



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 nommer 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.

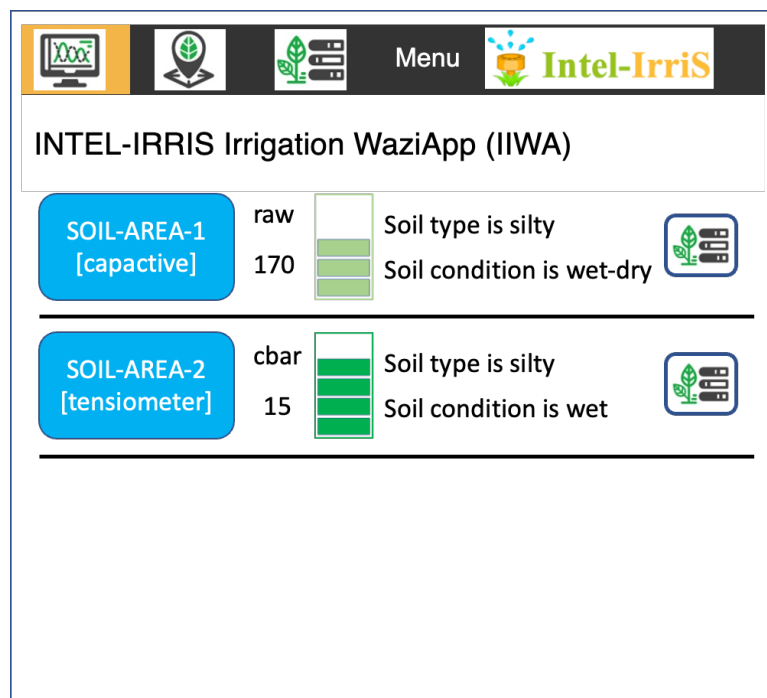


Figure 0 –

Exemple

d'attendue potentielle

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

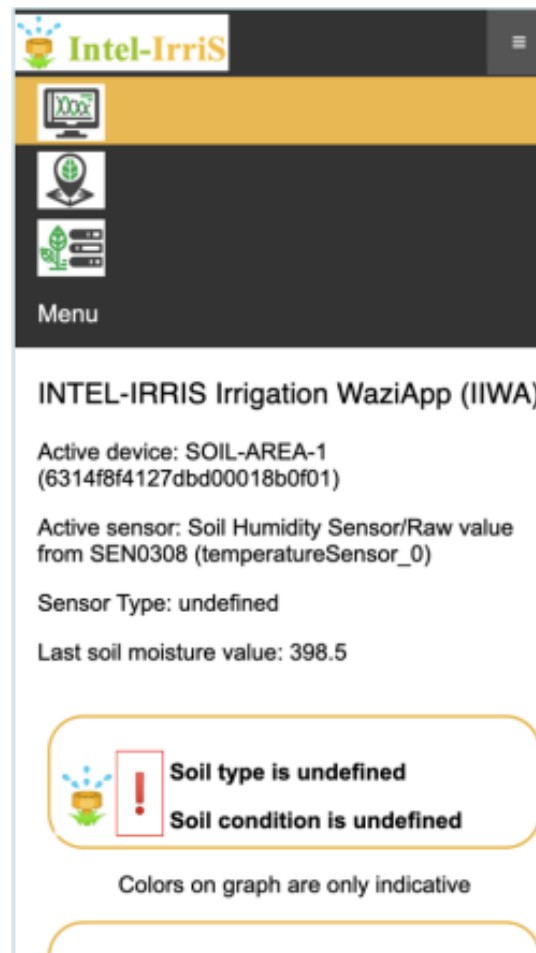


Figure 1.1.1 – Dashboard

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 affiché.
- « Sensor Type » : Type du capteur qui peut être un capteur d'humidité capacitif ou un tensiomètre. Les 2 capteurs initialement visés 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 capturée 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.

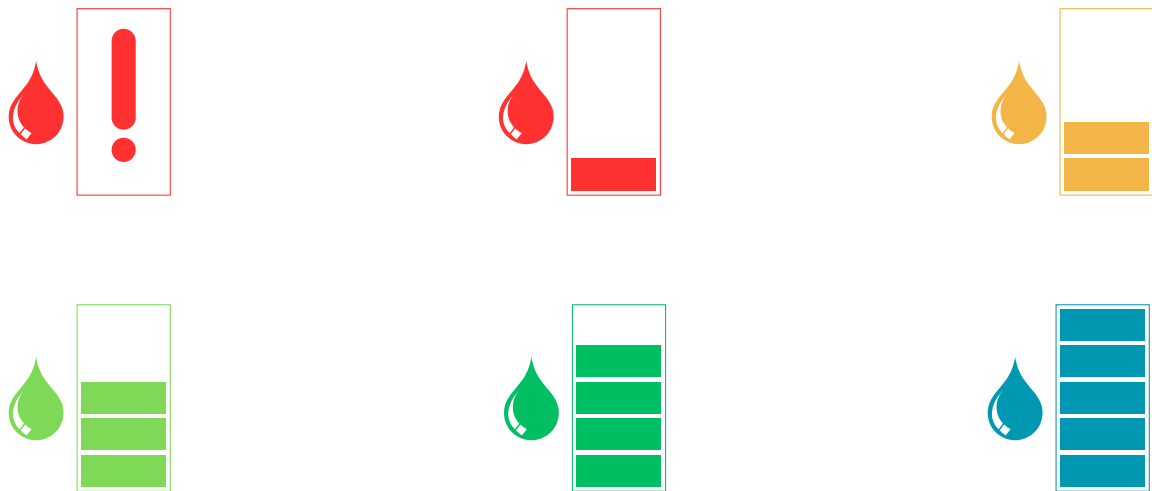


Figure 1.1.2 – Différent niveau possible

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

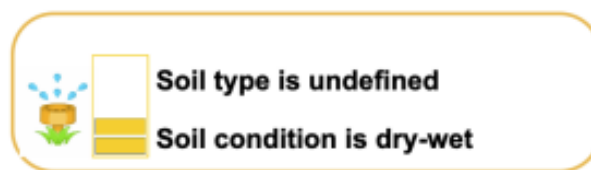


Figure 1.1.3 – Exemple d'un état du sol

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 allez 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.

Intel-IrriS

IIWA Device Manager

List of devices added to IIWA.

Devices added to IIWA

| DEVICE ID | DEVICE NAME | SENSORS |
|-----------|-------------|---------|
|-----------|-------------|---------|

Active device determines sensors values visualization and data source for humidity index value computation.

Select an active device and sensor

Active device: **none**. Select from list

▼

Select

Active sensor: **none**. Select from list for Dashboard & humidity index value computation

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.

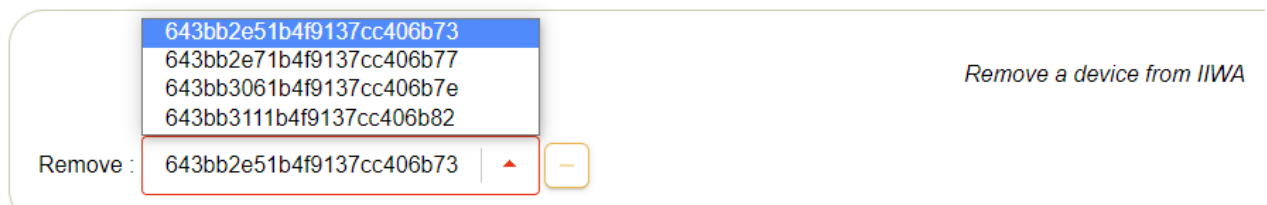


Figure 1.2.4 – Affichage initial pour supprimer un appareil.

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 comme le nom.

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 ».

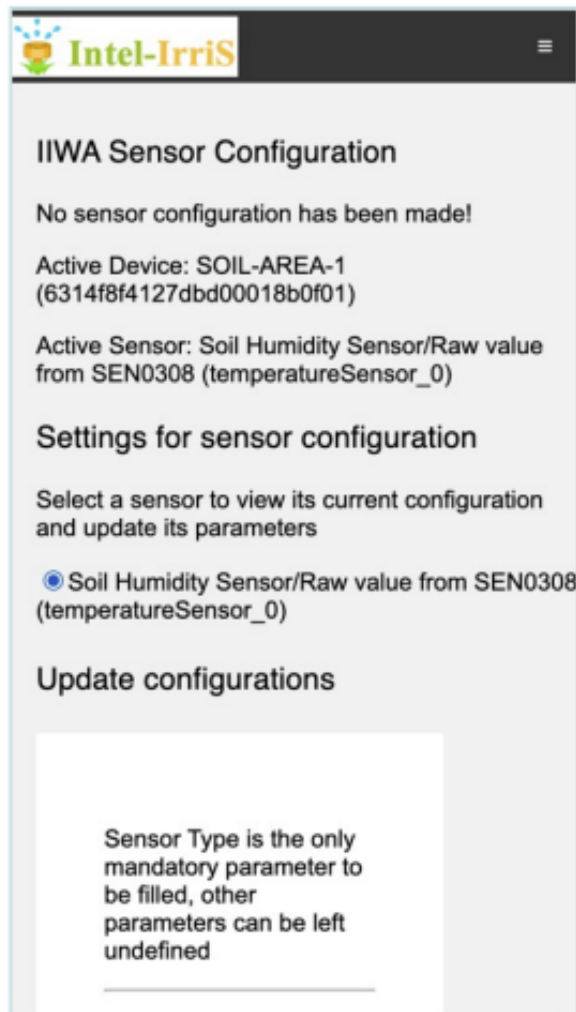


Figure 1.3.1 –

Informations sur les

capteurs.

Sensor parameters

temperatureSensor_0

Sensor Type

☒ Capacitive
☐ Tensiometer (cbar)
☐ Tensiometer (raw)

Sensor age

0

Min value

0

Max value

800

Soil parameters

Soil parameters

Soil Type

Clay

Soil Irrigation Type

☒ Undefined
☐ Furrow
☐ Sprinkler
☐ Drip

Soil Salinity

disabled

Soil Bulk Density

disabled

Soil temperature

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 sois la page nous paraissait assez complète.

Prévision et méthode de travail.

Pour prévoir nos avancement par rapport au objectif que nous nous sommes fixé nous avons décidé d'établir un diagramme de Gant. Ce diagramme nous a permis d'avoir une certaine ligne à suivre pour l'avancement du projet et de ses fonctionnalité pour que la finalité correspondante au attente du client.

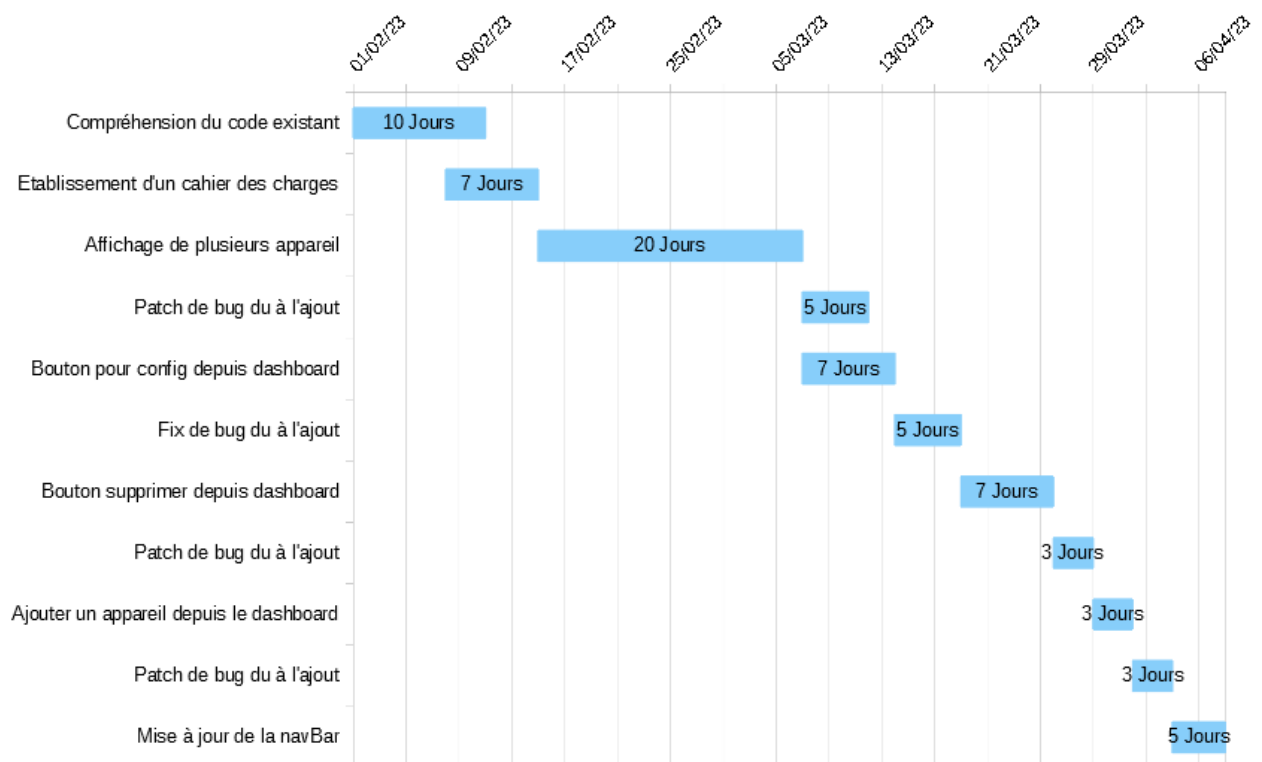


Figure 2.1.1 – Diagrammes de Gant initiale

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 3.1.1 – Code qui liste tout les devices

Pour pouvoir afficher ces différent appareil nous nous sommes renseigné 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 boucle 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 3.1.2 – Code qui liste tout les devices

Par la suite nous avons crée une boucle for parcourant la liste all_devices_tuple pour crée un affichage pour chacun des appareils existant avec pour chaque appareils tout les paramètre 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 colspan="4">
          <a onclick="deviceConfiguration('{{ device[0] }}')">
      </tr>
      <tr>
        <td colspan="4">
          <!-- button to remove the device-->
          <a onclick="deleteDevice('{{ device[0] }}')">
      </tr>
    </thead>
  </table>
{% endfor %}
```

Figure 3.1.3 – Code qui liste tout les devices

Avec l'ajout d'un peu de CSS ce code nous a permit d'avoir l'affichage suivant qui est l'affichage quasi final pour la partie dashboard

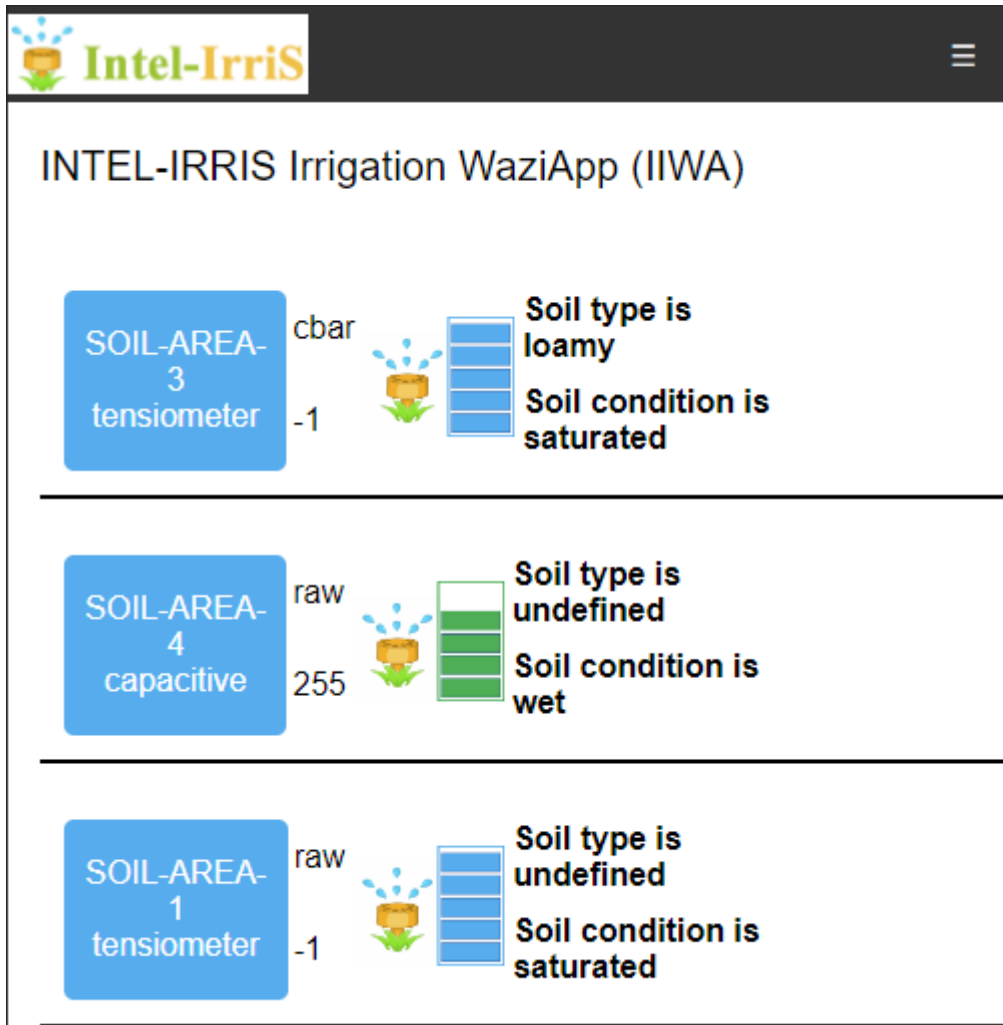


Figure
3.1.4 –

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 un bug que nous avons du patch qui est le suivant :

Lors de l'ajout d'un appareil si aucun appareil n'était sélectionner initialement, pour corriger cela nous avons rajouter un capteur par défaut. Ça permet aussi donc d'avoir un device à configurer par défaut dans le configurateur.


```
def set_default_device():
    global active_device_id
    # Set the first device as default device
    active_device_id = all_devices[0][0]
    active_sensor_id = all_devices[0][3]
    # Then, update the active-device json
    active_device_sensor_dict = [{
        'device_id': active_device_id,
        'sensor_id': active_sensor_id
    }]
    # convert python dict to JSON
    jsString = json.dumps(active_device_sensor_dict)
    jsFile = open(active_device_filename, "w")
    jsFile.write(jsString)
    jsFile.close()
    print("Successfully updated active sensor id!")
# -----#
```

Figure
un
défaut

3.1.5 – Définie
capteur par

Comme visible sur la capture d'écran ci dessus la méthode utiliser est assez simple, nous récupérons l'id du premier appareil ainsi que l'id de son premier capteur. Juste après nous les attribuons au device « actif »

Bouton pour configurer et supprimer

Dans un second temps nous nous somme occuper de crée un bouton pour rediriger un appareil depuis le dashboard vers la page du configurateur pour pouvoir modifier ses paramètre.

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 3.2.1 – fonction POST pour configurer un appareil

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

#-----#

3.2.2 –

Fonction POST dans le python lié au bouton pour config

Le bouton suivant a été ajouté 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 été utilisée, envoyant lui aussi l'id de l'appareil au python.

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

```

function deleteDevice(device_id) {
  console.log(device_id);
  if (confirm("Are you sure you want to delete this device?")) {
    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 3.2.4 – 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
    
```

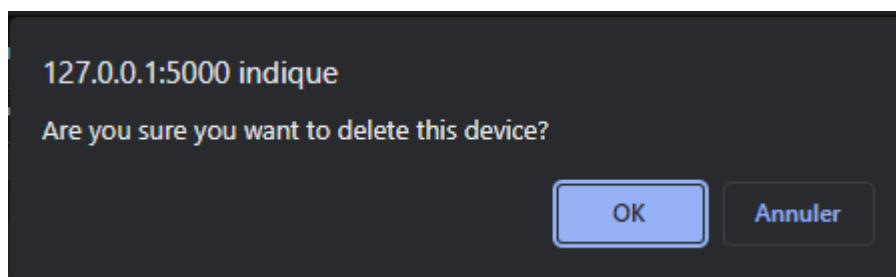
Figure 3.2.5 – Une des partie du POST dans le python.

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



Figure 3.2.6 – Bouton pour supprimer

Après discussions avec notre encadrant nous nous somme rendue compte que nous avions oublier d'ajouter une « alert » permettant de valider ou non la suppression d'un appareils. Nous l'avons donc rajouter ce qui nous affiche ceci avant de supprimer un appareils.



Figure

3.2.7 –

Message de confirmations

Nous avons hésitez à rajouter une alerte plus « esthétique » et donc rendre l'application plus accueillante. Cependant l'application étant destiné à être assez légère et fonctionnelle sur différent type d'appareil mobile nous avons décider de garder l'alerte HTML de base.

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 3.3.1 – Code dans l'HTML

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

Figure 3.3.2 – 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

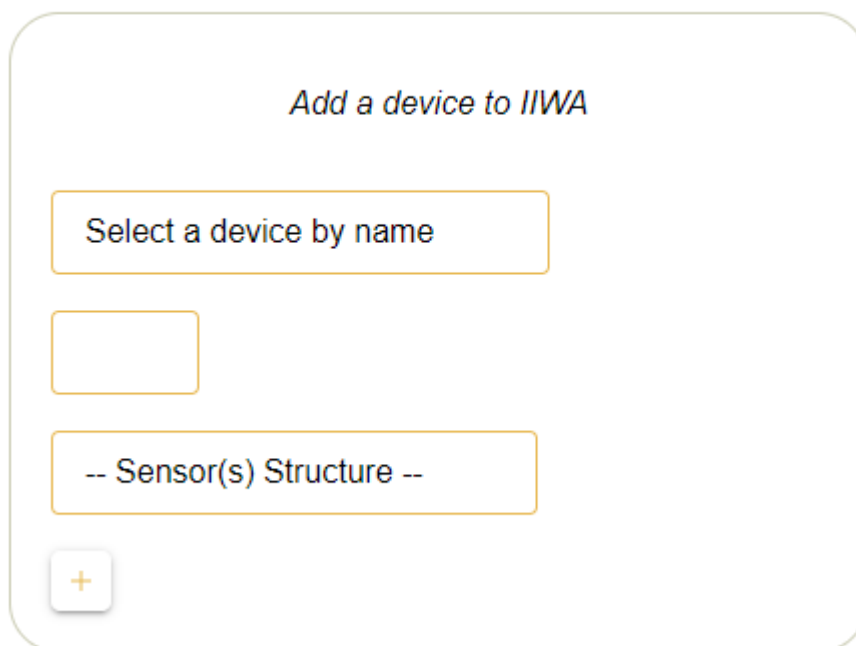
3.3.3 – Affichage final de navbar

Comme vous avez pu le remarquer sur la capture d'écran juste au dessus, nous avons décidé de ne plus afficher le device manager. Cependant pour des questions de développement et potentiellement de debugage nous avons décidé de laisser le code dans le programme initiale et donc, si l'utilisateur rentre le bon URL, il pourra avoir accès au device manager et à ses fonctionnalité.

Affichage Finale du Dashboard

à rajouter :

Add a device using the form below.

A screenshot of a web form titled "Add a device to IIWA". The form is enclosed in a light green rounded rectangle. It contains three main input fields: a dropdown menu labeled "Select a device by name", a text input field, and another dropdown menu labeled "-- Sensor(s) Structure --". At the bottom left of the form is a small square button with a plus sign (+).

- parler des bug

existant qui pourrait être patch

- Compléter diagramme de Gant
- Autre diagramme de Gant pour montrer notre avancé réellement

Ce qui nous donne cet affichage, qui sera l'affichage final pour le dashboard.

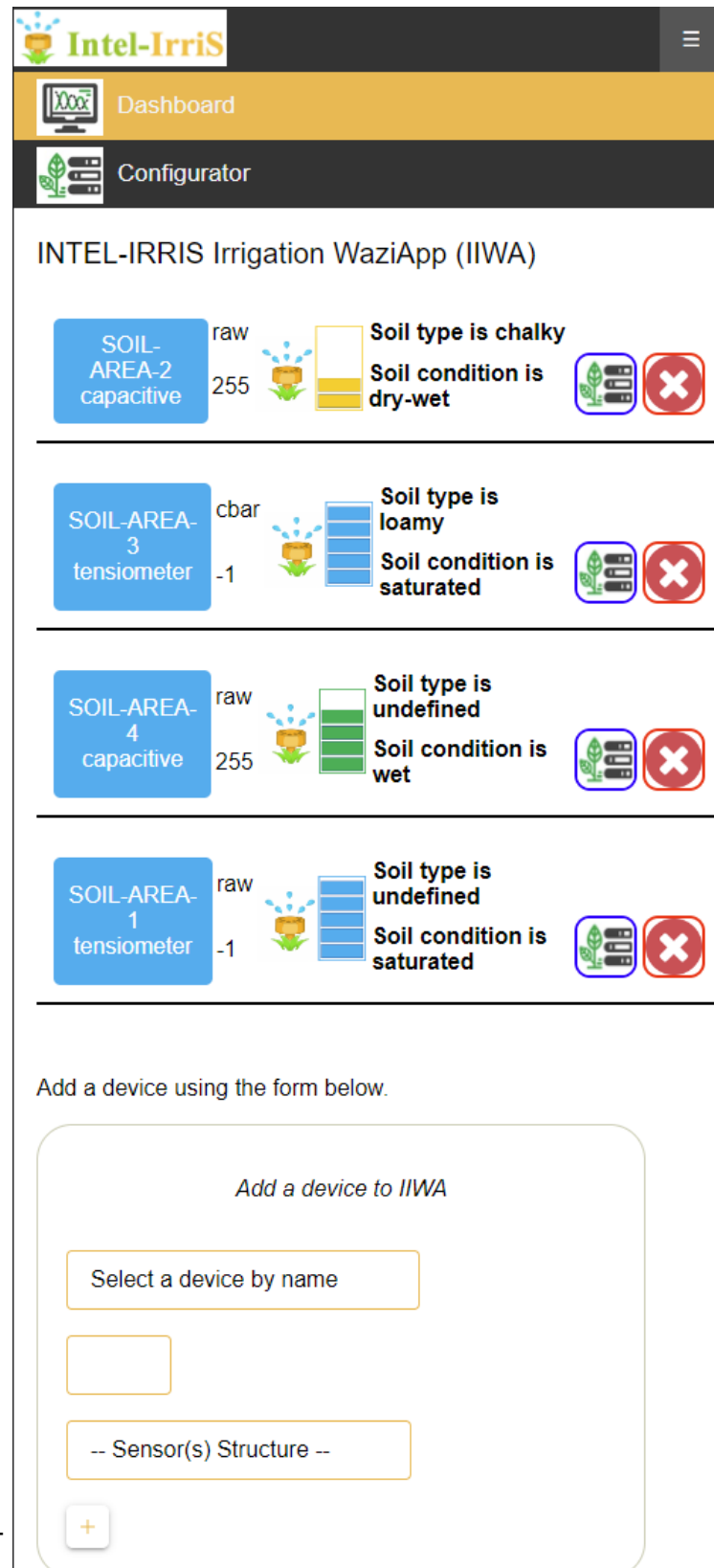


Figure 2.2.7 –
du dashboard

Affichage finale

(Le odt a tout cassé chez moi j'ai le nom de figures mal mises et des espaces en trop)
Documentation pour Linux (bon pour le coup je crois que j'explique un peu trop):

Lors de la mise en place des différentes dépendances du projets, nous nous sommes rendu compte qu'une documentation pour les systèmes d'exploitations basés sur Linux était absente (la documentation macOS était utilisé pour l'installation sur les systèmes basés sur Linux)

MacOS instructions (and Linux?)

Cependant, la mise en place de ces dépendances sous Linux est différente de la mise en place sous les systèmes Windows et macOS, nous avons donc rajouté une documentation (disponible dans le dossier build-local puis linux_instructions.md) pour les systèmes basés sur Linux (en reprenant celle de macOS car certains points se ressemblent), pour la rédaction de cette documentation, nous avons choisis d'expliquer comment faire en utilisant la distribution Debian car c'est une distribution qui est notamment utilisé comme base de plein d'autres distribution tel qu'Ubuntu qui est très populaire ou Raspbian qui est la version Raspberry Pi de Debian.

La première différence se situe au niveau du port à utiliser, sous macOS, il était impératif de le modifier car celui-ci était utilisé par une application intégré au système macOS, or sous Linux cette application n'est pas présente, il n'est donc pas nécessaire de le modifier.

La deuxième différence avec macOS est l'installation de Go et surtout de MongoDB qui diffère car la version de MongoDB disponible dans le gestionnaire de paquet apt est supérieure à la version 5.0 et il n'est pas possible de sélectionner une version antérieure, il faut donc la télécharger depuis le site web de MongoDB et de l'installer manuellement:

```
Install WaziEdge and its dependencies. Only MongoDB up to v5.0 is supported.
---
> sudo apt install golang-go

Download MongoDB Community Server from https://www.mongodb.com/try/download/community, select the correct platform/architecture and
select the server package, then install the package with dpkg:

> sudo dpkg -i mongodb-org-server_5.0.x_arch.deb (with x corresponding to the subversion number and arch=amd64 or arm64)
```

La troisième différence se situe au niveau du lancement de mongod et de wazigate-edge qui demandent des droits administrateur (utilisation de la commande sudo):

```
Run the IIWA local instance
---

We will start MongoDB and wazigate-edge. In one terminal window:

> sudo mongod --config /etc/mongod.conf &
> cd wazigate-edge
> sudo ./wazigate-edge

In another terminal, start the IIWA application:

> cd intel-irris-waziapp-local
> . iiwa/bin/activate
> python3 app.py

Then open http://127.0.0.1:5000/ on your host computer's web browser.
```

Conclusion

bla bla bla dire qu'il faudrait retour de client pour amélioration, dire que ça convient à notre encadrant que ct intéreasnt etc...

Bibliographie

teclado, 9 octobre 2020, "How to display dynamic data tables with Python, Flask, and Jinja2"
Youtube, <https://www.youtube.com/watch?v=mCy52I4exTU>