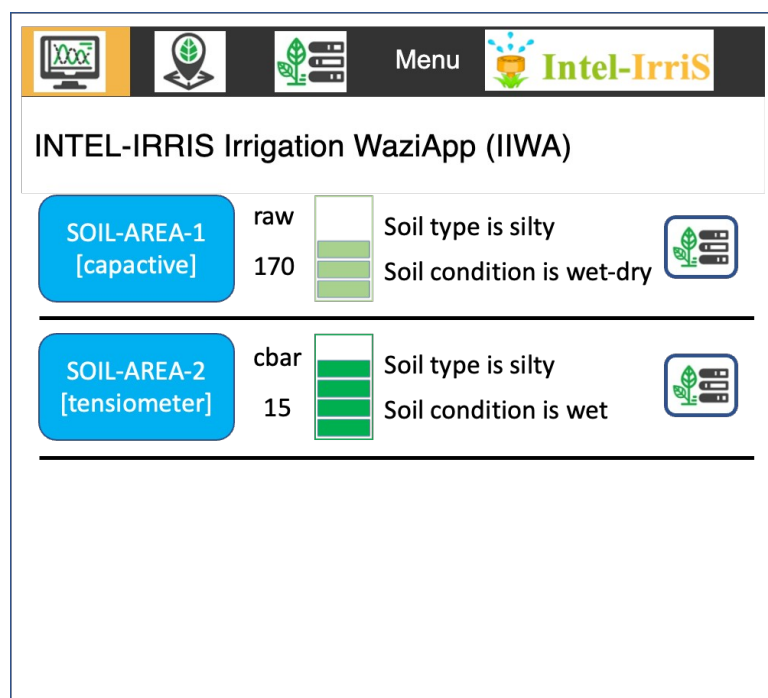


Figure 0 – Exemple d'attendue potentielle

Voici un diagramme de Gant présentant notre avancé du projet au fur et à mesure du temps

Diagramme de Gant

# Etat Initial de l'application

La première étape de notre projet a été de nous renseigner sur l'état initiale de l'application ainsi que sur sa documentation pour essayer de comprendre son fonctionnement. L'une des informations les plus importante que nous avons appris est que l'interface utilisateur doit être fait pour le format mobile.

Initialement pour la partie interface utilisateur cette application était diviser en 3 page html.

## Le dashboard

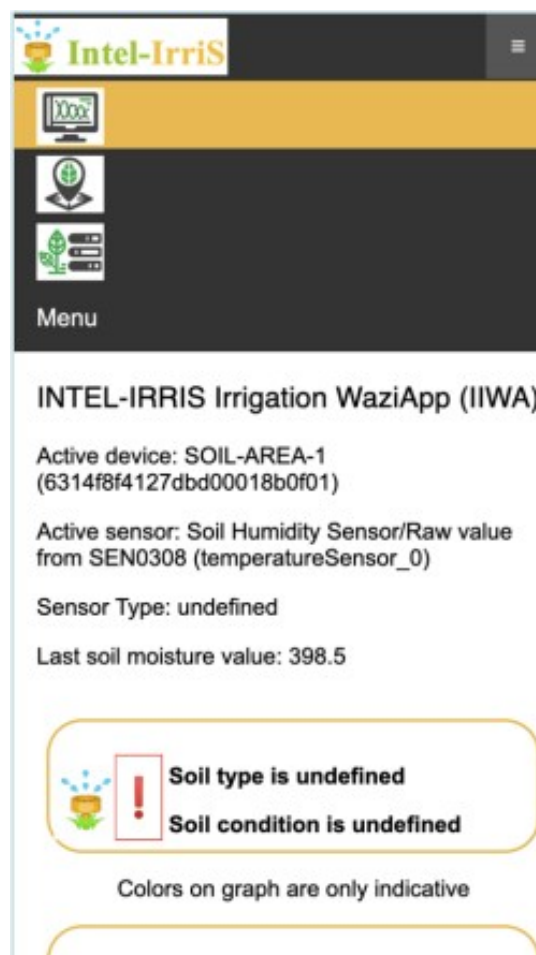


Figure 1.1.1 – Dashboard

Comme visible sur la capture d'écran ci-dessus il y a un dashboard qui montrait à la base différentes informations comme

- « Active Device » : Correspond à l'appareil qui sera affiché.

- « Active Sensor » : Correspond au capteur de l'appareil qui sera afficher.
- « Sensor Type » : Type du capteur qui peut être un capteur d'humidité capacitif ou un tensiomètre. Les 2 capteurs initialement visées par le projet IIWA sont le SEN0308 qui va être un capteur d'humidité et le WATERMARK 200SS qui va être un tensiomètre.
- « Last soil moisture value » : Une valeur d'humidité capturé par le capteur.

Les valeurs d'humidité du sol sont ensuite analysées et les informations sont présentées sous la forme d'indicateurs colorés. Ici, le rouge représente très sec, l'orange pour sec, le vert pour humide et le bleu clair pour très humide.

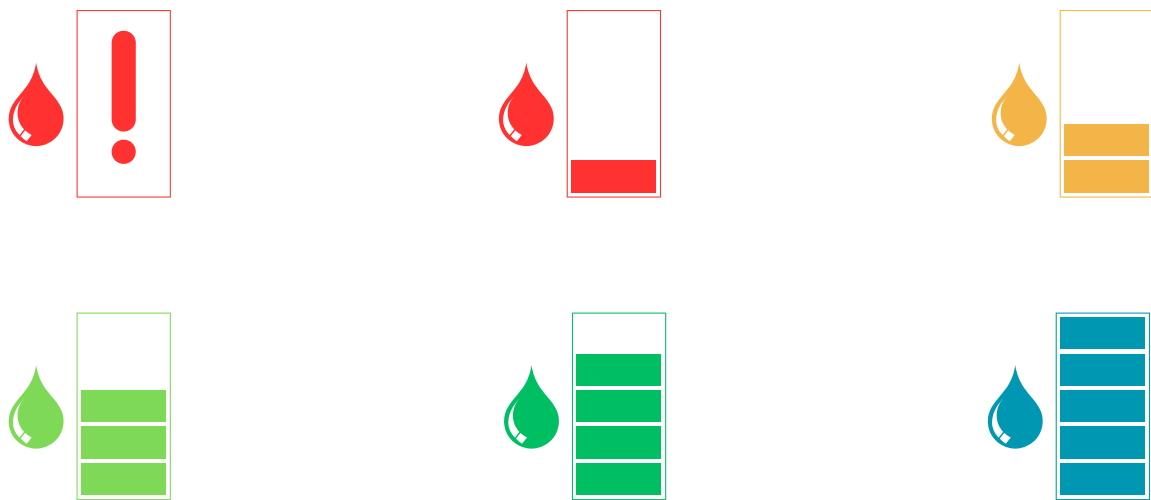


Figure 1.1.2 – Différent niveau possible

Nous pouvons aussi y trouver le type du sol et ainsi que son état (sec, semi-humide, humide).

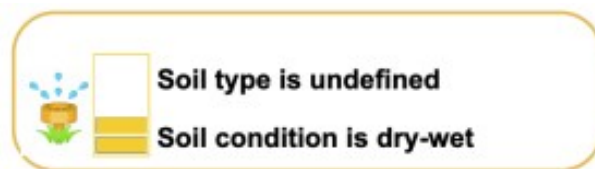


Figure 1.1.3 – Exemple d'un état du sol

Dans cette page nous avons remarquer plusieurs choses qui pouvait être refaite. Par exemple le fait qu'il n'y ait qu'un seul appareil d'afficher et qu'il faille aller le sélectionner dans une page différentes pouvait rendre l'utilisation de ce menu assez peu intuitif.

Du texte qui n'était que relativement peu précis pour l'utilisateur pouvait potentiellement le perdre. Exemple le fait d'afficher l'id de l'appareil, le `temperature_sensor0` ou autre alors que ce qui pouvait l'intéresser dans ces 2 informations n'était que le nom de l'appareil et le type du capteur.

## Le Device Manager

Il y avait à la base dans la 2eme fenêtre le « device manager », un endroit depuis lequel l'utilisateur pouvait :

- Voir la liste de tout les appareils, leur nom ainsi que les type de capteurs associer
- Sélectionner un appareil « actif », qui serait donc afficher dans le dashboard et qui pourra être configurer.
- Ajouter un nouvelle appareil en précisant le type de capteurs associer.
- Supprimer un appareils.

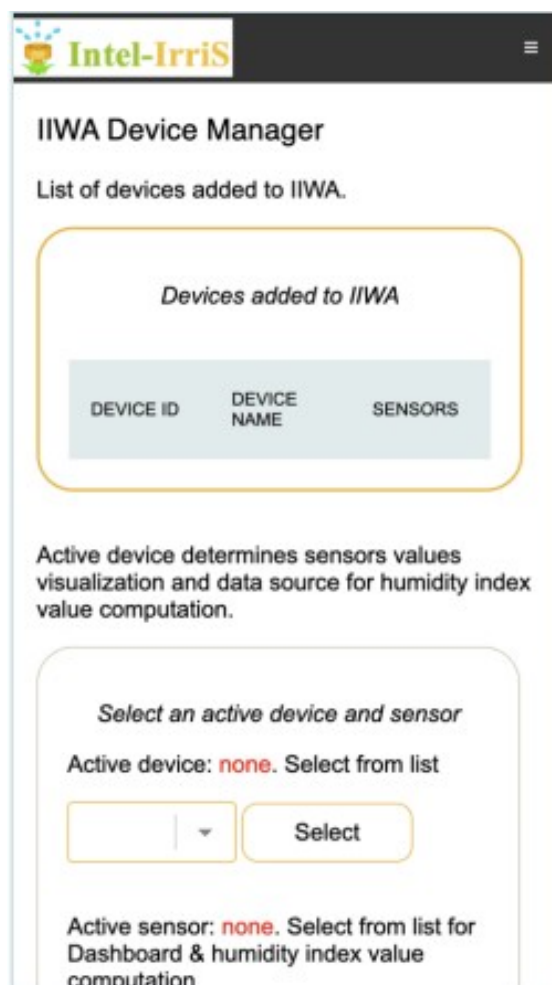


Figure 1.2.1 – Device manager initiale.



Initialement nous souhaitions modifier plusieurs choses sur cette page, par exemple lorsque l'utilisateur appuyait sur le menu en haut à droite le menu en surbrillance était forcément celui du dashboard alors que l'utilisateur ne s'y trouve pas et le nom des différentes fenêtre possible n'était pas afficher.

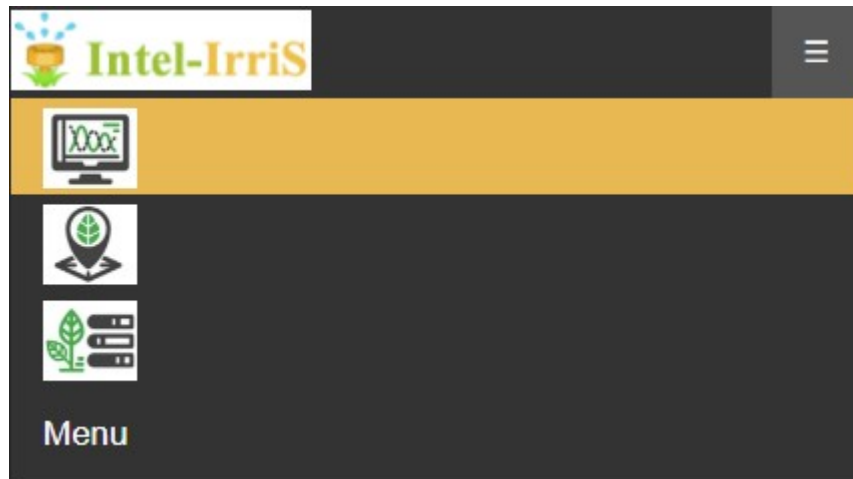


Figure 1.2.2 – Affichage initial pour sélectionner un appareil actif.

Il y avait aussi le fait que pour supprimer ou sélectionner un appareil actif ça se faisait par son ID et non par son nom. Cette méthode obligeait l'utilisateur à vérifier la liste de tout les appareils et regarder pour chaque appareil si l'id de l'appareil correspond à celui qu'il souhaitait supprimer, ce qui n'est vraiment pas intuitif.

Select an active device and sensor

Active device: SOIL-AREA-3 (643bb3061b4f9137cc406b7e). Select from list

643bb2e51b4f9137cc406b73 | ▲

Select

Active sensor: Soil Humidity Sensor/Raw value from SEN0308 (temperatureSensor\_0). Select from list for Dashboard & humidity index value computation

Soil Humidity Sensor/Raw value from SEN0308 | ▼

Select

Figure 1.2.3 – Affichage initial pour sélectionner un appareil actif.

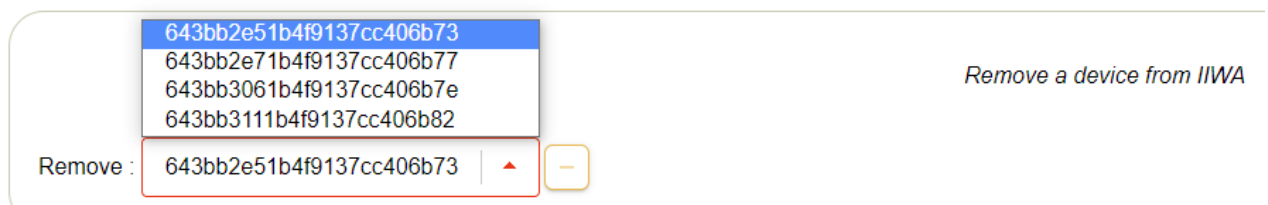


Figure 1.2.4 – Affichage initial pour supprimer un appareil.

En modifiant ces informations par directement leur nom cela nous aurait permis de modifier aussi l’affichage de la liste de tout les appareils qui montrer l’id qui n’est pas réellement une informations utile que nous aurions pu remplacer par d’autre information.

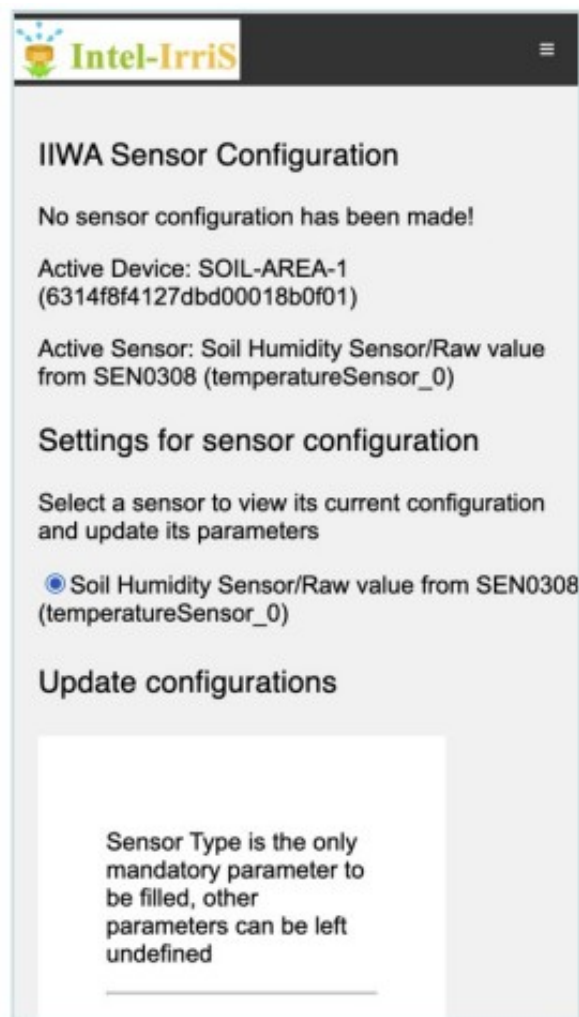
Cependant au fur et à mesure du temps nous nous somme rendu compte que cette page pouvait être supprimer et que tout pouvait être incorporé directement dans le menu dashboard.

Ce choix permettrait à l’utilisateur de pouvoir gérer l’ensemble de ses appareils depuis une seul et unique fenêtre et de ne pas avoir à naviguer plusieurs fois entre 2 page différente juste pour avoir des informations sur chacun de ses appareils. Ce choix permettrait aussi d’alléger l’application que ce soit visuellement ou physiquement et ne pas avoir à préciser un device actif.

# Le Configurator

Le configurator est une page qui comme son nom l'indique permet de configurer un appareil en modifiant plusieurs paramètres comme le type de capteur, l'âge du capteur, la région, le type de sol, le type d'irrigation, la plante/culture, la salinité du sol et la densité apparente. Toutes ces informations seront spécifiées à l'aide d'un formulaire que l'utilisateur validera.

Pour récupérer l'appareil qui devra être configuré, le configurateur va chercher l'id de l'appareil dit « actif » dans le fichier json suivant « intel-irris-active-device.json » puis va modifier les valeurs du bon capteur dans le fichier de configuration suivant intel-irris-conf.json.



The screenshot shows a web interface for "Intel-IrriS". The main heading is "IIWA Sensor Configuration". Below it, a message states "No sensor configuration has been made!". It then displays the "Active Device" as "SOIL-AREA-1" with a long hexadecimal ID. The "Active Sensor" is identified as "Soil Humidity Sensor/Raw value from SEN0308 (temperatureSensor\_0)". A section titled "Settings for sensor configuration" instructs the user to "Select a sensor to view its current configuration and update its parameters". A radio button is selected for the "Soil Humidity Sensor/Raw value from SEN0308 (temperatureSensor\_0)". Below this is an "Update configurations" button. At the bottom, a note specifies that "Sensor Type is the only mandatory parameter to be filled, other parameters can be left undefined", followed by a text input field.

Figure 1.3.1 – Informations sur les capteurs.

The image displays two side-by-side screenshots of a web application interface, likely for managing agricultural sensors and soil data. Both panels have a light gray background and a white content area.

**Left Panel: Sensor parameters**

- Sensor ID:** A text input field containing "temperatureSensor\_0".
- Sensor Type:** A group of radio buttons:
  - ☒ Capacitive
  - ☐ Tensiometer (cbar)
  - ☐ Tensiometer (raw)
- Sensor age:** A text input field containing "0".
- Min value:** A text input field containing "0".
- Max value:** A text input field containing "800".
- Section Header:** A horizontal line followed by the text "Soil parameters".

**Right Panel: Soil parameters**

- Soil Type:** A text input field containing "Clay".
- Soil Irrigation Type:** A group of radio buttons:
  - ☒ Undefined
  - ☐ Furrow
  - ☐ Sprinkler
  - ☐ Drip
- Soil Salinity:** A text input field containing "disabled".
- Soil Bulk Density:** A text input field containing "disabled".
- Section Header:** A horizontal line followed by the text "Soil temperature".
- Section Header:** A horizontal line followed by the text "Plant parameters".

Figure 1.3.2 – Affichage de quelque paramètre.

Au niveau de cette page nous n'avons pas trouver grand-chose à refaire, en soit la page nous paraissait assez complète. Nous avons tout de même penser à modifier comme pour l'exemple du device manager que lorsque la barre de navigation est ouverte c'est encore le dashboard qui est afficher en orange et que les noms ne sont pas afficher (voir Figure 1.2.2).

# Travail réalisé

La première étape que nous nous sommes donnée était de permettre l'affichage de plusieurs appareils. Par la suite nous avons permis à l'utilisateur de configurer. Pour terminer, nous nous sommes occupés de corriger plusieurs bugs et d'optimiser le code et avons permis à l'utilisateur d'ajouter ou supprimer un appareil directement depuis le dashboard et donc supprimer le « device manager ».

## Affichage de plusieurs appareils

Pour effectuer cette partie nous avons créé une liste (nommée tuple en python) `all_devices_tuple` qui contenait les différents appareils avec pour chacun leur id, leur nom, le type de capteur, l'id du capteur, l'humidité du sol, l'indicateur coloré à afficher, le type du sol ainsi que l'état du sol.

```
for i in range(length):
    if(read_devices[i]['device_id'] != 'default'):
        device_id = read_devices[i]['device_id']
        device_name = read_devices[i]['device_name']
        sensor_id = "undefined"
        sensor_type = "undefined"
        soil_moisture = 0
        value_index_file = 'images/level'+str(0)+'.png'
        value_type = "undefined"
        soil_type = "undefined"
        soil_condition = "undefined"
        if(len(sensors) > 0):
            for j in range(len(sensors)):
                if(len(sensors[j]) == 7):
                    if sensors[j][0] == device_id:
                        print("Affichage sensor: ")
                        print(sensors[j])
                        sensor_id = sensors[j][1]
                        if sensors[j][2] == "tensiometer_cbar":
                            sensor_type = "tensiometer"
                            value_type = "cbar"
                        elif sensors[j][2] == "tensiometer_raw":
                            sensor_type = "tensiometer"
                            value_type = "raw"
                        else:
                            sensor_type = sensors[j][2]
                            value_type = "raw"
                        soil_moisture = sensors[j][3]
                        value_index_file = 'images/level'+str(sensors[j][4])+'.png'
                        soil_type = sensors[j][5]
                        soil_condition = sensors[j][6]

        device = (device_id, device_name, sensor_type, sensor_id, soil_moisture, value_index_file, value_type, soil_type, soil_condition)
        all_devices.insert(i - 1, device)

no_devices = False
monitor_all_configured_sensors()
```

Figure 2.1.1 – Code qui liste tous les devices

Pour pouvoir afficher ces différents appareils nous nous sommes renseignés sur le fonctionnement du framework Flask qui permet la gestion de la partie Web de l'application.

Nous avons pu apprendre que ce framework était fait en partie à l'aide du moteur de template jinja2, ce moteur nous permet de réaliser des boucles for dans de l'html et d'y envoyer des données depuis un fichier python.

Ce que nous avons fait est donc d'envoyer les informations que nous souhaitons, c'est à dire les données des appareils.

```
if no_devices:
    return render_template("intel-irris-dashboard.html", added_devices=added_devices, no_devices=no_devices)
else:
    if (active_device_id == "undefined" or active_device_configuration == {}):
        set_default_device()
    all_devices_tuple = tuple(all_devices)
    return render_template("intel-irris-dashboard.html", no_devices=no_devices, all_devices_tuple=all_devices_tuple)
```

Figure 2.1.2 – Code qui liste tout les devices

Par la suite nous avons créé une boucle for parcourant la liste all\_devices\_tuple pour créer un affichage pour chacun des appareils existant avec pour chaque appareil tous les paramètres de celui-ci.

```
{% for device in all_devices_tuple %}
<div id="soil_parameters" class="plot_center">
  <table class="tg">
    <thead>
      <tr>
        <td id="deviceid" rowspan="2">{{ device[1] }} <br> {{ device[2] }} </td>
        <td >{{ device[6] }} </td>
        <td class="tg-c3ow" rowspan="2"></td>
        <td class="tg-0pky" colspan="2"><b>Soil type is {{ device[7] }}</b></td>
      </tr>
      <tr>
        <td id="lastvalue" colspan="2">{{ device[4] }}</td>
        <td class="tg-0pky" colspan="2"><b>Soil condition is {{ device[8] }}</b></td>
      </tr>
      <tr>
        <td>
          <a onclick="deviceConfiguration('{{ device[0] }}')">
        <td colspan="3"><!-- button to remove the device-->
      </tr>
      <tr>
        <td>
          <a onclick="deleteDevice('{{ device[0] }}')">
        <td colspan="3">
      </tr>
    </thead>
  </table>
{% endfor %}
```

Figure 2.1.3 – Code qui liste tout les devices

Ce code nous a permis d'avoir l'affichage suivant qui est l'affichage quasi final pour la partie dashboard

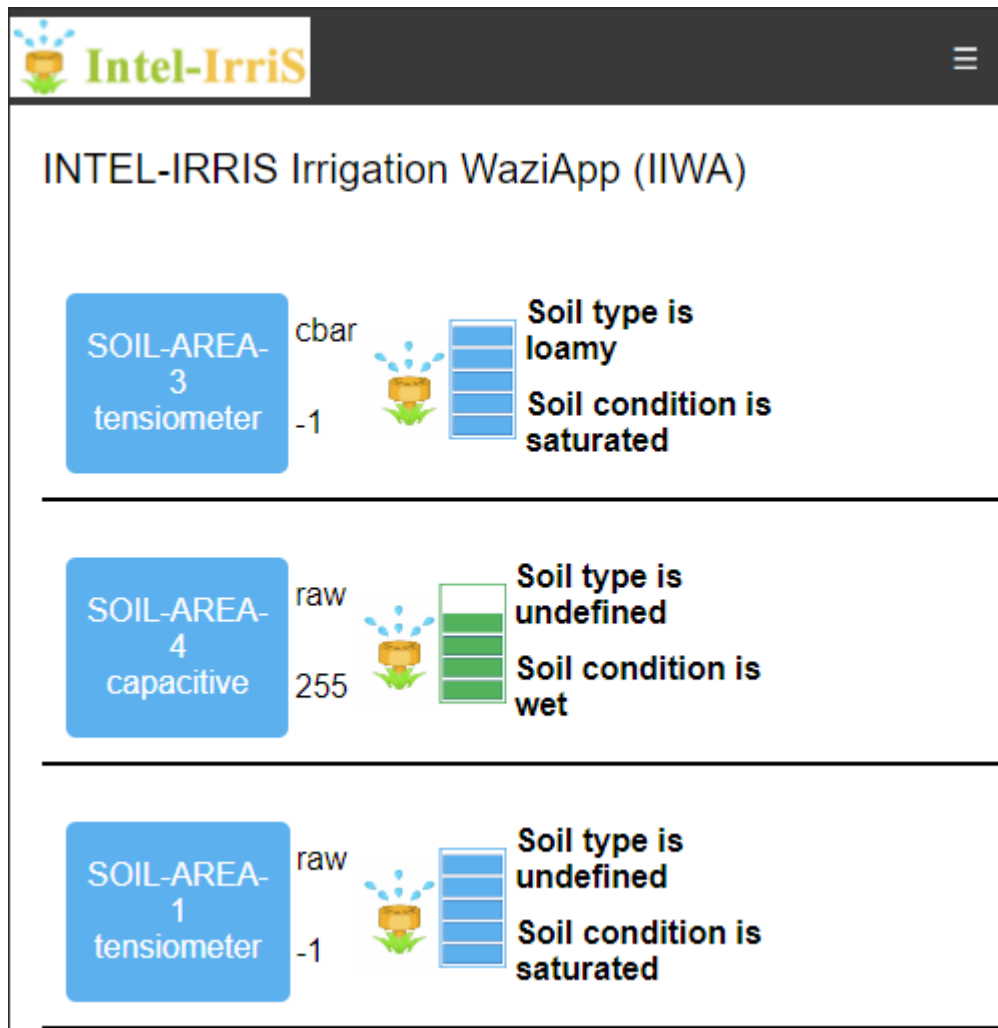


Figure 2.1.4 – Affichage avec de multiple appareils

Comme nous pouvons le voir tout les appareil sont visible, en fonction de leur ordre d'ajout et avec, pour chacun, tout leur paramètre d'inclus.

Cependant nos rajouts ont crée des bug que nous avons du patch qui sont les suivant :

Lors de l'ajout d'un appareil si la liste des capteurs était vide nous recevions une erreur, pour corriger cela nous avons rajouter un capteur par défaut pour éviter cela.

## Bouton pour configurer et supprimer

Dans un second temps nous nous sommes occupés de créer un bouton pour rediriger un appareil depuis le dashboard vers la page du configurateur pour pouvoir modifier ses paramètres.

Pour commencer nous nous sommes renseignés sur les différents moyens possibles pour réaliser ce que nous souhaitons et avons assez vite conclu que la manière la plus simple allait être de créer une fonction POST en envoyant l'id du capteur puis rediriger vers la page du configurateur si la réponse du serveur est 200.

```
function deviceConfiguration(device_id) {
  console.log(device_id);
  fetch(`${window.origin}/device-configuration`, {
    method: "POST",
    credentials: "include",
    body: JSON.stringify(device_id),
    cache: "no-cache",
    headers: new Headers({
      "content-type": "application/json"
    })
  }).then(response => {
    if(response.status == 200) {
      window.location.href = "{{url_for('intel_irris_sensor_config')}}";
    }
    else {
      // Handle error if needed
      console.error("Failed to delete device");
    }
  }).catch(error => {
    // Handle any network or other errors
    console.error("Failed to delete device", error);
  });
}
```

Figure 2.2.1 – fonction POST pour configurer un appareil



Par la suite ce poste sera traiter dans le fichier python qui va charger le fichier json intel-irris-active-device.json qui contient l'id de l'appareil qui sera configurer et va le modifier par l'id envoyer par l'html.

```
@app.route("/device-configuration", methods=['POST'])
def device_configuration():
    global selected_device_id
    data = request.get_json()
    print("Device id returned by the HTML: " + data)
    selected_device_id = data

    # charger le JSON à partir de active_device_filename
    with open(active_device_filename, 'r') as f:
        jsonval = json.load(f)

    # modifie l'ID du device
    jsonval[0]['device_id'] = data

    # enregistre la modif dans le fichier
    with open(active_device_filename, 'w') as f:
        json.dump(jsonval, f)
    # mettre temps de chargement car l'update prend du temps
    return("")

#-----#
```

Figure 2.2.2 – Fonction POST dans le python lié au bouton pour config

Le bouton suivant a été ajouter dans la boucle for de l'html (voir fig 2.1.3) pour représenter la redirection vers le configurateur.



Figure 2.2.3 – Bouton configurateur

Pour supprimer un device une méthode similaire a était utiliser, envoyant lui aussi l'id de l'appareil au python.

Le code utilisait dans la partie python est celui qui était déjà présente dans le programme initialement.

```

function deleteDevice(device_id) {
  console.log(device_id);
  fetch(`${window.origin}/delete-device`, {
    method: "POST",
    credentials: "include",
    body: JSON.stringify(device_id),
    cache: "no-cache",
    headers: new Headers({
      "content-type": "application/json"
    })
  }).then(response => {
    // Handle the response from the server
    if (response.ok) {
      // Device deleted successfully, update the UI or perform any other actions
      console.log("Device deleted successfully");
      window.location.href = "{{url_for('dashboard')}}";
    } else {
      // Handle error if needed
      console.error("Failed to delete device");
    }
  }).catch(error => {
    // Handle any network or other errors
    console.error("Failed to delete device", error);
  });
}

```

Figure 2.2.4 – Fonction POST de l'HTML pour supprimer un device

```

@app.route("/delete-device", methods=['POST'])
def delete_device():
    try:
        # Get the device_id from the request data
        idtodelete = request.get_json()
        print("Device id returned by the HTML: " + idtodelete)
        request_removed_from_devices = False
        request_removed_from_active = False
        request_removed_from_sensors = False

        # ** first remove id from devices **
        if (not request_removed_from_devices):
            # 1. Read file contents
            with open(added_devices_filename, "r") as file:
                read_data = json.load(file)
                no_devices = len(read_data)

            # 2. Check index of submitted device id
            for x in range(0, no_devices):
                if read_data[x]['device_id'] == idtodelete:
                    requested_removal_exists = True
                    print("requested device_id to remove is valid...")
                    # remove device id and name
                    read_data.pop(x)
                    print("new devices data to be saved in config = %s" %
                          read_data)

            # 3. Write json file
            with open(added_devices_filename, "w") as file:
                json.dump(read_data, file)
            print("Device list updated!")
            request_removed_from_devices = True
            break
    
```

Figure 2.2.5 – Une des partie du POST dans le python.

Pour la suppression d'un appareil le logo suivant a été ajouter.



Figure 2.2.6 – Bouton pour supprimer

Ce qui nous donne cet affichage, qui sera l'affichage final pour la partie appareil de la section dashboard.

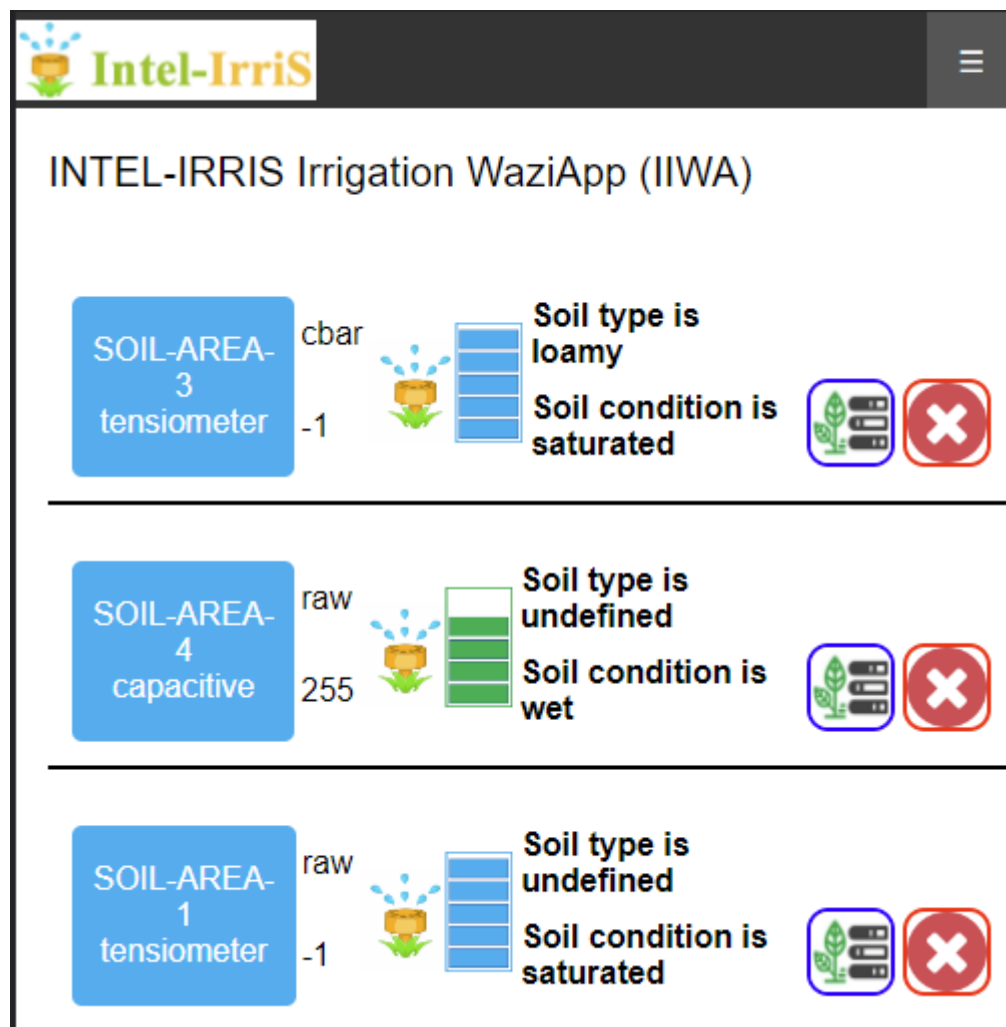


Figure 2.2.7 – Affichage finale de la partie appareil du dashboard

# NavBar & Device Manager

Comme dit lors de la présentation du Device Manager (voir page 8) la navbar n'affichait pas le nom des différentes page possible et ne changer pas la couleur en fonction de celle ci.

Pour changer la couleur il a suffi de changer la classe « active » qui était une classe associée à chaque fois à l'ancre (<a> en HTML) du dashboard et l'attribuer à celle de la page correspondante.

Pour afficher le nom de la page une balise <p> associée à une classe « show-on-mobile » a été créée pour chacun des choix possible.

```
<p class="show-on-mobile" >Dashboard</p> </a>  
0px"><p class="show-on-mobile" >Device Manager</p></a> -->  
<p class="show-on-mobile" >Configurator</p></a>
```

Figure 2.3.1 – Code dans l'HTML

```
.topnav.responsive .show-on-mobile {  
  display: block;  
  display: inline;  
  margin: 0 0 0 10px;  
}
```

Figure 2.3.1 – Code dans le CSS

La classe topnav.responsive dans le CSS est une classe déjà présente dans le CSS de base qui ne s'affiche que si la personne est sur mobile. L'application étant destinée à une utilisation sur mobile nous avons décidé de faire que le texte ne s'affiche que si l'utilisateur est sur mobile.



Figure 2.3.1 – Affichage final de navbar

à rajouter :

Add a device using the form below.

*Add a device to IIWA*

Select a device by name

-- Sensor(s) Structure --

+

- parler des bug existant qui pourrait être patch
- dire qu'on a desactiver le device manager car inutile désormais

## Conclusion

## Bibliographie