



Délivrable 3

Farmbot : un potager automatisé



Yoann d'Erneville - Thomas Legris - Théo Robin

Team Farmbot

Sommaire

Table des matières

Sommaire	1
I. Objectifs du projet	2
1. Rappel du cahier des charges	2
2. Révisions du planning	3
II. Avancement du projet	3
1. Installation de l'application web	3
2. Installation du Farmbot	5
3. Objectif n°1	5
1) Modification OS	5
2) Modification de l'application Web	9
a. Présentation	9
b. Avancé des modifications	12
4. Objectif n°2	17
Pour le capteur d'humidité nous avons un autre code différent du premier. Nous avons tout d'abord connecté une LED sur le port 3 de l'Arduino. Nous avons le code suivant :	20
Comment programmer les actions du Farmbot en fonction des valeurs ?	21
5. Objectifs restants	22
6. Evolution du site Web	24
III. Problèmes et solutions	25
1. Evolution du site Web	25
2. Problème d'installation	25
3. Modification du code de l'application web	25
4. Gestion de l'énergie via la modification de l'OS	25
V- Les solutions mises en place et à mettre en place	26
5. Problème d'installation	26
6. Modification du code de l'application web	26
7. Gestion de l'énergie via la modification de l'OS	26
Conclusion	27
Annexes	29
Références	30

I. Objectifs du projet

1. Rappel du cahier des charges

Dans cette partie, nous avons réalisé un tableau récapitulant notre cahier des charges. Dans la partie "Avancement du projet", nous préciserons notre avancé par rapport à ces objectifs.

Niveau d'importance	Description des objectifs
Objectif 1 (haute importance)	<ul style="list-style-type: none">- Rendre autonome le Farmbot grâce aux énergies renouvelables. Utilisation de l'énergie solaire pour l'alimentation électrique et la récupération d'eau de pluie pour l'arrosage.- Modification et amélioration du logiciel avec l'ajout de conseils et suggestions en fonction de la saison
Objectif 2 (moyenne importance)	<ul style="list-style-type: none">- Optimisation de la gestion de l'eau en installant une station météo. Modification du logiciel pour indiquer à l'utilisateur les différentes prévisions (soleil et précipitations).
Objectif 3 (basse importance)	<ul style="list-style-type: none">- Gestion efficace des eaux pluviales, par exemple : éviter que l'eau stagne.- Création d'un outil permettant de retourner la terre dans le bac.

Figure 1 - Tableau récapitulatif des objectifs

2. Révisions du planning

Au début du second semestre, nous avons réalisé un nouveau planning avec l'outil Gantt. Nous verrons dans la suite que celui-ci n'est plus tout à fait à jour avec l'avancé du projet. Nous avons eu plusieurs retards qui n'ont pas permis d'avancer sur tous les points de ce projet. Vous trouverez donc en annexe le nouveau Gantt datant du 23 avril 2018.

II. Avancement du projet

1. Installation de l'application web

Un de nos objectifs principaux est la modification de l'application web pour y ajouter un onglet avec notamment des conseils de plantage ainsi que d'autres éléments. Pour cela, nous avons à installer l'application web en local afin de modifier et de compiler sur nos propres machines. Nous avons alors suivi le guide d'installation fourni sur GitHub que nous pouvons voir sur la figure ci-dessous :

Q: How do I Setup an instance locally?

Prerequisites

You will need the following:

1. A Linux or Mac based machine. We do not support windows at this time.
2. [Docker 17.06.0-ce or greater](#)
3. [Ruby 2..5.0](#)
4. [ImageMagick](#) (`brew install imagemagick` (Mac) or `sudo apt-get install imagemagick` (Ubuntu))
5. [Node JS >= v8.9.0](#)
6. `libpq-dev` and `postgresql` and `postgresql-contrib`
7. `gem install bundler`

Setup

NOTE: A step-by-step setup guide for Ubuntu 17 users can be found [here](#)

Setup for non-Ubuntu users after installing the dependencies listed above:

1. `git clone https://github.com/FarmBot/Farmbot-Web-App --branch=master --depth=10`
2. `cd Farmbot-Web-App`
3. `bundle install`
4. `yarn install`
5. Database config: Copy `config/database.example.yml` to `config/database.yml` via `cp config/database.example.yml config/database.yml`
6. App config: **MOST IMPORTANT STEP.** Copy `config/application.example.yml` to `config/application.yml` via `mv config/application.example.yml config/application.yml`. **Please read the instructions inside the file. Replace the example values provided with real world values.**
7. Give permission to create a database*

Figure 2 : Guide d'installation de l'application

L'une des étapes la plus importante est la modification d'un fichier nommé *application.yml* où sont décrites les configurations de l'application web à savoir notamment l'adresse IP de l'API et du serveur MQTT, le port utilisé ou bien le type d'adresse e-mail. Notre fichier *application.yml* ressemble à ceci :

```
# STOP
# READ THIS BEFORE USING IT.
# SEE NOTES BELOW:
#
# You will hit issues if any of these are set to the wrong value.
# Please read each line of this file before starting the server.
#
# PLEASE READ ALL ENTRIES.
# =====
# Self hosting users can safely delete this (a new key will be created).
# This key is used to exchange secrets between bots and MQTT servers (important
# if you don't use SSL)
# SERVER WON'T WORK IF YOU FORGET TO DELETE THIS EXAMPLE TEXT BELOW.
# ADD A REAL RSA_KEY OR DELETE THIS LINE!!
RSA_KEY:
# If you use Let's Encrypt for SSL,
# you must set this when renewing SSL.
# Otherwise, not required and CAN BE REMOVED.
ACME_SECRET:
# If your server is on a domain (eg: my-own-farmbot.com), put it here.
# DONT USE 'localhost'.
# DONT USE '127.0.0.1'.
# DONT USE '0.0.0.0'.
# Use a real ip or domain name.
API_HOST: "148.60.142.217"
# 3000 for local development. 443 is using SSL. You will need 'sudo' to use PORT
# 80 on most systems.
API_PORT: "3000"
# This can be set to anything.
# Most users can just delete it.
# This is used for people writing modifications to the software, mostly.
DOCS:
# Most users can delete this.
# Used by people who pay for managed database hosting.
DATABASE_URL:
# MUST REPLACE.
# Generate a secret by typing 'rake secret' into the console.
DEVISE_SECRET:
a9dff15fcbdf2584c93ec6b5cf9bf805674814598986a577eb778f63e7aa50ef39df3394254ac3b2250d1
# Most personal server users can delete this.
FORCE_SSL:
# FarmBot OS update server. Use default if you don't have a special use case.
# Off grid servers may have issues connecting to our update URL.
OS_UPDATE_SERVER: "https://api.github.com/repos/farmbot/farmbot-os/releases/latest"

# Deleting this will save to disk.
# Most self hosting users will want to delete this.
GCS_BUCKET:
# Google Cloud Storage ID for image data.
# Deleting this will save images to disk.
# Most self hosting users will want to delete this.
GCS_ID:
# Most self hosting users will want to delete this.
GCS_KEY:
# Most self hosting users will want to delete this.
HEROKU_SLUG_COMMIT:
# Where is your MQTT server running?
# Use a REAL IP ADDRESS if you are controlling real bots.
# 0.0.0.0 is only OK for software testing. Change this!
MQTT_HOST: "148.60.142.217"
# Delete this line if you are not an employee of FarmBot, Inc.
NPM_ADDON:
# Same as above. Can be deleted unless you are a Rollbar.IO customer.
ROLLBAR_ACCESS_TOKEN:
ROLLBAR_CLIENT_TOKEN:
# FarmBot, Inc. uses SendGrid to send emails.
# Delete these if you don't use send grid.
SENDGRID_PASSWORD:
SENDGRID_USERNAME:
# For email delivery. Who is your email host?
SMTP_HOST: "smtp.gmail.com"
# Optional with default of 587
SMTP_PORT: "587"
# Set this if you don't want to deal with email verification of new users.
# (self hosted users)
NO_EMAILS: "TRUE"
# Self hosting users can delete this line.
# If you are not using the standard MQTT broker (eg: you use a 3rd party
# MQTT vendor), you will need to change this line.
MQTT_WS:
# ENV var used by FarmBot employees when building different versions of the JWT
# auth backend plugin.
# Can be deleted safely.
API_PUBLIC_KEY_PATH:
# If you are using a shared RabbitMQ server and need to use a VHost other than
# "/", change this ENV var.
MQTT_VHOST: "/"
# If you run a server with multiple domain names (HINT: You probably don't),
# you can list the names here. This is used by FarmBot employees so that they
# can securely host the same server on multiple domain names
# ex: mv.farm.bot, mv.farmbot.io
```

Figure 3 : fichier de configuration *application.yml*

Après avoir réalisé ces étapes, nous avons pu lancer l'API puis le serveur MQTT. Une fois cela effectué, nous nous sommes connectés au port 3000 en local. Nous arrivons donc sur la page de connexion de l'application web du Farmbot.

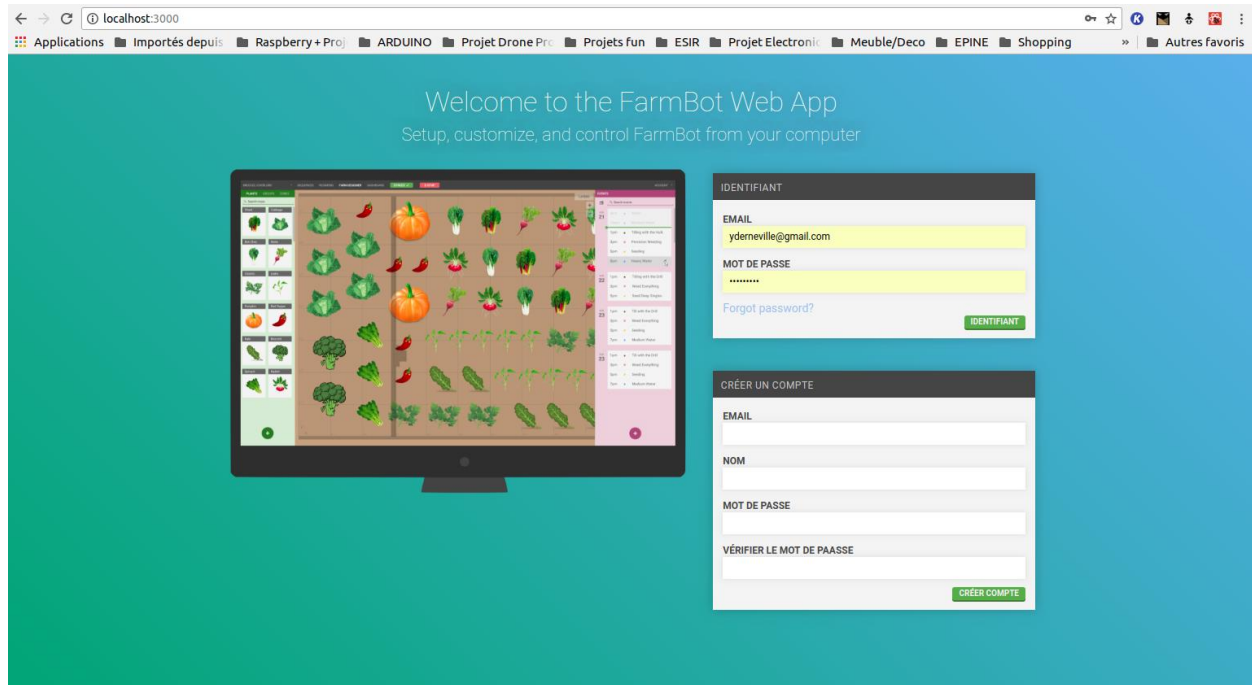


Figure 4 : Aperçu de la page de connexion

2. Installation du Farmbot

Le Farmbot doit être installé dans le jardin de l'université de Rennes 1 mais pour le moment nous n'avons malheureusement pas pu le faire en raison des contraintes météorologiques. Nous avons déjà assemblé une grosse partie du robot : le bras, la partie électronique avec l'Arduino et la Raspberry, les différentes têtes et les rails. Nous espérons pouvoir le monter très prochainement afin de voir l'aspect matériel de notre projet.

3. Objectif n°1

1) Modification OS

Un de nos objectifs principal est de réaliser une amélioration de l'OS du Farmbot pour que celui-ci soit plus économe en termes d'énergie. Actuellement, les différentes parties du Farmbot sont alimentées en continue et donc très énergivores. Pour cela, nous avons décidé de travailler sur la partie gérant l'alimentation des moteurs et commandant la partie physique du Farmbot. En effet, pour nous l'Arduino est une partie essentielle mais il peut être éteint lorsque le Farmbot n'a pas à réaliser de tâches. Dans un premier temps, le but était de réussir à éteindre l'Arduino à partir de la Raspberry sur laquelle est placé l'OS du Farmbot. Durant nos premières recherches, nous pensions que l'Arduino était alimenté par les pins de la Raspberry prévu à cette effet (5V). Nous avons donc réalisé une fonction en *Elixir* qui permettait de contrôler les pins concernés. Mais nous avons rencontré un

problème car il s'agit de pins de puissance et leur état ne peut pas être modifié plusieurs fois. Nous avons donc imaginé une solution où nous utilisons une des pins programmables de la Raspberry qui viendrait commander un transistor permettant de laisser passer ou non le courant. Nous n'avons pas terminé nos essais sur ce sujet car nous nous sommes rendu compte que l'Arduino était branché sur un des ports USB de la Raspberry. Il nous suffit donc de commander ce port pour permettre d'allumer et d'éteindre l'Arduino. Pour cela, nous allons utiliser la librairie libusb-dev. Une fois ajoutée, celle-ci nous permet d'utiliser la commande hub-ctrl pour changer l'état des ports USB. Nous pouvons ensuite préciser le port sur lequel nous souhaitons travailler et à quel état nous voulons le mettre.

```
defmodule PonArd do
  def power_on() do
    :os.cmd('./hub-ctrl -h 0 -P 2 -p 1')
  end
end
```

Figure 5 : Fonction allumant l'arduino

L'image ci-dessus est un exemple où dans le module PonArd on retrouve la fonction power_on() qui va venir allumer ("-p 1") le port 2 où est branché l'Arduino ("-P 2").

L'étape suivante était de réaliser une fonction plus conforme à nos attentes. C'est à dire que cette fonction viendrait éteindre l'Arduino puis la rallumer après un laps de temps passé en paramètre. Ce temps correspondra par la suite au temps qui s'écoule entre deux actions programmées par l'utilisateur via l'application, pour cela nous utilisons la fonction timer.

```
defmodule PowerArd do
  def timetowait(time) do
    :os.cmd('./hub-ctrl -h 0 -P 2 -p 0')
    :timer.sleep(time)
    :os.cmd('./hub-ctrl -h 0 -P 2 -p 0')
  end
end
```

Figure 6 : Fonction permettant le mode veille

L'étape suivante est de déterminer ce temps entre deux actions. Lorsque l'utilisateur programme un événement à réaliser, comme par exemple l'arrosage d'une de ses plantes, une base de données est remplie. Cette base de données contient des informations telles que le type d'événement à réaliser, la date de début et de fin de l'événement etc. Nous allons plus particulièrement utiliser ces deux informations.


```

create_table "farm_events", id: :serial, force: :cascade do |t|
  t.integer "device_id"
  t.datetime "start_time"
  t.datetime "end_time"
  t.integer "repeat"
  t.string "time_unit"
  t.string "executable_type", limit: 280
  t.integer "executable_id"
  t.index ["device_id"], name: "index_farm_events_on_device_id"
  t.index ["end_time"], name: "index_farm_events_on_end_time"
  t.index ["executable_type", "executable_id"], name: "index_farm_events_on_executable_type_and_executable_id"
end

```

Figure 7 : Attributs de la base farm_events

En effet, nous avons décidé de mettre en place un algorithme qui va permettre de récupérer le temps que l'on aura à attendre. Pour cela, nous récupérerons l'heure et la date actuelle de la Raspberry sous le format datetime, qui est le format utilisé dans les différentes parties du Farmbot. Une fois ce temps connu, nous parcourons les différents temps de début d'événement pour récupérer l'événement qui est le prochain à arriver c'est à dire le plus proche de la date actuelle. Une fois cette date récupérée, il suffit de faire une opération entre la date actuelle et celle du prochain événement pour pouvoir connaître le temps d'attente en seconde. C'est ce temps qui est ensuite passé à la fonction réalisée précédemment. Nous avons dans un premier temps réalisé cela en Javascript car il s'agit d'un langage de programmation avec lequel nous avons l'habitude de travailler. Une fois celui-ci fonctionnel, nous avons réalisé les différents programmes en Elixir. Nous programmons ainsi pour pouvoir ensuite intégrer au mieux notre code dans celui de l'OS. De plus, si la date du prochain événement est trop proche alors il n'est pas nécessaire d'éteindre et de rallumer la carte Arduino. Nous avons donc pour l'instant décidé de prendre un temps minimal de cinq secondes.

Comme nous l'avons vu précédemment, lorsqu'un événement est programmé par l'utilisateur, différentes informations sont stockées dans une base de données. Cette base est ensuite exploitée par l'OS pour réaliser les différentes tâches. Pour le projet Farmbot, différents outils sont consacrés à la partie base de données. Ces outils sont notamment Ecto et PostgreSQL. Ce dernier est un système de gestion de base de données relationnelle fréquemment utilisé. Ecto est un projet contenant un wrapper de base de données et un langage de requête intégré. Ecto est composé de 4 composants principaux : Ecto.Repo, Ecto.Schema, Ecto.Changeset et Ecto.Query. Le premier permet de créer un dépôt où sera stocké la base de données. Avec Ecto.Repo, nous pouvons insérer, créer, supprimer et interroger un dépôt. Ecto.Schema est employé pour mapper une source de données dans une structure Elixir. Ensuite, le composant Ecto.Changeset permet de modifier les paramètres de la base de données. Pour finir, Ecto.Query permet quant à lui de réaliser des requêtes.

Nous avons donc créé une base similaire à la base farm_events du Farmbot pour pouvoir l'utiliser dans notre programme. Nous réalisons également le schéma correspondant pour pouvoir utiliser cette base de données dans nos programmes Elixir. Nous avons décidé de simplifier la table en indiquant comme attribut uniquement un id ("ide") correspondant à l'id de l'événement, une date de début d'événement ("startdate") et une date de fin d'événement ("enddate"). Nous avons conçu également la fonction changeset pour pouvoir agir sur les attributs. Pour pouvoir créer cette base, nous avons dû définir un utilisateur postgresQL. A

celui-ci, nous devons attribuer les droits de création d'une base de données pour que cela fonctionne.

```
defmodule Farmbot.Item do
  use Ecto.Schema
  import Ecto
  import Ecto.Changeset
  import Ecto.Query

  schema "farmbdd" do
    field :ide, :integer
    field :startdate, :utc_datetime
    field :enddate, :utc_datetime
  end

  @fields ~w(ide startdate enddate)

  def Changeset(data, params \\ %{}) do
    data
    |> cast(params, @fields)
    |> validate_required([:ide, :startdate])
  end
end
```

Figure 8 : Schéma de la base de données

Nous avons également réalisé la migration de la base pour pouvoir, si besoin, modifier le schéma depuis un programme Elixir. Nous avons ensuite, pour tester la bonne création de notre base de données, insérer des données ainsi que réaliser des requêtes comme nous pouvons le voir ci-dessous.

```
iex(7)> Repo.insert %Item{ide: 4, startdate: DateTime.utc_now}

12:06:25.266 [debug] QUERY OK db=16.2ms
INSERT INTO "farmbdd" ("ide","startdate") VALUES ($1,$2) RETURNING "id" [4, {{2018, 4, 24}}, {10, 6, 25, 248340}}]
{:ok,
 %Farmbot.Item{
  __meta__: #Ecto.Schema.Metadata<:loaded, "farmbdd">,
  enddate: nil,
  id: 3,
  ide: 4,
  startdate: #DateTime<2018-04-24 10:06:25.248340Z>
 }}
```

Figure 9 : Insertion

```

iex(8)> Repo.one from i in Item, select: count(i.id)

12:07:00.079 [debug] QUERY OK source="farmbdd" db=1.8ms
SELECT count(f0."id") FROM "farmbdd" AS f0 []
3

```

Figure 10 : Requête

Nous savons donc que notre base de données a bien été créée et que les actions effectuées depuis l'invite de commande Elixir nous donnent les réponses souhaitées. Une fois toutes ces étapes réalisées nous pouvons réaliser des requêtes pour interroger la base dans notre programme. On réalise donc dans un premier temps une fonction qui viendra supprimer les événements avant la date actuelle. Ensuite on implémente une autre fonction qui vient parcourir la table pour récupérer les "startdate" de chacun des événements. Une fois cela effectué, on compare toutes ces dates pour déterminer laquelle est la prochaine à arriver donc qu'elle est la date de début du prochain événement.

On utilise pour cela les fonctions du module Enum telles que fetch ou tolist. Du fait que l'on utilise une base de données Ecto la syntaxe est différente et un peu complexe. Une fois cette date bien récupérée, il suffit de soustraire celle-ci à la date actuelle pour récupérer le temps qu'il reste avant le début du prochain événement. On passe ensuite ce résultat à la fonction qui gère l'allumage de l'Arduino pour que celui-ci ne s'allume qu'une fois le temps écoulé. On regarde également si ce temps d'attente est inférieur à cinq secondes. En effet, il faut environ cinq secondes à l'Arduino pour s'allumer et être fonctionnel donc si le temps d'attente est inférieur à cinq secondes, il n'y a pas d'intérêt à éteindre.

```

20:02:44.976 [debug] QUERY OK source="farmbdd" db=0.1ms
SELECT f0."startdate" FROM "farmbdd" AS f0 WHERE (f0."startdate" > $1) [{2018,
5, 24}, {18, 2, 44, 971655}]
2018-05-25 17:00:07.000000Z
l'Arduino va s'éteindre pendant
82643
secondes

```

Figure 11 : Affichage du temps d'attente

Ce programme est fonctionnel et marche d'après nos tests mais pour le valider totalement il faut l'ajouter dans l'OS du Farmbot. Nous souhaitons qu'il soit appelé à chaque fois que l'information avertissant qu'un événement est terminé est remontée. Cependant nous n'avons pas réussi à réaliser cela à cause de la complexité de l'OS du Farmbot.

2) Modification de l'application Web

a. Présentation

L'API est codée en Typescript en utilisant la librairie React qui permet de créer des composants. Ici, seulement 5 pages html sont modifiées dynamiquement suivant le passage d'un onglet à un autre. React est accompagné de Redux, une autre librairie permettant de

gérer d'une meilleure façon les applications. En effet, Redux permet de créer la notion de "store" qui contiendra toutes les informations de plusieurs composants React, ce qui est notamment pratique en cas de forte interdépendance. C'est le cas dans notre projet, les différentes entités dépendent d'autres, on ne peut pas faire telle ou telle action si nous n'avons pas modifié l'état de certaines variables. Prenons un exemple, en cas d'appui sur un bouton, nous allons envoyer une requête au serveur, en même temps nous pourrions éventuellement créer un évènement au niveau graphique et en plus modifier l'état d'autres boutons et donc générer d'autres requêtes et d'autres évènements. Nous pouvons voir ci-dessous un cheminement suite à l'appui sur un bouton.

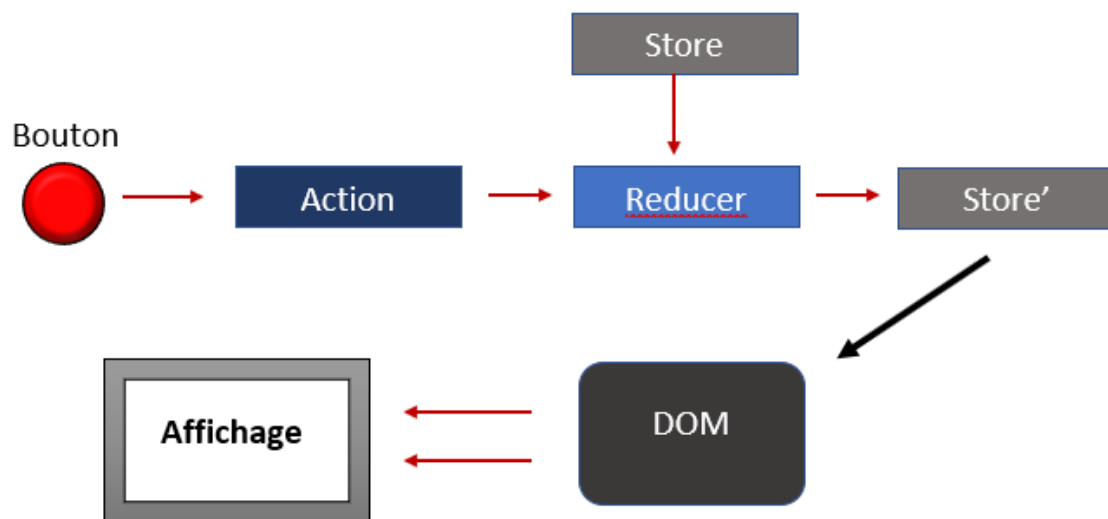


Figure 12 : Organisation de React et Redux

L'appui sur le bouton génère une action, qui va prendre un ensemble de paramètres (store) pour en créer une nouvelle grâce à la fonction Reducer.

Le travail le plus important est de rentrer en profondeur dans le code et dans l'architecture et donc dans tous les dossiers qui composent cette application. L'architecture est complexe mais nous sommes capables discerner très vite les pages html de connexion, d'erreur mais surtout la page principale. Cette page contient tous les différents onglets que nous pouvons voir sur la figure suivante.

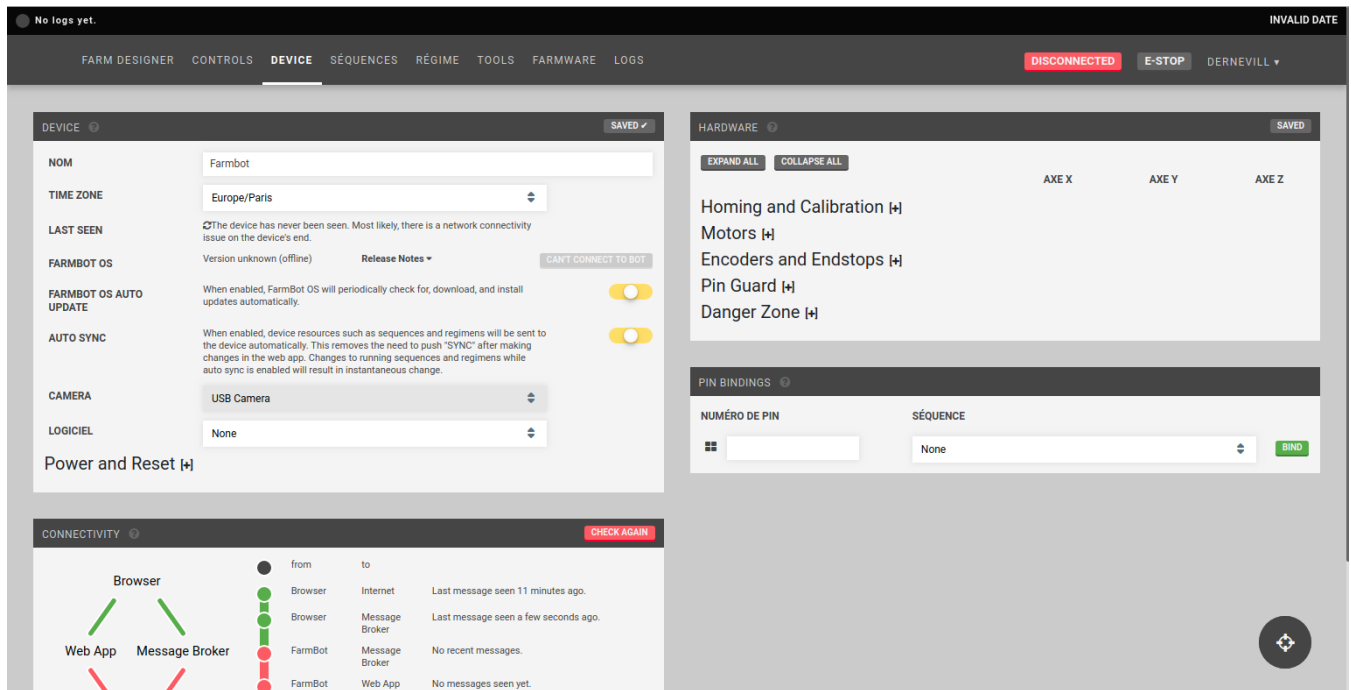


Figure 13 : onglet *Device* de l'application web

Nous pouvons remarquer qu'il y a plusieurs onglets correspondants à différentes fonctionnalités données à l'utilisateur. Chaque page est représentée par un répertoire du même nom. Ce répertoire comporte un fichier de type "index.tsx" et tous les composants de cette partie. Si l'on prend l'exemple de l'onglet "device", il y a plusieurs composants (*components*) tels que des boutons ("check again" en rouge dans connectivity) des inputs ou bien du texte. Les onglets sont tous présents dans le dossier nommé *Webpack*.

Nous pouvons donner un exemple d'architecture pour le bouton "check again" en rouge, présent dans le panneau *connectivity*.

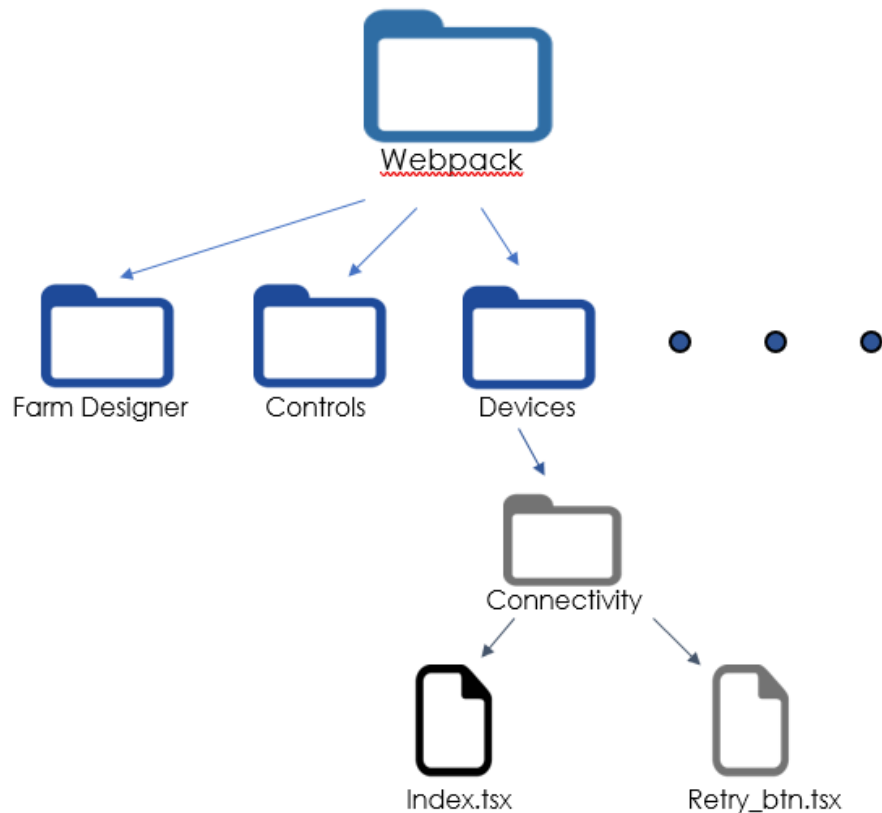


Figure 14 : Architecture de l'onglet device

Le fichier `index.tsx` va utiliser la fonction `render()` de React afin de modifier l'aspect de la page. Dans cette fonction `render`, la partie "connectivity" est organisée, elle fait appel à des composants tels que `retry_btn` (le bouton "check again").

En réalité, il n'y a pas que des composants et un fichier `.ts` central. Comme expliqué précédemment, la bibliothèque React plus Redux est un modèle de type modèle-vue-contrôleur. Il y a donc des fonctions telles que : `action` ou `reducer` (qui correspond à la partie contrôleur, la partie qui modifiera les différents champs) qui seront présentes dans le dossier de l'onglet.

b. Avancé des modifications

Afin de tester l'application, nous avons effectué une modification simple de l'interface en traduisant un des textes de l'image ci-dessus (écrits en anglais). Pour cela nous avons effectué la modification au niveau de la fenêtre "Connectivity" :

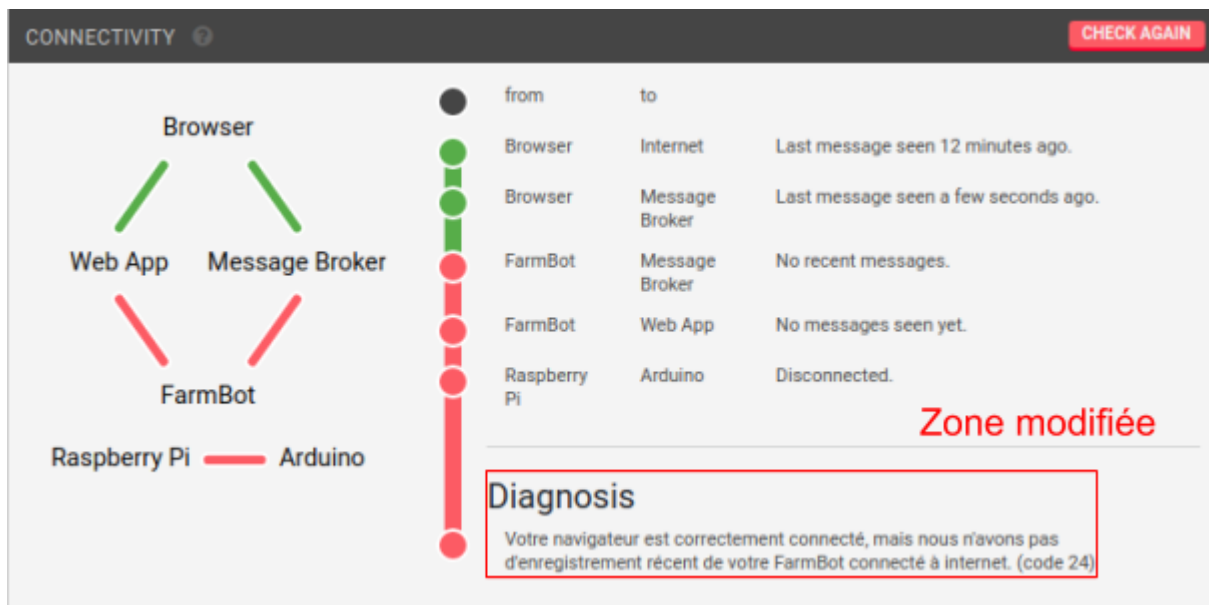


Figure 15 : fenêtre connectivity modifiée

Pour permettre cette modification, nous avons modifié la partie *Diagnosis* grâce au fichier suivant :

```

1 import { trim } from "../../util";
2 import { t } from "i18next";
3
4 export namespace DiagnosticMessages {
5   // "SCV good to go, sir." is also appropriate.
6   export const OK = t("All systems nominal.");
7
8   export const MISC = trim(t("Some other issue is preventing FarmBot from
9     working. Please see the table above for more information."));
10
11   export const TOTAL_BREAKAGE = trim(t("There is no access to FarmBot or the
12     message broker. This is usually caused by outdated browsers
13     (Internet Explorer) or firewalls that block WebSockets on port 3002."));
14
15   export const REMOTE_FIREWALL = trim(t("FarmBot and the browser are both
16     connected to the internet (or have been recently). Try rebooting FarmBot
17     and refreshing the browser. If the issue persists, something may be
18     preventing FarmBot from accessing the message broker (used to communicate
19     with your web browser in real-time). If you are on a company or school
20     network, a firewall may be blocking port 5672."));
21
22   export const WIFI_OR_CONFIG = trim(t("Votre navigateur est correctement connecté,
23     mais nous n'avons pas d'enregistrement récent de votre FarmBot connecté à
24     internet."));
25

```


Code modifié

Figure 16 : Modification du code de la partie Diagnosis

Pour reprendre notre objectif principal, nous souhaitons ajouter un onglet de conseil à côté des autres. Il faut donc trouver les différents liens permettant de créer ce fichier et surtout d'afficher un rendu propre et organisé.

Pour cela, l'étude du dossier "webpack n'est pas suffisante". En effet, il faut s'intéresser à la route utilisée pour afficher des fichiers.

Deux fichiers sont très importants, celui présentant la modification du chemin et celui qui permet l'ajout du nouvel onglet.




```
export const links = [
  { name: "Farm Designer", icon: "leaf", slug: "designer" },
  { name: "Controls", icon: "keyboard-o", slug: "controls" },
  { name: "Device", icon: "cog", slug: "device" },
  { name: "Sequences", icon: "server", slug: "sequences" },
  { name: "Regimens", icon: "calendar-check-o", slug: "regimens" },
  { name: "Tools", icon: "wrench", slug: "tools" },
  { name: "Farmware", icon: "crosshairs", slug: "farmware" },
  { name: "Logs", icon: "list", slug: "logs" },
  { name: "NouvelOnglet", icon: "list", slug: "NouvelOnglet" },
];

export const NavLinks = (props: NavLinksProps) => {
  const currPageSlug = getPathArray()[2];
  return <div className="links">
    <div className="nav-links">
      {links.map(link => {
        const isActive = (currPageSlug === link.slug) ? "active" :
        return <Link
          to={"/app/" + link.slug}
          className={` ${isActive}`}
          key={link.slug}
          onClick={props.close("mobileMenuOpen")}>
            <i className={`fa fa-${link.icon}`} />
            {t(link.name)}
          </Link>
        }
      )}
    </div>
  </div>
```

Dans le dossier NavLinks, nous établissons le chemin pour aller au nouvel onglet.

Tout d'abord, nous avons rajouté le lien "NouvelOnglet" dans la liste des liens. Dans la fonction *NavLinks*, nous retournons le champ qui permet de créer le chemin de cet onglet. Cela permet de donner le chemin suivant :
/app/NouvelOnglet

Figure 17 : Création du chemin jusqu'au nouvel onglet



```
111   page("app/tools",
112     () => import("./tools/index"),
113     "Tools"),
114   page("app/logs",
115     () => import("./logs/index"),
116     "Logs"),
117   page("...",
118     () => import("./404"),
119     "FourOhFour"),
120   //rajout du nouvel onglet : chemin
121   page("app/NouvelOnglet",
122     () => import("./NouvelOnglet/index"),
123     "NouvelOnglet"),
124 ]
125 };
```

Nous sommes obligés de charger les liens dans le fichier. Nous devons donc modifier le fichier suivant nommé "route_config". Le programme est composé de la fonction *page* qui prend en paramètre le chemin est associé ce chemin à un titre d'onglet ou de sous-onglet.

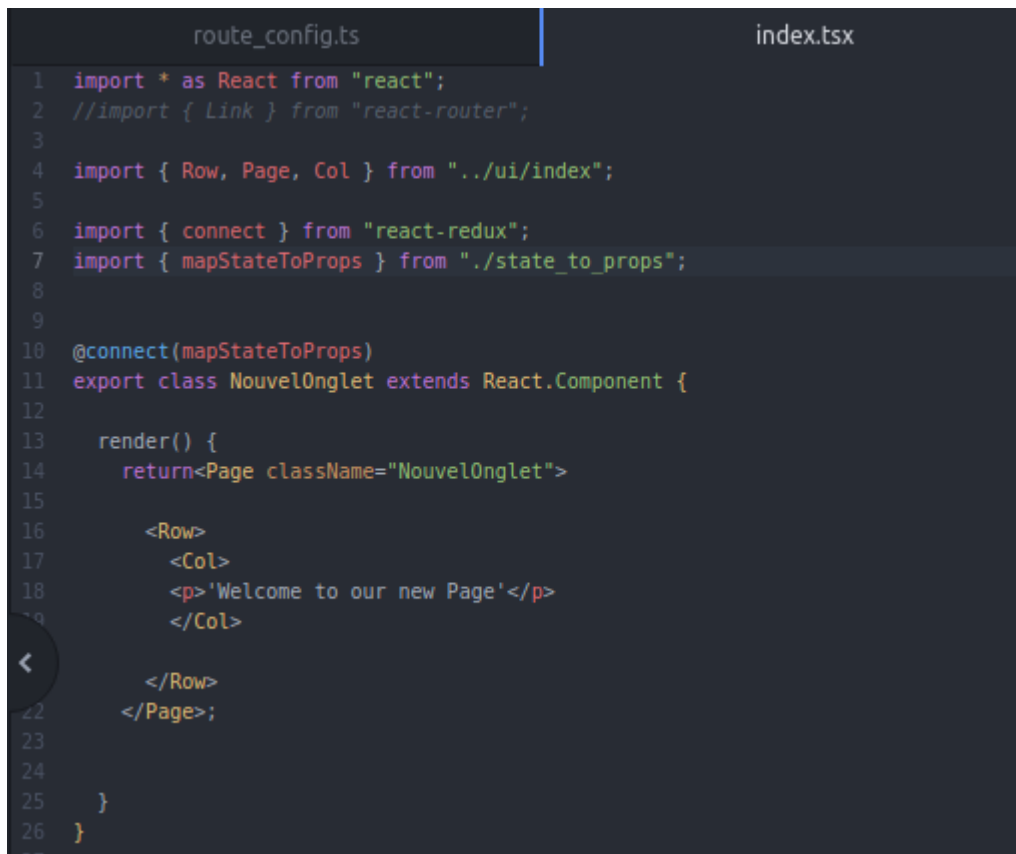
De plus, nous liions un chemin avec un dossier contenant le code de cet onglet.

Figure 18 : Référence au dossier du nouvel onglet

Nous avons donc ajouté le dossier `NouvelOnglet` à notre chemin (prédéfini dans le fichier précédent) dans la fonction `topLevelRoutes`.

Finalement, cette fonction sera "affichée" dans le fichier `route.tsx`, dans le `render()`.

Après ces 2 étapes, il faut ajouter le contenu de l'onglet, 'est à dire le dossier présent dans le webpack contenant les différents composants.



```
route_config.ts                                     index.tsx
1  import * as React from "react";
2  //import { Link } from "react-router";
3
4  import { Row, Page, Col } from "../ui/index";
5
6  import { connect } from "react-redux";
7  import { mapStateToProps } from "../state_to_props";
8
9
10 @connect(mapStateToProps)
11 export class NouvelOnglet extends React.Component {
12
13   render() {
14     return<Page className="NouvelOnglet">
15
16       <Row>
17         <Col>
18           <p>'Welcome to our new Page'</p>
19         </Col>
20
21       </Row>
22     </Page>;
23
24   }
25 }
26 }
```

Figure 19 : code du nouvel onglet

Dans ce fichier, nous avons juste créé une classe affichant un simple rendu, nous souhaitons afficher un texte 'Welcome to our new Page'.

Voici le rendu de cette seconde étape :

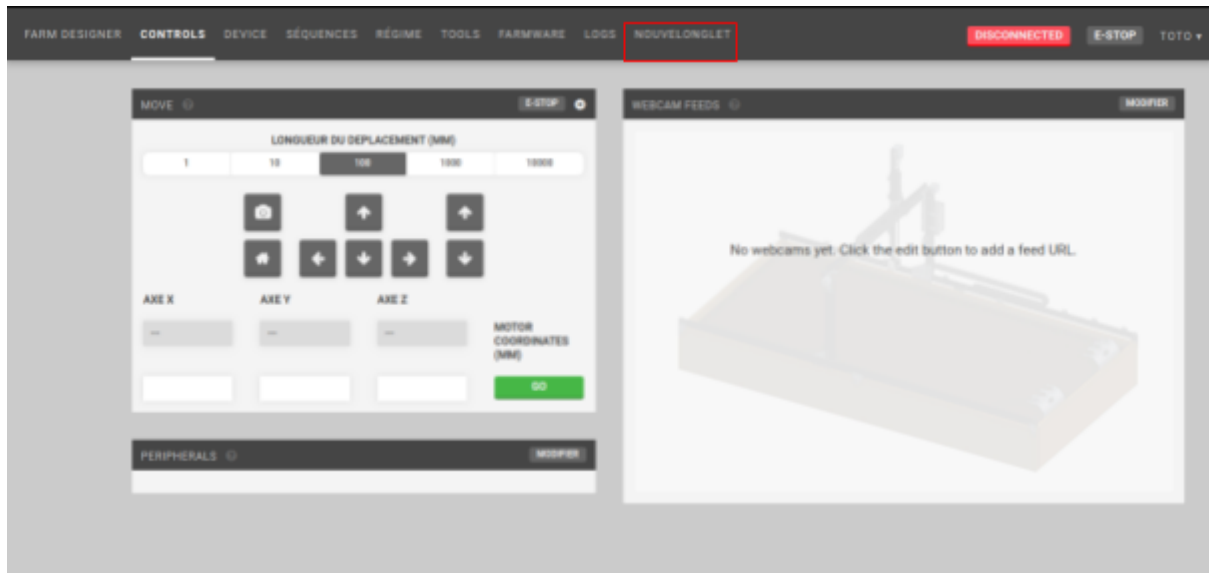


Figure 20 : Application Web avec nouvel onglet

Cependant, notre programme n'arrive pas à compiler, lorsque nous ouvrons l'onglet, nous avons l'erreur 'Page not found'. Nous n'avons pour le moment pas réussi à régler ce problème.

Pour ce qui est du contenu de la page, nous avons juste créé une page avec du texte. De plus, nous avons commencé à travailler sur le rendu en Node.js. Nous avons commencé par créer une base de données (sqlite3 dans notre cas) contenant des légumes et des dates. Le programme indique le légume à planter selon la date actuelle en prenant en compte les légumes sélectionnés par l'utilisateur. C'est encore un programme simple. Mais le but est de le continuer pour obtenir un rendu graphique sur le nouvel onglet en connectant ce programme aux légumes sélectionnés par l'utilisateur dans l'application web.

4. Objectif n°2

Nous devons réaliser une station météo pour le Farmbot grâce à un capteur de pression, un capteur de température et un anémomètre. Dans un premier temps nous avons effectué les branchements vers l'Arduino avec des résistances. Nous avons le montage suivant :

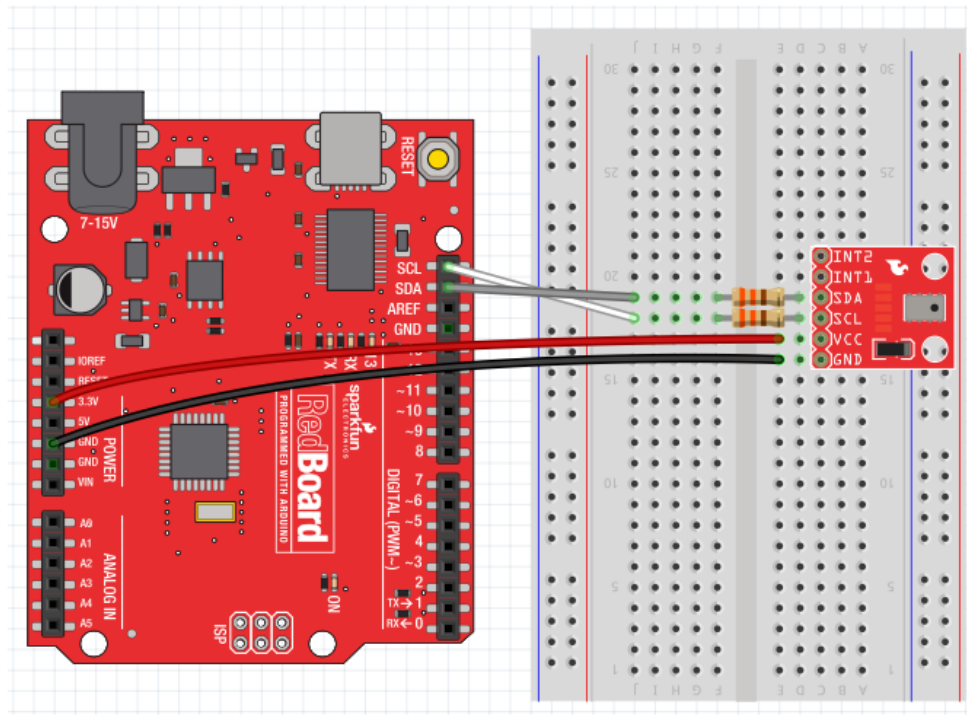


Figure 21 : Montage du capteur de pression

Le code Arduino ressemble à ceci :

PressureSparkfun | Arduino 1.6.7 Hourly Build 2015/11/13 11:42
Fichier Édition Croquis Outils Aide

```

PressureSparkfun $
.setoversamplerate(byte) Sets the # of samples from 1 to 128. See datasheet.
.enableEventFlags() Sets the fundamental event flags. Required during setup.

*/

#include <Wire.h>
#include "SparkFunMPL3115A2.h"

//Create an instance of the object
MPL3115A2 myPressure;

void setup()
{
  Wire.begin();          // Join i2c bus
  Serial.begin(9600);    // Start serial for output

  myPressure.begin();    // Get sensor online

  // Configure the sensor
  //myPressure.setModeAltimeter(); // Measure altitude above sea level in meters
  myPressure.setModeBarometer(); // Measure pressure in Pascals from 20 to 110 kPa

  myPressure.setOversampleRate(7); // Set Oversample to the recommended 128
  myPressure.enableEventFlags(); // Enable all three pressure and temp event flags
}

void loop()
{
  float pressure = myPressure.readPressure();
  Serial.print("Pressure(Pa):");
  Serial.print(pressure, 2);

  Serial.println();
}

```

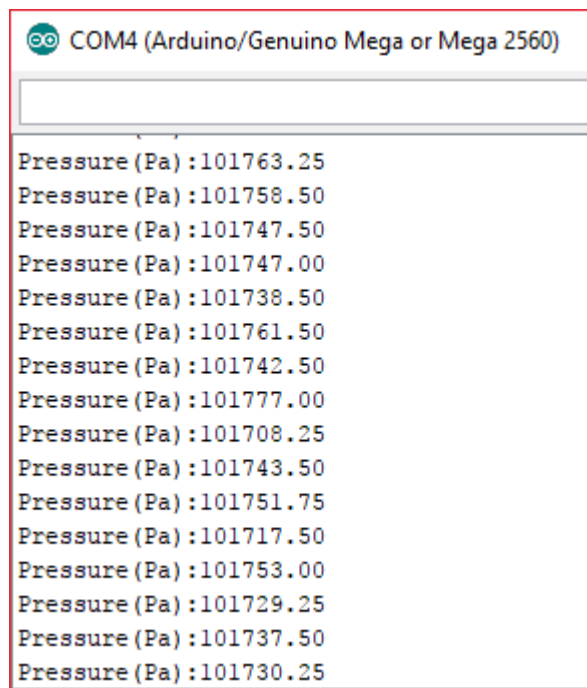
Figure 22 : Code Arduino du capteur de pression

Le code Arduino a été pris depuis une librairie disponible avec le capteur sur le site de sparkfun ici : <https://learn.sparkfun.com/tutorials/mpl3115a2-pressure-sensor-hookup-guide>. Dans ce code, nous mesurons la pression toute grâce à la ligne

float pressure = myPressure.readPressure();

Nous n'avons pas encore trouvé le moyen de mesurer la température avec ce capteur.

En téléversant le code sur l'Arduino, nous obtenons la mesure de la pression ici



The screenshot shows the serial monitor window for a COM4 (Arduino/Genuino Mega or Mega 2560) connection. The output displays a series of 16 pressure readings in Pascals (Pa), ranging from approximately 101729.25 to 101763.25. The text is as follows:

```
COM4 (Arduino/Genuino Mega or Mega 2560)
Pressure (Pa) : 101763.25
Pressure (Pa) : 101758.50
Pressure (Pa) : 101747.50
Pressure (Pa) : 101747.00
Pressure (Pa) : 101738.50
Pressure (Pa) : 101761.50
Pressure (Pa) : 101742.50
Pressure (Pa) : 101777.00
Pressure (Pa) : 101708.25
Pressure (Pa) : 101743.50
Pressure (Pa) : 101751.75
Pressure (Pa) : 101717.50
Pressure (Pa) : 101753.00
Pressure (Pa) : 101729.25
Pressure (Pa) : 101737.50
Pressure (Pa) : 101730.25
```

Figure 23 : Mesure de la pression

Pour le capteur d'humidité nous avons un autre code différent du premier. Nous avons tout d'abord connecté une LED sur le port 3 de l'Arduino. Nous avons le code suivant :

```
Humidit_
int PinAnalogiqueHumidite =0;
int hsol;
int secheresse;
int Pinled = 3;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(PinAnalogiqueHumidite, INPUT);
  pinMode(Pinled,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  hsol = analogRead(PinAnalogiqueHumidite);
  Serial.print("Humidite ");
  Serial.println(hsol);
  delay(1000);
  if (hsol <50){
    digitalWrite(Pinled,HIGH); //Led allumée
  }
  else
  {
    digitalWrite(Pinled, LOW); //Led éteinte
  }
}
```

Figure 24 : Code Arduino pour la sonde d'humidité

Sur notre écran nous avons le résultat suivant :

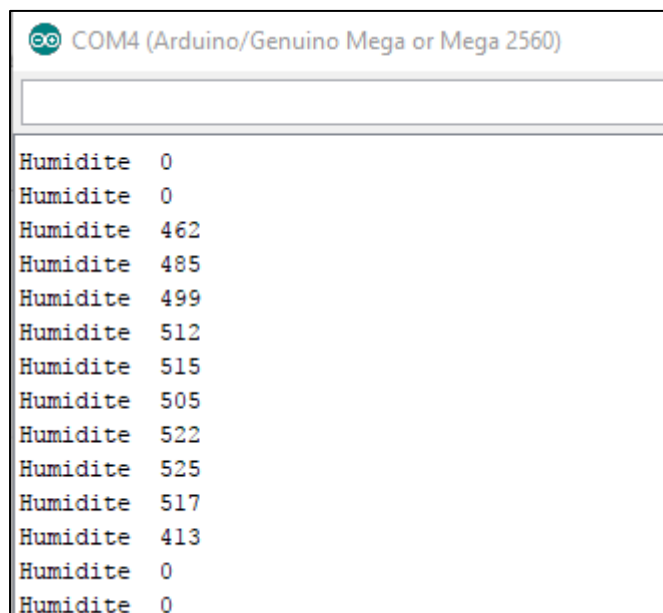


Figure 25 : Valeurs de la sonde d'humidité

En effet, les valeurs de la sonde d'humidité vont de 0 à 530. Nous pouvons coder des prédictions d'événements en fonction de ces valeurs.

Comment programmer les actions du Farmbot en fonction des valeurs ?



Figure 26 : éditeur de séquence

L'éditeur de Séquence de l'IHM sur la figure ci-dessus, nous montre comment nous pouvons programmer des actions du Farmbot. Ici, il est très facile de programmer le comportement d'une station météo. Tout devient facile. Il suffit de brancher les éléments de météo sur un des ports analogiques du Farmduino et, sur l'IHM, renseigner le port sur lequel on souhaite lire les informations. Il faut faire attention à bien renseigner le mode du PIN car on a des données analogiques qui entrent. Après cela il faut renseigner la condition par exemple si sur la PIN4 on a une valeur inférieure à 50, le Farmbot exécutera une séquence qui arrosera la plante se trouvant aux coordonnées {X;Y;Z}. Dans le cas contraire, il ne ferait rien.

Sur l'IHM il ne nous restera qu'à créer les séquences d'arrosage et de plantage en fonction de la météo. Nous pouvons faire en sorte que le Farmbot puisse envoyer des messages sur les conditions météorologiques comme nous pouvons le voir ci-dessous :



Figure 27 : Envoi d'un message depuis le Farmbot

5. Objectifs restants

Nous avons, pour le moment, passer beaucoup de temps sur la compréhension du projet et sur les différents aspects qui nous étaient complètement inconnus au début. En ce moment, nous passons beaucoup de temps sur les objectifs principaux énoncés ci-dessus mais nous gardons en tête les objectifs de priorité basse.

Le premier étant de gérer la qualité des eaux pluviales, nous ne pourrons pas le faire tant que le Farmbot ne sera pas installé au jardin. Par ailleurs, la cuve d'arrivée d'eau n'a pas été commandé.

L'objectif de créer un outil retournant la terre a été commencé mais ne peut pas être continué tant que l'objectif n°1 sera actif. Une autre condition est que le Farmbot soit installé. Nous avons déjà assemblé beaucoup de pièces, ce qui nous a permis de voir comment étaient fixées les différentes têtes. Nous avons, pour le moment créé un prototype d'outil qui pourra s'enclencher à la tête principale du robot. Ce prototype a été réalisé avec le logiciel gratuit 'onshape'. Vous pouvez voir la forme du futur outil qui ressemble globalement à une petite fourche.

Pour retourner la terre, il faut d'abord enlever les mauvaises herbes. Si l'on réussit à créer une séquence visant à bêcher le potager, nous devons d'abord ajouter l'action de détecter et de détruire les mauvaises herbes, sachant que cette fonctionnalité est déjà présente.

Ensuite, nous pourrions nous servir de la tête comme une fourche, en soulevant la terre au fur et à mesure des passages. Pour cela, les trois pâtes doivent être inclinées, et le bras devra avancer de façon périodique sur de petites distances.

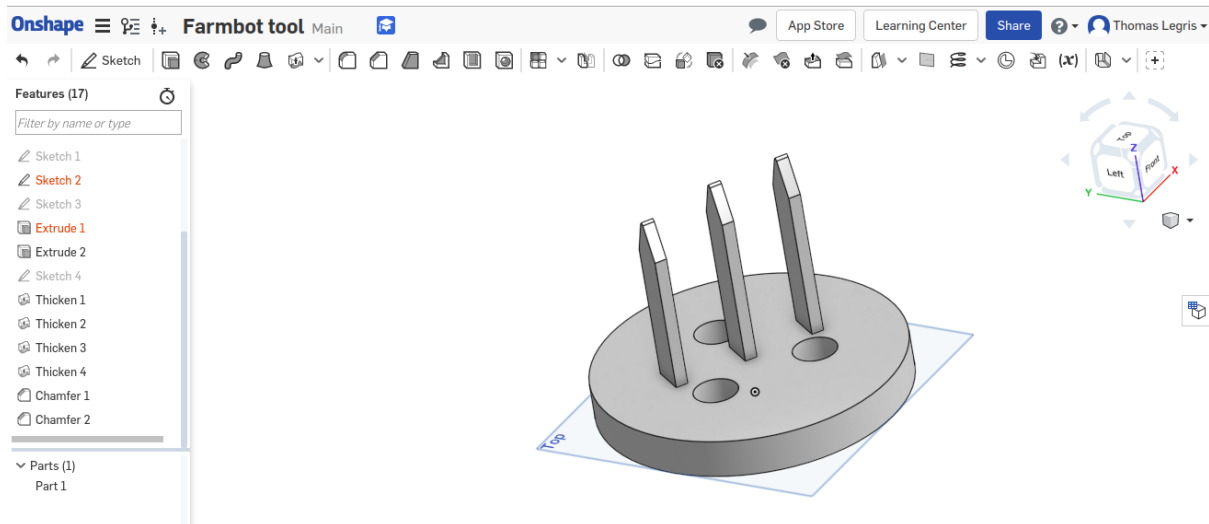


Figure 28 : aperçu en 3D de l'outil

Nous pouvons voir ci-dessous la tête principal fixé au bras du Farmbot. La tête a été conçu pour être facilement utilisable avec d'autres têtes. La connectique se fait via différents éléments dont notamment trois aimants comme nous pouvons le voir sur l'image ci-dessous.

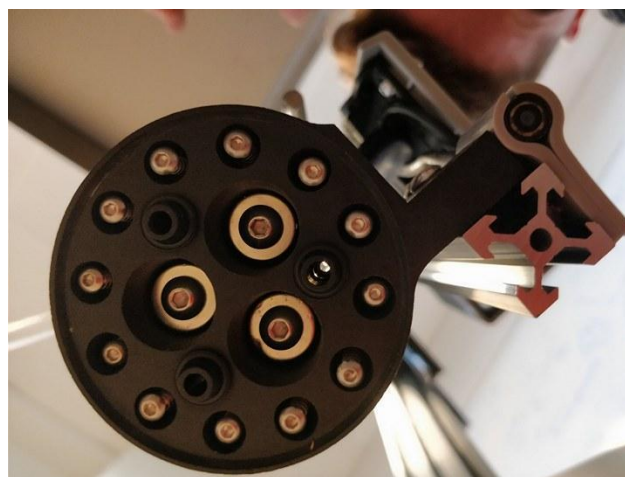


Figure 29 : Tête du bras robot

6. Evolution du site Web

Précédemment, nous y avons affiché le cahier des charges et nos objectifs. Nous rappelons que nous travaillons avec Jekyll. Nous avons rajouté quelques posts montrant l'avancement de notre projet. Voici la première page du site présentant l'ensemble des posts que nous avons.

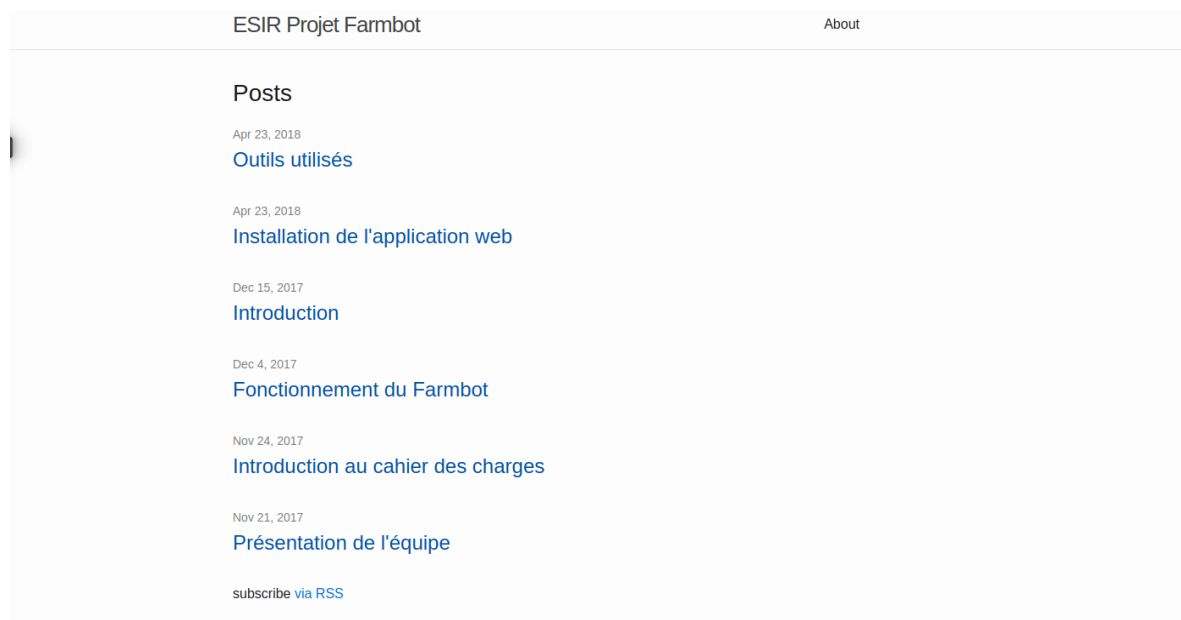


Figure 30 : Site web du farmbot

Nous avons inclus des posts expliquant un peu les langages utilisés et notre progression quant à l'installation de l'application web. Sur ce site, nous voulons donner des conseils au lecteur lui permettant de mieux réussir les différentes étapes et comprendre le fonctionnement du système.

III. Problèmes et solutions

1. Evolution du site Web

Notre avancé dans le projet a été retardé suite à quelques difficultés. Celles-ci se trouvent à différents niveaux (installation et modification de l'application web et gestion de l'énergie).

2. Problème d'installation

Nous avons mis beaucoup de temps à faire fonctionner l'application web. Elle a été difficile à installer pour de nombreuses raisons. Premièrement, les étapes étaient nombreuses et complexes. Elles ne sont pas forcément toutes expliquées. C'est pour cela que nous avons eu recours à différentes aides.

De plus, il manquait quelques étapes comme l'installation du module jest (permettant de tester les codes typescript). Par ailleurs, l'application n'est pas vraiment stable.

3. Modification du code de l'application web

Les codes source de l'application web sont très complexes et très nombreux. Nous ne savons pas comment ils sont reliés et nous ne pouvons pas prévoir le comportement de l'application suite à une modification. De plus, les langages de programmations utilisés sont inconnus pour nous (Typescript associé à React et Redux).

Nous avons créer l'onglet mais il nous reste beaucoup de travail et d'effort pour arriver à réaliser cet objectif.

4. Gestion de l'énergie via la modification de l'OS

Comme c'est le cas de l'application web, l'OS du Farmbot est composé d'une multitude de fichiers dont la majorité est codée en Elixir. Le premier obstacle est la compréhension de ce langage. Il nous est totalement inconnu et donc sa compréhension à nécessité une implication importante. Dans un deuxième temps, nous nous sommes heurtés à des soucis dû aux outils utilisés par le Farmbot comme Ecto et PostgreSQL. En effet, malgré les différents tutoriels disponibles sur le sujet, les étapes pour leurs utilisations sont nombreuses et complexes.

IV- Les solutions mises en place et à mettre en place

Pour résoudre les différents problèmes que nous avons cité dans la partie précédente, nous avons mis en place des solutions de nature variées.

5. Problème d'installation

En s'appuyant sur les différentes issues postées sur le forum Farmbot, nous avons réinstallé pas à pas l'IHM afin de vérifier que chaque étape fonctionne, en limitant au plus les avertissements. On nous a conseillé de réinstaller yarn, c'est une technologie permettant de gérer le projet en lui même, puis d'ajouter jest. Tout ceci, couplé à la génération d'une clé (avec la commande `rake secret generate`). Rick Carlino, co-fondateur du projet nous a été d'une grande aide pour l'installation, très disponible et pédagogue afin de chercher une solution à notre problème.

Par ailleurs, Mr Bourcier nous a donné une image VM, nous nous sommes rendus compte d'une erreur lors de notre installation. Dans le fichier de config "application.yml", introduit plus haut, il ne fallait pas supprimer les champs en entiers mais juste supprimer les valeurs. Lorsque l'on lançait l'application, elle était donc instable.

6. Modification du code de l'application web

Pour ce qui est de l'application web, nous avons commençons par étudier l'architecture et le langage utilisé. On a d'abord modifié les éléments de textes pour visualiser un peu le comportement du site par rapport aux modifications. Puis nous essayons de prendre en main les différents langages utilisés afin de mieux comprendre comment le code source est rédigé.

Pour cela, nous avons décidé d'apprendre à utiliser React. Dans le cadre du projet IOT que l'on réalise avec Mr Bourcier, nous avons eu le choix de réaliser une API en Java - Android ou en web, ceci afin de s'initier au langage du web (JS plus React). Nous travaillons actuellement sur les deux technologies. Nous avons pris connaissance de certain module tel que React redux.

7. Gestion de l'énergie via la modification de l'OS

En ce qui concerne le problème dû au fait que nous ne connaissions pas le langage Elixir, nous avons décidé qu'il était préférable de consacrer une partie de notre temps à apprendre ce langage via différents sites internet et vidéos prévues à cet effet. Pour ce qui est de la programmation, nous avons dans un premier temps réalisé notre code en JavaScript puis nous avons fait l'équivalent en Elixir. Au niveau de l'utilisation de Ecto, nous avons principalement été aidé par les forums car les exemples d'utilisation ne sont pas très

nombreux. Nous avons notamment utilisé le forum "*Elixirforum*". Sur ce forum, nous avons obtenus des réponses à nos questions de façon très rapide par les autres utilisateurs.

Conclusion

En conclusion de ce projet, nous pouvons dire que nous sommes un peu dessus de la finalité car nous n'avons pas réussi à remplir nos objectifs. Tout d'abord, au niveau de l'installation du Farmbot en lui-même le fait de ne pas l'avoir monté entièrement et fait fonctionner est décevant. Ensuite, pour le reste du projet, nous n'avons pas imaginé l'ampleur de la partie compréhension lorsque l'on travaille à partir d'un projet existant. En effet cette partie-là a été très compliqué pour nous (notamment à cause de l'usage de langage et d'outil inconnus pour nous) et nous a mis en retard sur nos objectifs. De plus, l'organisation de notre équipe sur l'ensemble du projet n'était peut-être pas optimale pour pouvoir avancer plus rapidement. Mettre des objectifs moins élevés et plus courts aurait pu être une solution pour avancer étapes par étapes. Nous avons cependant bien avancé dans certains objectifs comme la gestion de l'alimentation et la mise en place de la station météo. Notre perte de temps du début fait que nous sommes en décalage et que si nous avions passé moins de temps sur cette partie nos objectifs seraient beaucoup plus avancé.

Ce projet annuel nous a permis d'apprendre à travailler en équipe sur un projet de longue durée. Cela nous a obligé à s'organiser et communiquer pour que l'équipe avance. Il nous a aussi permis de nous familiariser à certain langage tel que l'Elixir. Il est pour nous une bonne expérience qui pourra nous être utile dans le futur même si nous n'avons pas totalement rempli nos objectifs.

Conclusion personnelle :

Yoann :

Un projet ambitieux et enrichissant. J'ai d'abord travaillé sur le site web puis sur l'IHM et enfin sur la station météo. L'objectifs n'était pas totalement atteint, cependant. J'ai passé beaucoup de temps à chercher à comprendre le système et c'est ce qui m'a permis d'apprendre des choses sur le langage Elixir, Jekyll, le fonctionnement du Farmbot, la station météo. Je pense que ma montée en compétence me permettra d'aider les autres à travailler sur le Farmbot.

Thomas :

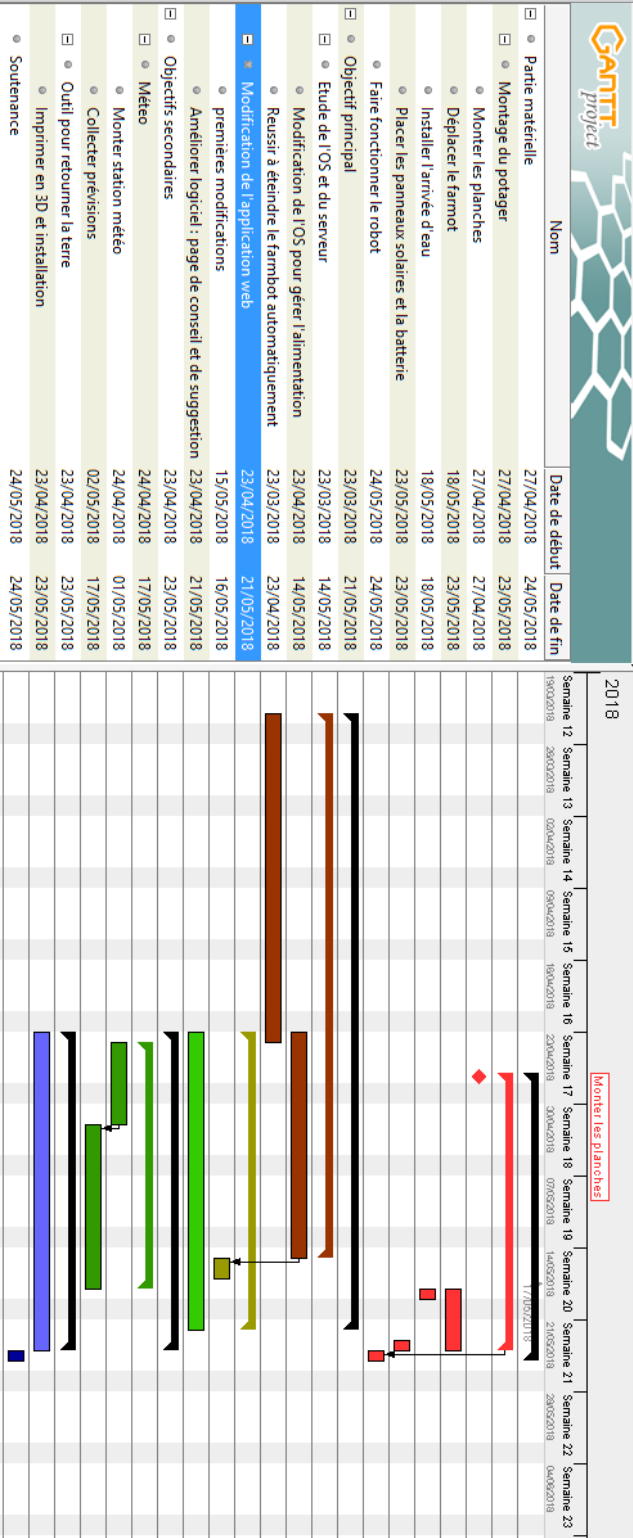
Travailler sur ce projet tout au long de l'année a été bénéfique autant humainement que scientifiquement. Nous avons appris à travailler en équipe sur une période longue et cela a été très enrichissant. Je n'avais pas l'habitude de travailler de façon totalement autonome, cela m'a permis de développer cette compétence et d'apprendre à utiliser certains outils de moi-même. J'ai appris à utiliser le langage Type script (très proche de javascript) en parallèle du cours de web du 1er semestre mais aussi la librairie React. J'ai eu des beaucoup difficultés pour installer l'application web en local, cela m'a découragé mais grâce à différentes aides j'ai pu me remettre dans le projet en éclaircissant l'organisation de

l'app et en apprenant à utiliser React. Finalement, j'ai pu réaliser qu'un projet avec des deadlines était complexe et qu'on pouvait très vite engranger des retards.

Théo :

Pour ma part, ce projet a été enrichissant. J'ai travaillé sur la partie gestion de l'énergie du Farmbot. Même si l'objectif n'est pas complètement atteint, celui-ci est bien avancé. Le fait d'utiliser un nouveau langage (Elixir) et de nouveaux outils (Ecto, PostgreSQL...) a été au début un frein à mon avancement mais, une fois bien compris comment ils fonctionnent et avec un peu de recul, je trouve que cela a été bénéfique et m'a permis d'élargir mon éventail de compétences.

Annexes



Références

(s.d.). Récupéré sur <https://www.postgresql.org/>

(s.d.). Récupéré sur React: <https://reactjs.org/>

Barroso, B. (2014). *Meet Ecto*. Récupéré sur toptal: <https://www.toptal.com/elixir/meet-ecto-database-wrapper-for-elixir>

Carlino, R. (s.d.). Récupéré sur FarmBot Web App: <https://my.farmbot.io>

Combelles, C. (2016, 04 16). Récupéré sur Anybox: <https://anybox.fr/blog/appli-web-histoire-du-dom-react-redux>

Geoffrey Lessel. (s.d.). Récupéré sur <http://geoffreylessel.com/>

MOL3115A2 Pressure Sensor. (s.d.). Récupéré sur Sparkfun:
<https://learn.sparkfun.com/tutorials/mpl3115a2-pressure-sensor-hookup-guide>

Une brève histoire de DOM. (2016, Avril 4). Récupéré sur Anybox:
<https://anybox.fr/blog/appli-web-histoire-du-dom-react-redux>