



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
<b>Introduction.....</b>	<b>3</b>
<b>Etat Initial de l'application.....</b>	<b>4</b>
Le dashboard.....	4
Le Device Manager.....	6
Le Configurator.....	9
<b>Avancée prévisionnelle et méthode de travail.....</b>	<b>11</b>
<b>Travail réalisé.....</b>	<b>13</b>
Réalisation de la documentation.....	13
Affichage de plusieurs appareils.....	15
Bouton pour configurer et supprimer.....	19
NavBar & Device Manager.....	23
Ajout d'appareil depuis le dashboard et Affichage Final.....	24
<b>Avancée réelle et apprentissage personnel.....</b>	<b>26</b>
<b>Conclusion.....</b>	<b>28</b>
<b>Bibliographie.....</b>	<b>29</b>

# 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 une discussion avec notre encadrant nous n'avions pas de cahier des charges défini mais quelques pistes, ainsi que l'information que les données utilisées ne proviendront pas de vrais capteurs mais uniquement de données virtuelles qui émulent de vrais capteurs.

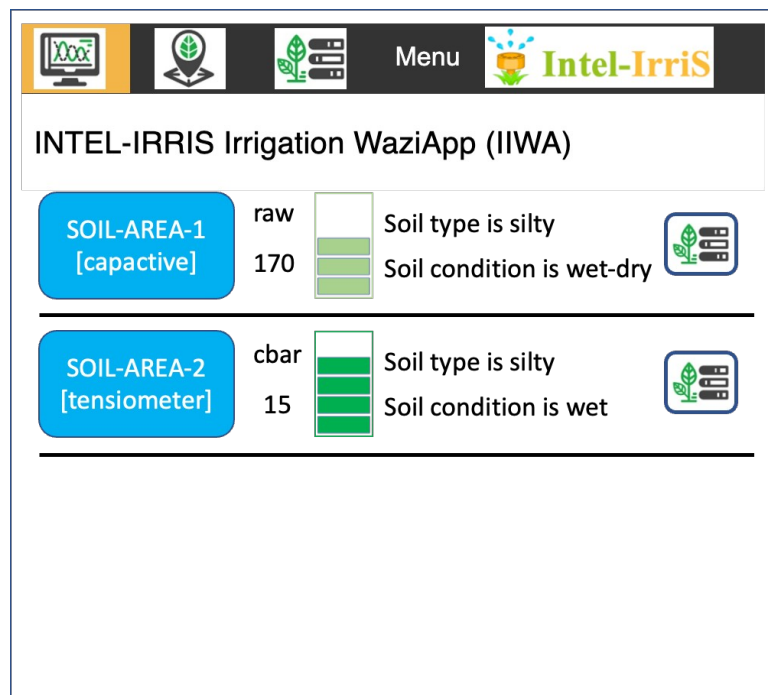


Figure 0 – Exemple d'attendue potentielle

# État Initial de l'application

La première étape de notre projet a été de nous renseigner sur l'état initial de l'application, ainsi que sur sa documentation pour essayer de comprendre son fonctionnement. L'une des informations les plus importantes que nous avons appris est que l'interface utilisateur doit être faite pour le format mobile. Initialement pour la partie interface utilisateur cette application était divisée en 3 pages HTML.

## Le dashboard

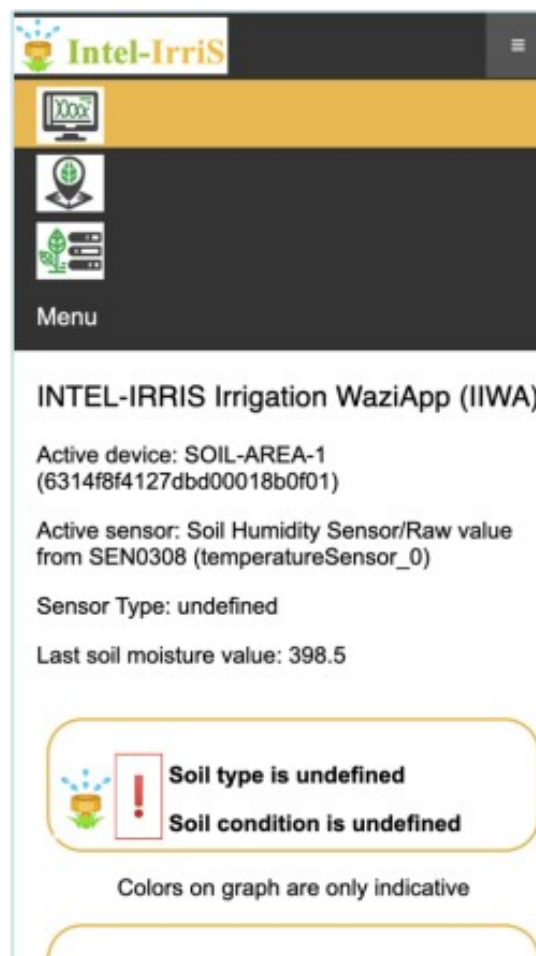


Figure 1.1.1 – Dashboard

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 pour très sec, l'orange pour sec, le vert pour humide et le bleu clair pour très humide.

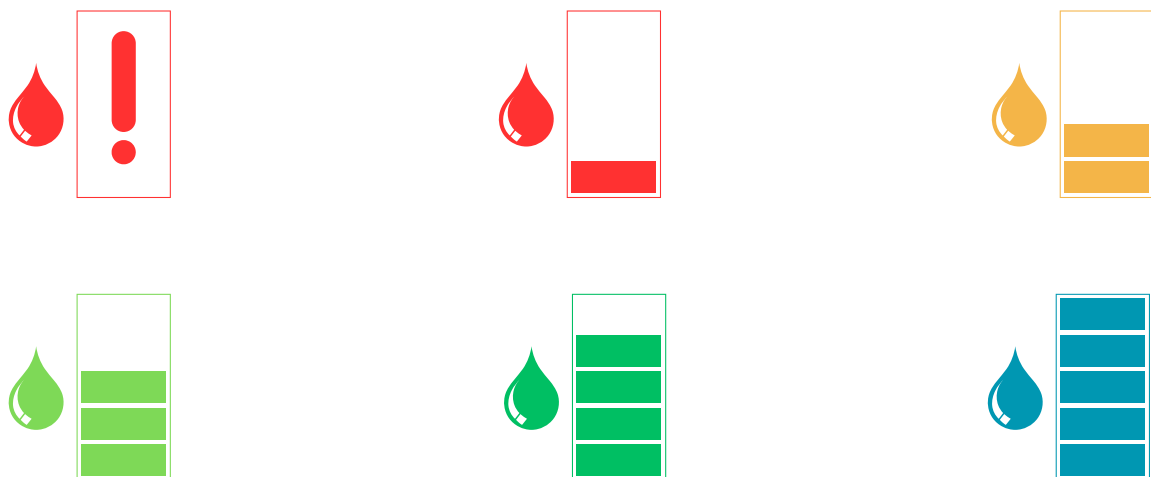


Figure 1.1.2 – Différents niveaux possibles

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

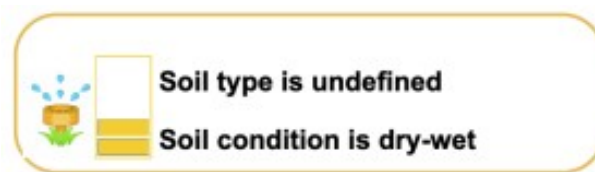


Figure 1.1.3 – Exemple d'un état du sol

Dans cette page nous avons remarqué plusieurs éléments qui pouvaient être retravaillés. Par exemple, le fait qu'il n'y ait qu'un seul appareil d'affiché et qu'il faille aller le sélectionner dans une page différente, 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. Par 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 2ème fenêtre le « device manager », un endroit depuis lequel l'utilisateur pouvait :

- Voir la liste de tous les appareils, leur nom ainsi que les type de capteurs associés
- Sélectionner un appareil « actif », qui serait donc affiché dans le dashboard et qui pourra être configuré.
- Ajouter un nouvel appareil en précisant le type de capteurs associés.
- Supprimer un appareil.

Figure 1.2.1 – Device manager initial.

Initialement nous souhaitions modifier plusieurs points 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 fenêtres possibles n'était pas affiché.

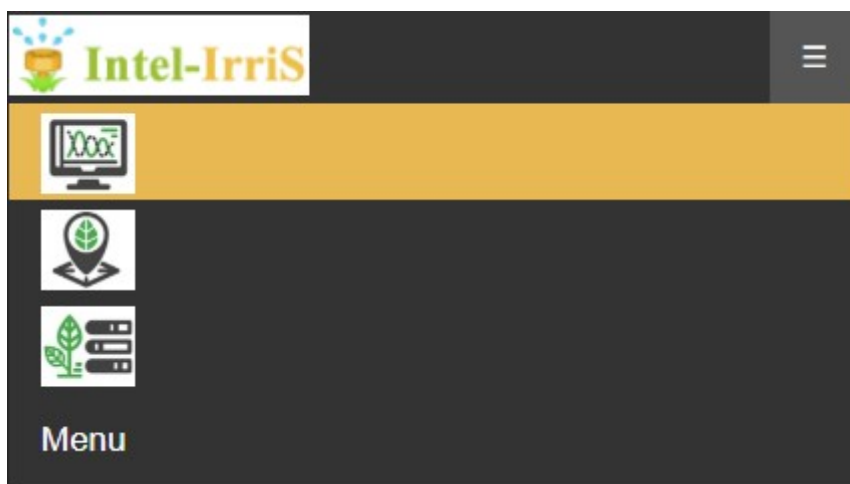


Figure 1.2.2 – Affichage initial pour sélectionner un appareil actif.

Nous avons aussi pensé à ajouter des filtres pour trier la liste des appareils par type, nom ou autre critère pertinent. Permettre ainsi à l'utilisateur de personnaliser les informations affichées pour chaque appareil dans la liste.

Il y avait aussi le fait que pour supprimer ou sélectionner un appareil actif cela se faisait par son ID et non par son nom. Cette méthode obligeait l'utilisateur à vérifier la liste de tous 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.

*Remove a device from IIWA*

643bb2e51b4f9137cc406b73

643bb2e71b4f9137cc406b77

643bb3061b4f9137cc406b7e

643bb3111b4f9137cc406b82

Remove : 

643bb2e51b4f9137cc406b73

▲

-

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 tous les appareils qui montraient l'id qui n'est pas réellement une informations utile que nous aurions pu remplacer par d'autres informations comme le nom de l'appareil.

Cependant au fur et à mesure de la réflexion nous nous somme rendu compte que cette page pouvait être supprimée 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 seule et unique fenêtre et de ne pas avoir à naviguer plusieurs fois entre 2 pages différentes simplement pour avoir des informations sur chacun de ses appareils. Ce choix permettrait aussi de rendre l'application plus facile à utiliser pour les utilisateurs novices, de réduire le temps nécessaire pour accomplir des tâches et ne pas avoir à préciser un device actif.

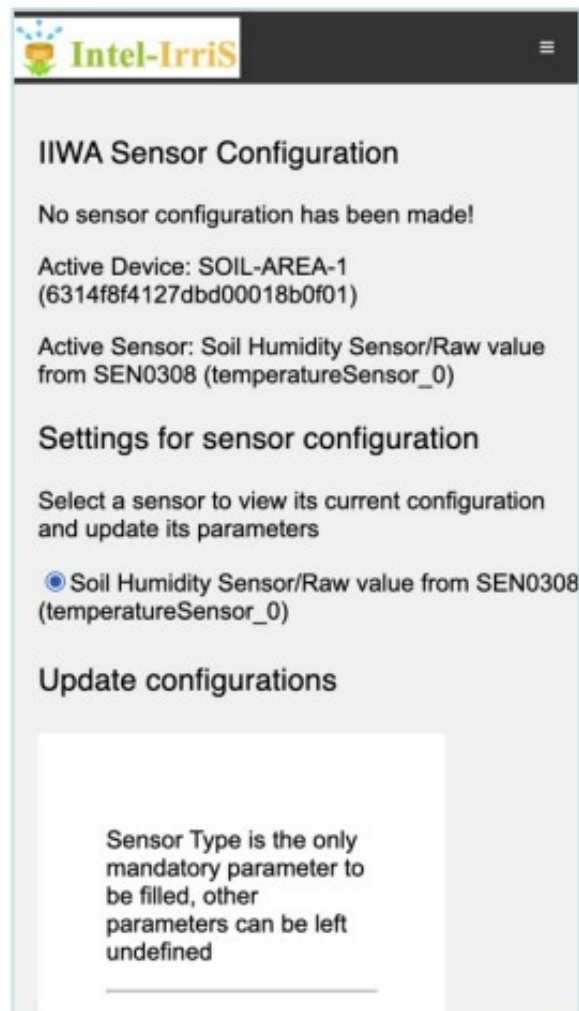
Par ailleurs, en effectuant cette suppression, ça pourrait également améliorer la sécurité de l'application, en réduisant les points d'entrée potentiels pour les attaques de piratage. Enfin, cette modification pourrait également faciliter le travail de maintenance et de mise à jour de l'application, en réduisant le nombre de pages et de codes à gérer. Cela pourrait également permettre une meilleure évolutivité de l'application en permettant, plus facilement, l'ajout de nouvelles fonctionnalités.



# 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' with a title 'IIWA Sensor Configuration'. It displays the following information:

- No sensor configuration has been made!**
- Active Device:** SOIL-AREA-1 (6314f8f4127dbd00018b0f01)
- Active Sensor:** Soil Humidity Sensor/Raw value from SEN0308 (temperatureSensor\_0)
- Settings for sensor configuration**
  - Select a sensor to view its current configuration and update its parameters
  - ☒ Soil Humidity Sensor/Raw value from SEN0308 (temperatureSensor\_0)
- Update configurations**
  - Sensor Type is the only mandatory parameter to be filled, other parameters can be left undefined
  - A text input field is visible below the instruction.

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 quelques paramètres.

Au niveau de cette page nous n'avons pas trouvé d'éléments à revoir. La page nous paraissait assez complète nous n'y avons donc que très peu touché.

# Avancée prévisionnelle et méthode de travail.

Pour prévoir nos avancements par rapport aux objectifs que nous nous sommes fixés nous avons décidé d'établir un diagramme de Gantt. Ce diagramme nous a permis d'avoir une certaine ligne à suivre pour l'avancement du projet et de ses fonctionnalités pour que la finalité corresponde aux attentes du client.

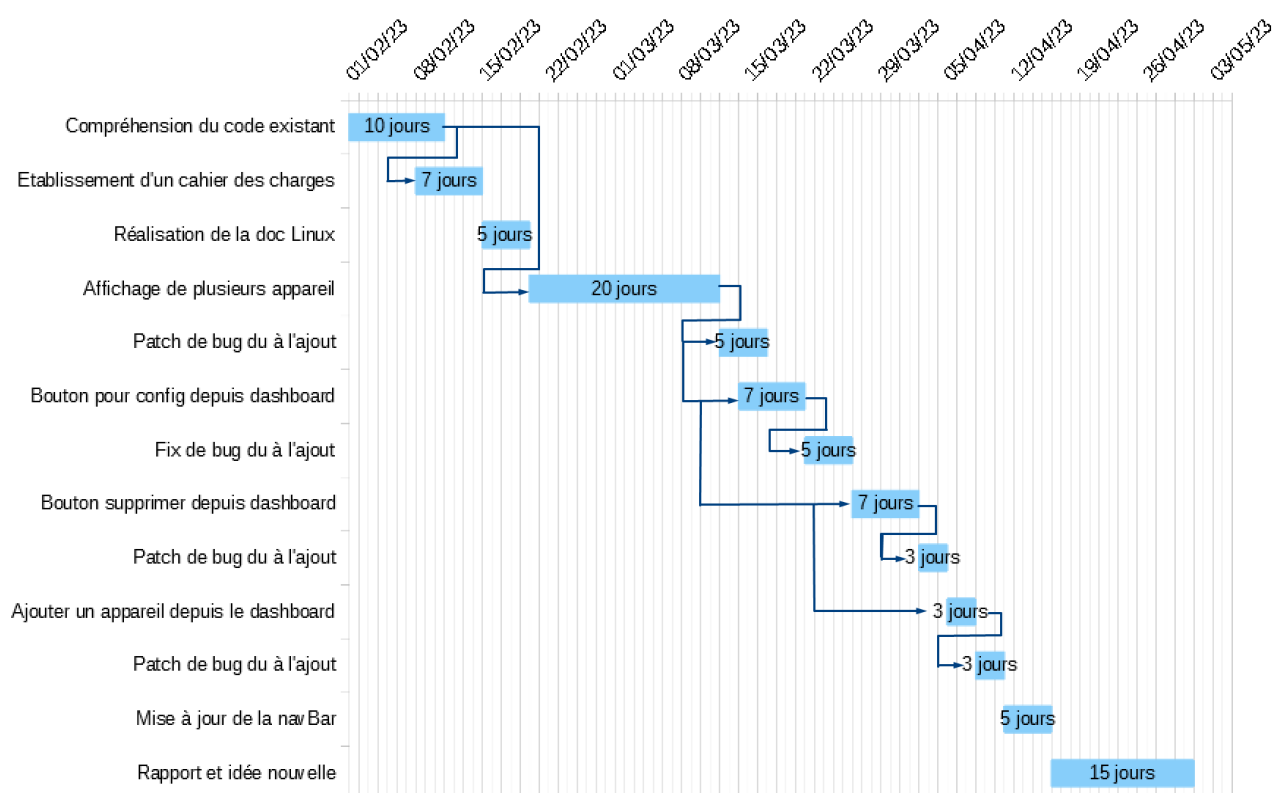


Figure 2.1.1 – Diagramme de Gantt initial

Comme visible, la plupart des tâches que nous avons prévues pour le projet nécessitaient que l'affichage de plusieurs appareils soit effectué. Cela impliquait donc que nous devions passer du temps à travailler sur la conception et le développement de cette fonctionnalité clé. De plus, chaque rajout que nous faisons était suivi d'une période de test et de validation pour nous assurer que tout fonctionnait correctement et qu'il n'y avait pas de bugs.

Dans l'ensemble, le diagramme de Gantt nous a aidés à organiser efficacement notre temps et à nous assurer que nous avançons sur le projet de manière régulière et structurée.

Au niveau de notre méthode de travail nous avons décidé d'utiliser une méthode qui se rapproche d'une dite en cascade. Cette méthode contient plusieurs points qui sont visibles sur le graphique ci-dessous.

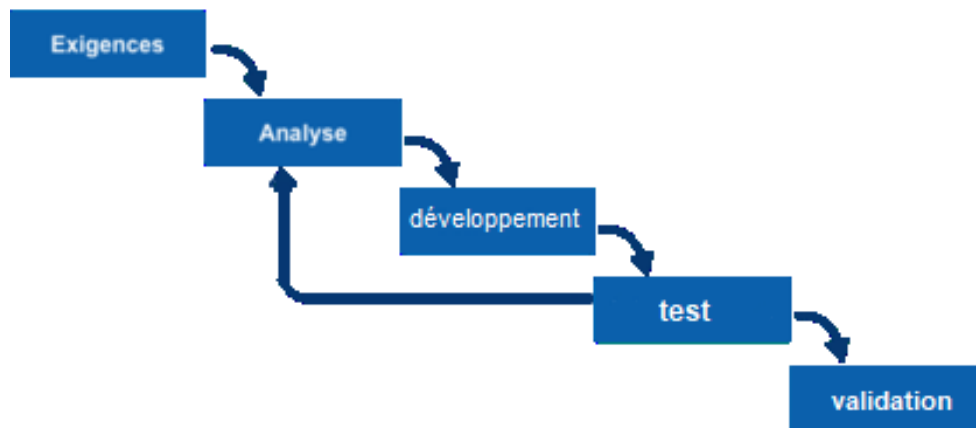


Figure 2.1.2 – Méthode de travail

Comme visible en premier point il y a les exigences qui vont représenter les besoins et les attentes du client ainsi que les fonctionnalités attendues du logiciel. Pour cette partie nous nous sommes basés sur un cahier des charges que nous avons fixé en plus d'une capture d'écran fournie par notre encadrant (voir figure 0).

Pour continuer nous avons la partie analyse qui va représenter les méthodes et outils qui seront utilisés pour réaliser les dites exigences. Le programme utilisant déjà certain framework nous avons décidé de garder ce utiliser pour éviter d'alourdir l'application et donc les outils était déjà trouver.

En troisième point nous avons la phase de développement où nous allons développer le code pour répondre aux exigences. Cette phase sera suivie par les tests, qui permettront de s'assurer que chaque composant du logiciel fonctionne correctement.

Si les test sont concluants nous remontons à l'étape d'analyse pour trouver une méthode pour réaliser ce que nous avons dans le cahier des charges, le développer et le tester ainsi de suite jusqu'à que le cahier des charges soit terminé.

Enfin, la phase de validation consiste à vérifier que le logiciel est prêt pour une utilisation réel. Cette étape permet de s'assurer que le logiciel est bien conforme aux exigences initiales et que les tests ont été effectués avec succès.

# Travail réalisé

La première étape que nous nous sommes donnée était de permettre l’affichage de plusieurs appareils et la réalisation de la documentation pour faire fonctionner le programme sur Linux. Par la suite nous avons permis à l’utilisateur de configurer chaque appareil rajouté sans avoir à en définir 1 en temps qu’actif. Pour terminer nous avons corrigé des bugs créés et optimiser le code et avons permis à l’utilisateur d’ajouter ou supprimer un appareil directement depuis le dashboard et donc supprimé le « device manager ».

## Réalisation de la documentation

Lors de la mise en place des différentes dépendances du projet, nous nous sommes rendu compte qu’une documentation, pour les systèmes d’exploitations basés sur Linux, était absente.

(la documentation macOS était utilisée pour l’installation sur les systèmes basés sur 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 fait le choix d’expliquer comment faire en utilisant la distribution Debian car c’est une distribution qui est notamment utilisée comme base d’autres distributions 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 intégrée 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 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)
```

Figure 3.1.1 – Installation de Go et MongoDB

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.
```

Figure 3.1.2 – Lancement du serveur

# Affichage de plusieurs appareils

Pour effectuer cette partie nous avons créé une liste (nommer tuple en python) `all_devices_tuple` qui contient 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 de 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 3.2.1 – Code qui liste tous les devices

Le développement de l'application web nécessitait une bonne compréhension du framework Flask utilisé pour la gestion de la partie Web de l'application. Flask étant un framework web en Python qui permet de créer des applications web rapidement et facilement en utilisant une architecture légère et extensible.

Dans un premier temps, nous avons donc étudié la documentation de Flask afin de bien comprendre son fonctionnement et sa structure. Nous avons également examiné les différents modules et packages disponibles pour Flask qui pourraient nous aider à construire notre 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 intégrer des données dynamiques en utilisant des fichiers qui vont être dits de modèles (template).

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

```
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 3.2.2 – Envoie le template à afficher à l'HTML

Dans le template HTML de la page dashboard, nous avons utilisé une boucle for de Jinja2 pour parcourir la liste des appareils et afficher les informations de chacun d'entre eux. Les balises Jinja2 nous ont également permis d'afficher les informations des capteurs de manière claire et lisible pour l'utilisateur.

L'utilisation de Jinja2 nous a permis de rendre notre code plus modulaire et facilement maintenable. En effet, nous avons pu séparer la logique de présentation de notre application de la logique de traitement des données. Cela nous a également permis de réduire la quantité de code HTML nécessaire pour afficher les données, ce qui a amélioré l'efficacité de notre application web.

```
{% 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>
          <a onclick="deleteDevice('{{ device[0] }}')">
      </tr>
    </thead>
  </table>
{% endfor %}
```

Figure 3.2.3 – Partie affichage de l'HTML



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

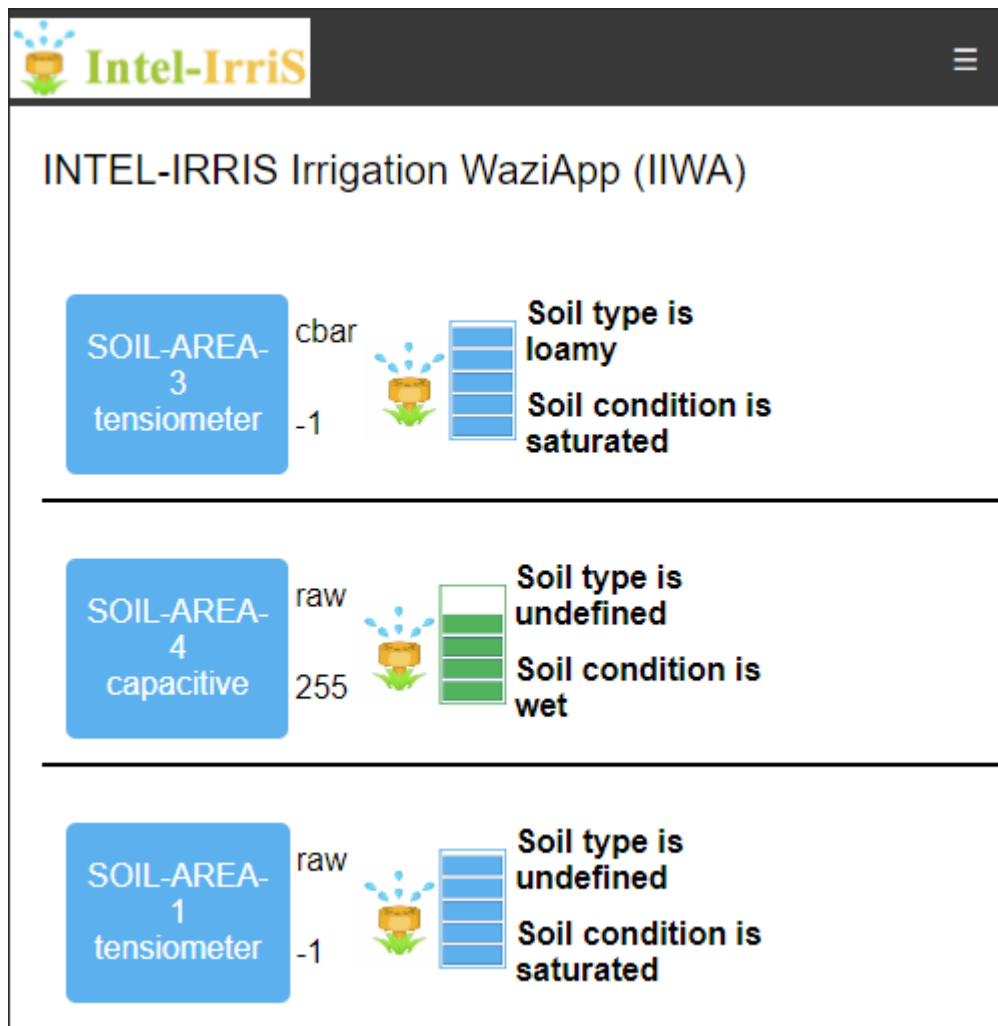


Figure 3.2.4 – Affichage de plusieurs appareils

Comme nous pouvons le voir tous les appareils sont visibles en fonction de leur ordre d'ajout, et avec, pour chacun, tous leurs paramètres inclus.

Cependant nos rajouts ont créé un bug que nous avons dû patch qui est le suivant : Lors de l'ajout d'un appareil si aucun appareil n'était sélectionné initialement comme appareil dit « actif » aucun appareil n'était affiché.

Pour corriger cela nous avons rajouté le fait que si il y avait au moins un capteur mais qu'aucun n'était défini comme actif alors le premier de la liste était sélectionné . Cette méthode permet aussi d'avoir un appareil 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 3.2.5 – Définit un capteur par défaut

Comme visible sur la capture d'écran ci-dessus la méthode utilisée est assez simple, nous récupérons l'id du premier appareil ainsi que l'id de son premier capteur. Juste après nous les attribuons en tant qu'appareil « actif » et l'enregistrons dans un fichier json.

## 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 était 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( reponse => {
    if(reponse.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 3.3.1 – fonction POST pour configurer un appareil

Bien entendu, une gestion des erreurs a aussi été mise en place, permettant par exemple d'éviter que la page charge avant qu'elle n'ait eu une réponse du serveur ayant pu causer l'affichage d'un appareil différent de celui que l'on souhaite configurer.

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 3.3.2 – 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 configurator.



Figure 3.3.3 – Bouton configurateur

Pour supprimer un appareil 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 présent initialement dans le programme.

```

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 3.3.4 – 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 3.3.5 – Une des parties du POST dans le python.

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



Figure 3.3.6 – Bouton pour supprimer

Après discussions avec notre encadrant nous nous sommes rendu compte que nous avions oublié d'ajouter une « alert » permettant de valider ou non la suppression d'un appareil. Nous l'avons donc rajoutée ce qui nous affiche ceci avant de supprimer un appareil.

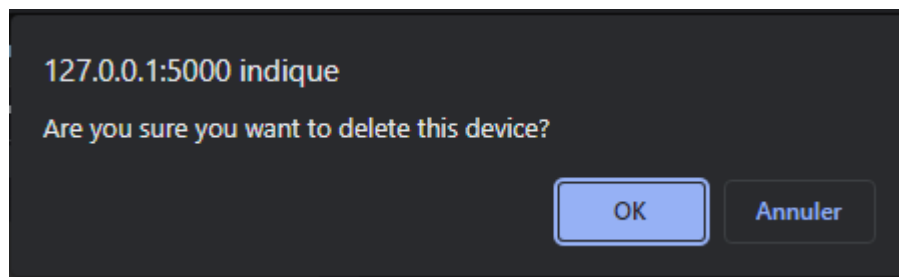


Figure 3.3.7 – Message de confirmation

Nous avons hésité à rajouter une alerte plus « esthétique » pour rendre l'application plus accueillante. Cependant l'application étant destinée à être assez légère et fonctionnelle sur différents types d'appareil mobile nous avons décidé 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 pages possibles et ne changeait pas la couleur en fonction de celles-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 possibles.

```
<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 3.4.1 – Code dans l'HTML

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

Figure 3.4.2 – 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 programmer que le texte ne s'affiche que si l'utilisateur est sur mobile.

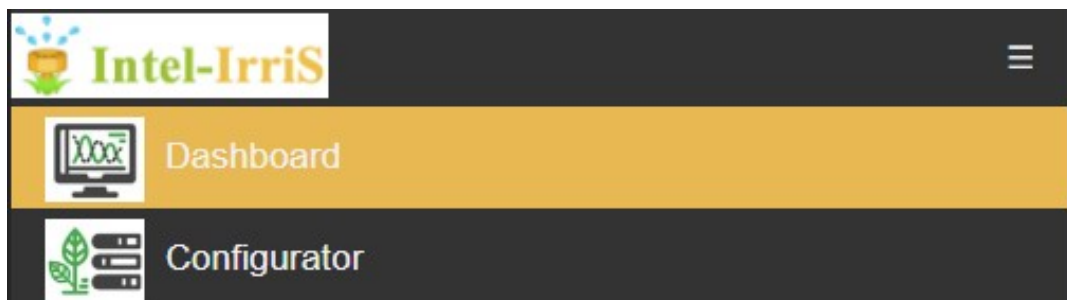


Figure 3.4.3 – Affichage final de navbar

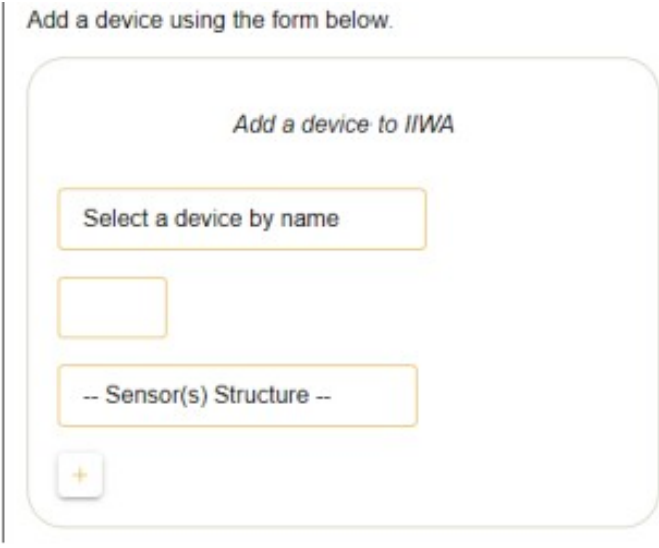
Comme vous avez pu le remarquer sur la capture d'écran juste au-dessus, nous avons décidé de ne plus afficher le device manager n'ayant plus aucune utilité pour l'utilisateur.

Cependant pour des questions de développement et potentiellement de debugage nous avons décidé de laisser le code dans le programme initial et donc, si l'utilisateur rentre le bon URL, il pourra avoir accès au device manager et à ses fonctionnalités.

## Ajout d'appareil depuis le dashboard et Affichage Final

Pour l'ajout d'un appareil depuis le dashboard, nous avons essayé de reprendre le code, que nous avions dans le device manager, en remplaçant la redirection qui était initialement vers le device manager désormais vers le dashboard. Lors de nos premiers tests nous n'avions pas remarqué de problème particulier. Cependant après des tests un peu plus avancés il a été trouvé que lorsque l'utilisateur appuyait sur F5 ou rechargeait la page générait un appareil en trop.

Ce problème est dû au fait que lorsque l'utilisateur appuie sur F5 ou actualise la page le programme réalise de nouveau un submit du POST. Pour régler ce souci, au lieu de faire un `render_template`, qui permet de régénérer la page, nous avons utilisé `redirect` qui permet de faire une redirection vers une page. Grâce à cela, nous avons redirigé l'utilisateur vers la page au lieu de juste la régénérer visuellement ce qui permet de faire « disparaître » le post du navigateur et permet donc d'éviter le bug.



The screenshot shows a web form titled "Add a device using the form below." inside a light gray box. Inside this box, there is a sub-header "Add a device to IIWA". Below the header, there is a text input field with the placeholder "Select a device by name". Underneath this is a smaller, empty text input field. Below that is a dropdown menu with the text "-- Sensor(s) Structure --". At the bottom left of the form area is a small square button with a plus sign (+).

Figure 3.5.1 – Ajout d'un appareil depuis le dashboard



Le rajout de cette fonctionnalité nous donne cet affichage qui sera l’affichage final. Ayant supprimé le device manager et n’ayant quasiment rien modifié dans le configurateur cette page est la partie qui a le plus changé et qui possède un affichage.

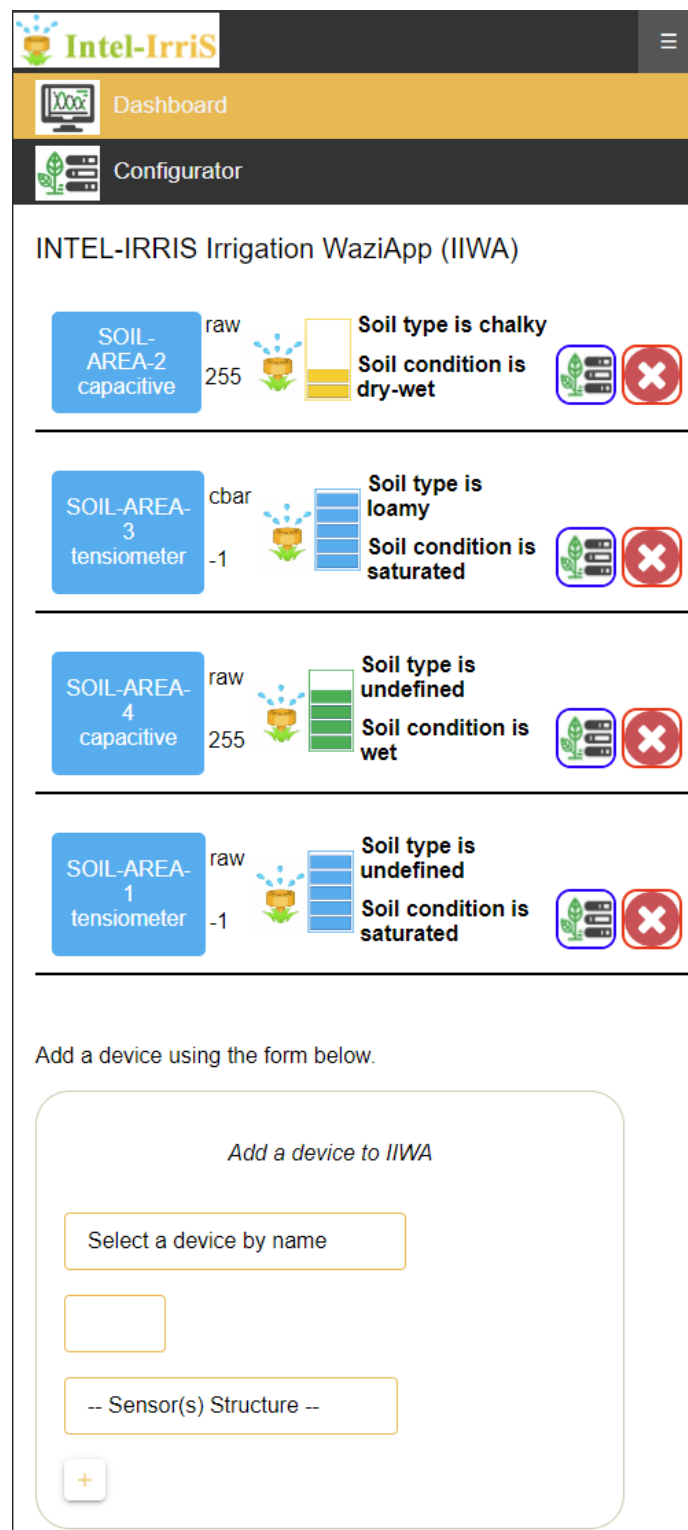


Figure 3.5.2 – Affichage final du dashboard

Si nous comparons cet affichage avec la maquette initiale nous pouvons assez facilement nous rendre compte qu’elle correspond aux attentes.

# Avancée réelle et apprentissage personnel

Dans le cas de notre projet, nous avons constaté que l'avancée réelle ne correspondait pas à l'avancée prévue. En effet, certains imprévus, comme le fait que nous ayons eu un autres projet assez important en temps à rendre pour début avril ou la complexité de certaines tâches ont retardé notre projet.

Cependant d'autres aspects ont été plus faciles et rapides à réaliser que prévu. Tout cela a eu pour effet de décaler légèrement notre calendrier. Nous avons tout de même su nous adapter et ajuster notre planning au fur et à mesure de l'avancée du projet pour maintenir notre objectif de livrer le logiciel avec toutes les fonctionnalités prévues avant début mai.

Voici un diagramme de Gantt permettant de montrer notre avancée réelle :

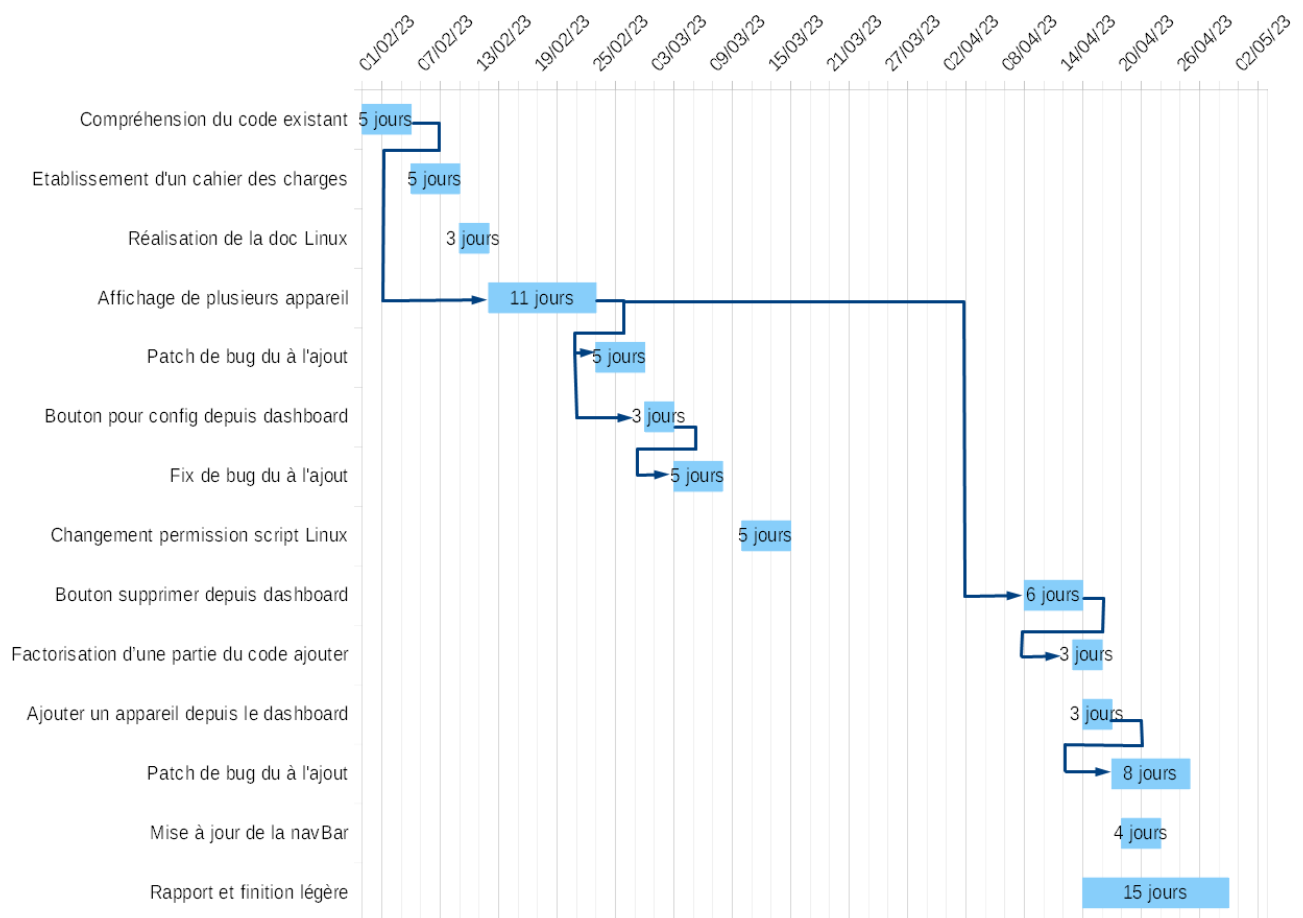


Figure 4.1.1 – Diagramme de Gantt de l'avancée réelle

Comme visible sur ce diagramme de Gantt la majorité de nos tâches ont pris moins de temps que prévu initialement et certaines tâches ont été remplacées par d'autres. Comme, par exemple, le fait de patch de potentiel bug dû à l'ajout de la suppression des appareils directement depuis le dashboard qui a été supprimé car aucun bug n'était visible et a été remplacé par un changement concernant les permissions que nécessitaient certains scripts pour être parfaitement fonctionnels sur Linux , changement qui n'était pas prévu initialement lorsque nous avons commencé le projet.

De mi-mars à début avril ayant été pris par un projet assez long ainsi que plusieurs examens nous avons décidé de mettre en pause le projet d'amélioration de l'interface utilisateur pour pouvoir nous concentrer sur nos autres obligations. Cela a retardé légèrement notre avancée cependant ayant un peu trop sur-estimé la charge de travail à la base nous avons repris le projet avec un planning modifié pour nous permettre de le finir dans les délais impartis.

Au cours de ce projet, nous avons acquis plusieurs compétences en matière de gestion de projet. En premier lieu, nous avons compris l'importance de la planification et de la définition des tâches à accomplir via un diagramme de Gantt avant de commencer le projet. Cette étape nous a permis de mieux comprendre l'ampleur du projet voire même de le sur-estimé un peu et de réduire les risques d'oublier des tâches qui pouvaient être dites critiques. Ce projet nous a également appris à identifier les priorités et à hiérarchiser les tâches en fonction de leur importance.

Cette étape a été très utile pour apprendre à mieux gérer notre temps et à nous concentrer sur les tâches les plus essentielles dès le début, pour nous concentrer sur l'amélioration de points que l'on considérait prioritaire. La communication avec notre encadrant a été un point essentiel. Lors des différentes réunions, où nous lui avons montré l'état de notre avancement et nos prévisions, il a su nous aider dans certains choix concernant l'interface.

L'importance de la flexibilité et de l'adaptation en cours de projet a été une variable importante. Lorsque des problèmes imprévus sont survenus, nous avons su réajuster la planification initiale et trouver des solutions alternatives pour terminer ce qui était demandé.

# Conclusion

En conclusion, notre projet de développement de la partie interface utilisateur de l'application IIWA a été une expérience enrichissante. Nous avons permis à l'utilisateur de désormais pouvoir gérer l'ensemble de ses appareils depuis une seule et unique fenêtre et de pouvoir les configurer très facilement. Quelques points qui pouvaient être imprécis ont été changés pour rendre l'application plus intuitive et simple d'utilisation pour les nouveaux utilisateurs.

Grâce à ce projet nous avons aussi acquis une expérience pratique dans la gestion de projet de développement logiciel, y compris la planification et l'organisation de tâches, la communication avec notre encadrant, et l'adaptation à des changements durant la période de développement. Nous avons également pu approfondir nos connaissances en matière de développement web, en utilisant des technologies telles que Flask et Jinja2, et en résolvant des problèmes de développement de logiciels, tels que la gestion des bugs et des erreurs. Nous sommes fiers du produit final, après discussion avec notre encadrant le résultat final lui convient, nous espérons que nos améliorations seront utiles pour les personnes qui vont utiliser l'application IIWA.

Malgré nos multiples ajouts et changements il reste toujours des possibilités d'améliorations de l'application comme potentiellement pouvoir trier l'ordre dans lequel s'affiche les appareils. Certaines autres tâches qui nécessitent d'avoir de vrais appareils et qui pourraient être faites sont de localiser la position de chaque appareil par rapport à la position de l'utilisateur ou via une latitude et longitude fixe ce qui pourrait être utile si l'utilisateur souhaite les retrouver après une intempérie par exemple. Une autre chose qui pourrait être faite serait de pouvoir inspecter le temps de réponse de chaque appareil pour savoir si il est potentiellement endommagé ou positionner dans une zone contenant des perturbations réseau, cette fonction permettrait aussi d'assurer à l'utilisateur que les données qui lui sont affichées sont belles et bien en temps réel.

Nous souhaitons remercier notre encadrant monsieur Congduc Pham qui nous a énormément aidés en nous fournissant plusieurs documents sur le programme pré-existant ainsi que son avis sur l'évolution du projet au fur et à mesure du temps.

## Bibliographie

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

2023, « Jinja2 Documentation » Jinja2,, <https://jinja.palletsprojects.com/en/3.1.x/>

2023, «Flask Documentation » Flask,, <https://flask.palletsprojects.com/en/2.3.x/>

WAZIUP, Octobre 2021, «Intelligent Irrigation System for Low-cost Autonomous

Water Control in Small-scale Agriculture» UPPA, version 1.0, [Lien vers la documentation](#)

Prof. Congduc Pham, Décembre 2022, « Le starter-kit du projet INTEL-IRRIS pour les petites exploitations agricoles », <https://intel-irris.eu/wp-content/uploads/2022/12/Oran-C-Pham-starter-kit-fr.pdf>

Intel Irris, 2023, <https://intel-irris.eu/>