Maximilien DEUTSCH Romain LHEPT Mathis PAULY Timothé BORDIER Jérémie HUGUES
Antoine POUPON

Projet Transmission de l'information :

Panneaux Solaires

Sommaire

Introduction

Face à l'urgence climatique et à la crise énergétique, il est temps de repenser notre manière de produire et de consommer l'énergie. Les panneaux solaires, emblèmes d'une révolution verte, transforment la lumière du soleil, une ressource gratuite et inépuisable, en électricité ou en chaleur. Derrière cette technologie simpliste se cachent pourtant des défis majeurs qui touchent à l'écologie, à l'économie et à la société tout entière. Ce projet est né d'une question simple mais essentielle : comment, à notre niveau, pouvons-nous contribuer à la construction d'un avenir plus respectueux ? Nous avons choisi d'explorer les panneaux solaires pour en comprendre le fonctionnement, mesurer leur impact réel au quotidien et découvrir leur rôle dans la transition énergétique, qu'il s'agisse d'alimenter une maison, une usine ou dans ce cas un bâtiment à toit plat, en énergie propre.

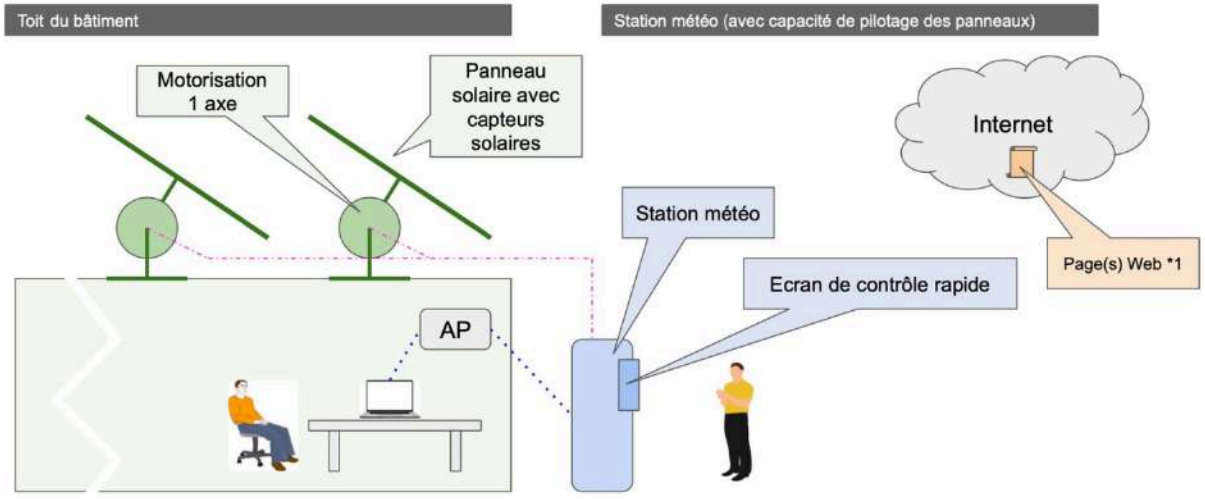
Dans cette étude, nous allons décrypter le principe de fonctionnement des panneaux solaires, puis évaluer leur rendement, leurs limites et leurs conditions d'efficacité; Ensuite nous allons analyser leur intégration dans différents contextes (urbain, rural, industriel, pays en développement…) et nous interroger sur les leviers pour les rendre plus accessibles.
Enfin nous examinerons l'impact environnemental de leur cycle de vie (fabrication, transport, recyclage)et nous mettrons en lumière des projets inspirants qui montrent la voie. Ce travail dépasse le simple exposé technique : il porte une ambition. Celle de démontrer que l'énergie solaire n'est pas qu'une solution parmi d'autres, mais une véritable opportunité de transformer notre façon de produire, de consommer et de vivre avec la planète. Car au-delà de la technologie, le solaire incarne une vision : celle d'une énergie abondante, décentralisée et respectueuse de l'environnement. Une énergie qui n' appartient à personne… et donc à tout le monde.

Présentation du Contexte

Notre système est constitué de deux parties: un panneau solaire et une station météo. La station météo est composée d'un ensemble de capteurs qui indiquent au panneau solaire quelle configuration il doit adopter en fonction de la situation météorologique. L'ensemble permet donc au panneau solaire d'être 100% autonome. Notre panneau solaire et notre station météo sont destinés à être implantés sur les bâtiments des entreprises qui disposent d'un toît plat.

Actuellement, l'ensemble du bâtiment est alimenté exclusivement grâce au réseau électrique national, avec des factures énergétiques dont le coût est élevé, à cause de fortes variations des prix de l'électricité. Le système photovoltaïque que nous concevons, spécialement adapté aux toitures plates, vise à couvrir une part significative des besoins du bâtiment tout en offrant une solution de rentabilité dès les premières années grâce à l'autoconsommation et à la revente du surplus, ainsi que la durabilité et la faible exigence en maintenance. En plus des avantages économiques, le système a pour vocation d'être respectueux de l'environnement car il réduit drastiquement les émissions de CO_2 dues à l'énergie consommée par les entreprises.

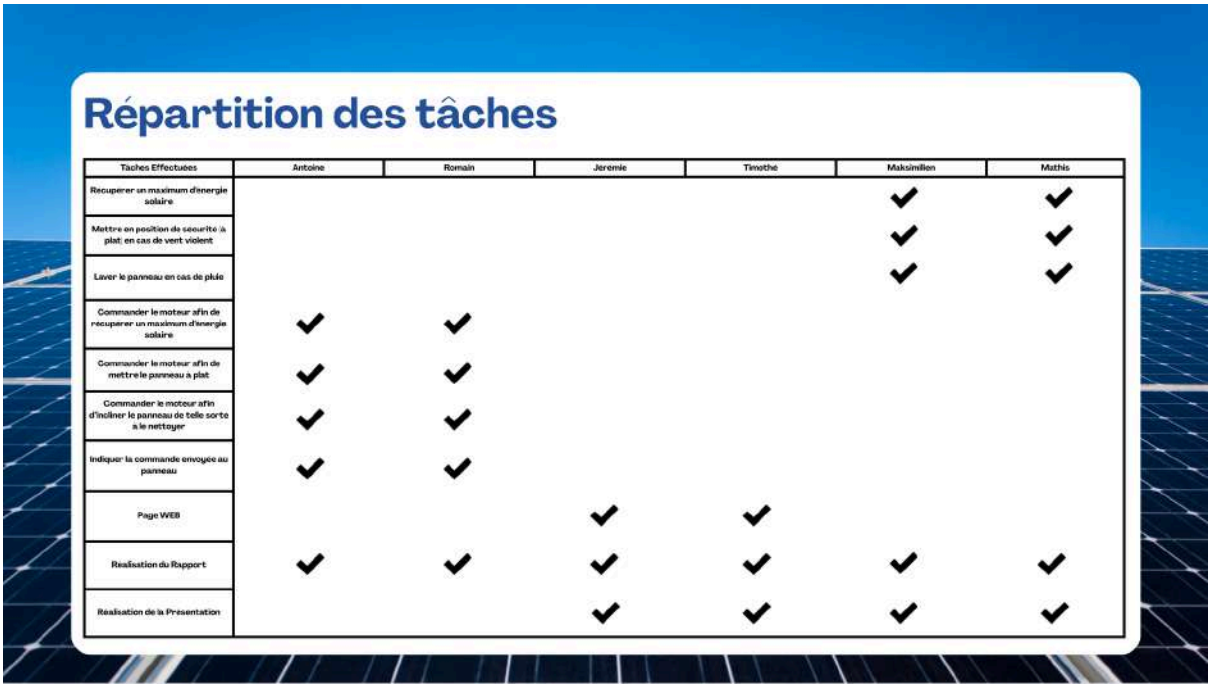
Notre analyse portera sur la faisabilité technique (dimensionnement, orientation, ombrages, intégration structurelle), la durabilité du système sur les années à venir et l'impact positif pour les entreprises. Cette contextualisation concrète dans un bâtiment industriel à toit plat nous permettra d'évaluer la performance réelle de notre solution dans un cas d'usage représentatif des nombreux immeubles d'entreprises existants ou en projet, et de démontrer tout le potentiel du solaire photovoltaïque en milieu professionnel.



Répartition des tâches

Pour ce projet, notre équipe était composée de Jérémie HUGUES, le chef du projet, ainsi que de cinq membres : Antoine POUPON, Romain LHEPT, Timothé BORDIER, Maksimilien DEUTSCH et Mathis PAULY. Nous nous connaissons depuis plus de deux ans à présent ce qui nous a permis d'avoir une bonne cohésion de groupe et de connaître les points forts et faiblesses de chacun.

Pour ce qui est de la réparation des tâches, nous avons choisi de séparer les tâches pour avancer de façon assez efficace même si beaucoup d'entre nous ont aidé à faire des tâches transverses comme la rédaction du rapport par exemple. Voici un diagramme qui synthétise les tâches du projet et les personnes concernées.



Maximilien et Mathis ont été responsables de récupérer un maximum d'énergie solaire, de mettre le panneau en position de sécurité à plat et de laver le panneau en cas de pluie.

Antoine et Romain se sont chargés de commander le moteur pour récupérer un maximum d'énergie solaire, de commander le moteur pour mettre le panneau à plat, de commander le moteur pour incliner le panneau afin de le nettoyer et d'indiquer la commande envoyée au panneau.

Jérémie et Timothé ont développé la page web.

Pour la réalisation du rapport, toute l'équipe a participé.

Enfin, la réalisation de la présentation a été faite par Jérémie, Timothé, Maximilien et Mathis.

Diagramme de Gantt

Pour s'assurer d'être en phase dans l'avancement du projet, nous avons utilisés plusieurs moyens tels que :

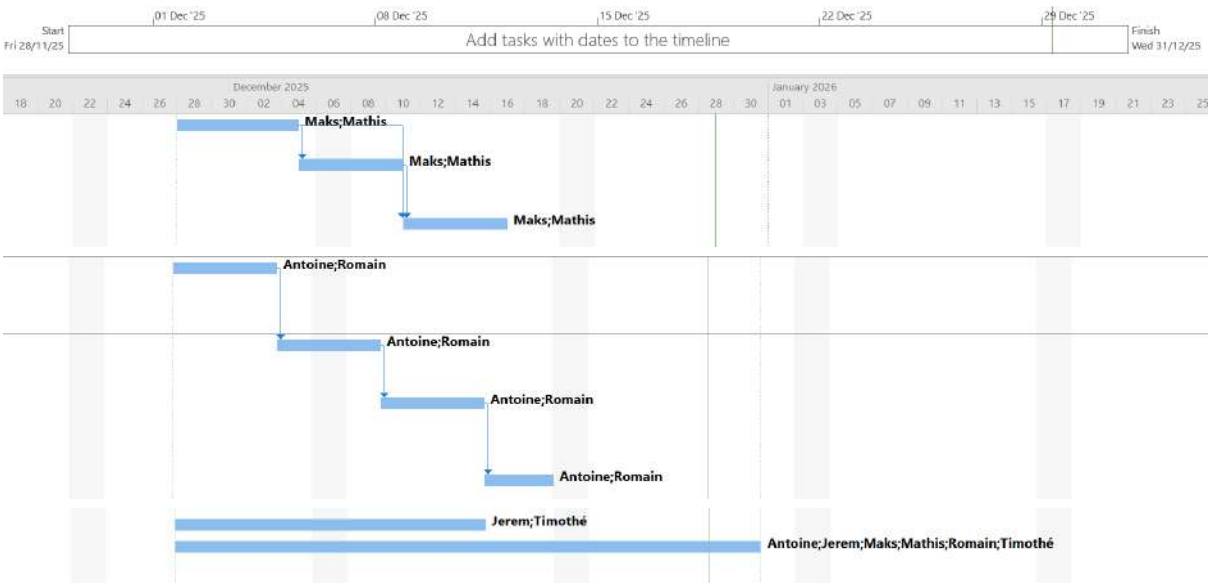
La collaboration en ligne : Nous nous sommes servi d'un drive pour centraliser nos ressources, les codes ou encore le rapport final

Les réunions de suivi : Des réunions en présentiel à l'EPF ou en distanciel par téléphone ont été organisées pour faire le point sur les problématiques que nous avons rencontrées et pour garantir une continuité dans le suivi.

L'encadrement et la rigueur : Nous avons essayé d'anticiper au maximum notre avancement pour les différentes étapes du projet. Cela nous a offert l'opportunité de bénéficier de l'aide de M. SOUDER et de M.SUARD afin qu'il puissent nous donner un avis objectif et rationnel sur le travail déjà réalisé avant les séances de suivi.

Enfin, on a récapitulé les différentes étapes du projet dans un diagramme de Gantt. Celui-ci nous a permis de visualiser toutes les tâches à effectuer et le temps à consacrer à chacune d'elles afin d'achever le projet dans les temps.

Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
	récupérer un maximum d'énergie solaire	5 days	Fri 28/11/25	Thu 04/12/25		Maks;Mathis
	le mettre en position de sécurité (à plat) en cas de vent violent	4 days	Fri 05/12/25	Wed 10/12/25	1	Maks;Mathis
	laver le panneau en cas de pluie	4 days	Thu 11/12/25	Tue 16/12/25	1;2	Maks;Mathis
	Commander le moteur afin de récupérer un maximum d'énergie solaire	4 days	Fri 28/11/25	Wed 03/12/25		Antoine;Romain
	Commander le moteur afin de mettre le panneau à plat	4 days	Thu 04/12/25	Tue 09/12/25	4	Antoine;Romain
	Commander le moteur afin d'incliner le panneau de telle sorte à le nettoyer	4 days	Wed 10/12/25	Mon 15/12/25	5	Antoine;Romain
	Indiquer la commande envoyée au panneau	4 days	Tue 16/12/25	Fri 19/12/25	6	Antoine;Romain
	Page WEB	12 days	Fri 28/11/25	Mon 15/12/25		Jerem;Timothé
	Rapport	24 days	Fri 28/11/25	Wed 31/12/25		Antoine;Jerem;Ma



Étude et Analyse de l'existant

Le développement des énergies renouvelables a conduit à une utilisation croissante de l'énergie solaire dans de nombreux domaines. Cette énergie peut être exploitée à l'aide de différentes technologies de panneaux solaires, dont les deux principales sont les panneaux photovoltaïques et les panneaux solaires thermiques.

Deux familles de panneaux solaires

1) Les panneaux solaires photovoltaïques ont pour fonction de produire de l'électricité à partir du rayonnement solaire. Ils sont constitués de cellules semi-conductrices, le plus souvent en silicium, capables de générer un courant électrique lorsqu'elles sont exposées à la lumière. Selon la technologie utilisée, ces cellules peuvent être monocristallines ou polycristallines. Les panneaux monocristallins se caractérisent par un rendement élevé et une bonne efficacité même lorsque l'ensoleillement est limité, tandis que les panneaux polycristallins représentent une alternative plus économique, mais nécessitent une surface plus importante pour une puissance équivalente.

2) Les panneaux solaires thermiques reposent sur un principe différent. Ils permettent de capter l'énergie solaire afin de la transformer en chaleur, utilisée pour chauffer un fluide, généralement de l'eau. Cette technologie est principalement destinée aux besoins domestiques ou industriels, comme pour la production d'eau chaude sanitaire. En revanche, elle ne permet pas de produire de l'énergie électrique, ce qui limite son usage dans les systèmes électroniques autonomes.

Cadre d'utilisation des technologies solaires

Les technologies solaires sont aujourd'hui déployées dans de nombreux contextes. Elles sont intégrées aux bâtiments résidentiels afin de réduire la consommation énergétique,

utilisées dans le secteur industriel pour alimenter des installations de grande taille, et largement présentes dans des systèmes autonomes tels que les équipements de signalisation, les lampadaires solaires ou les stations météorologiques situées dans des zones isolées. Notre cas est donc une application typique des technologies solaires.

Avantages et inconvénients

L'exploitation de l'énergie solaire présente des avantages majeurs. Il s'agit d'une source d'énergie renouvelable, respectueuse de l'environnement et contribuant à la diminution des émissions de gaz à effet de serre. De plus, une fois le système installé, les coûts de fonctionnement restent faibles.

Cependant, certaines contraintes doivent être prises en compte, notamment l'investissement initial, la variabilité de la production liée aux conditions météorologiques et les enjeux associés au recyclage des panneaux en fin de vie.

Choix du panneau

Enfin, cette étude de l'existant permet d'orienter les choix techniques du projet. Pour une station météorologique autonome pilotant un panneau solaire, l'utilisation d'un panneau solaire photovoltaïque polycristallin apparaît comme une solution pertinente. Associé à une station météo capable d'adapter son fonctionnement en fonction des conditions climatiques (ensoleillement, orientation, température), ce type de panneau permet d'assurer une production d'énergie fiable avec un rendement satisfaisant. Le pilotage du panneau par la station météo contribue à optimiser la production énergétique et à compenser le rendement intrinsèquement inférieur de la technologie polycristalline, tout en limitant les coûts du système.

Analyse Fonctionnelle

Dans le cadre de la démarche de développement et de conception du produit, l'analyse fonctionnelle est un outil essentiel. Elle permet d'identifier et de prendre en compte les besoins et attentes du client, tout en offrant un moyen de vérifier, tout au long du projet et à son terme, que les exigences définies sont respectées. Cette analyse constitue ainsi une base indispensable pour l'élaboration du cahier des charges.

La décomposition du système et l'identification des fonctions

L'analyse fonctionnelle est une méthode d'ingénierie qui consiste à décomposer un système complexe en sous-ensembles plus simples afin d'identifier les fonctions qu'ils assurent et les interactions entre eux. Cette approche permet de mieux comprendre le fonctionnement global du système et le rôle de chacun de ses éléments.

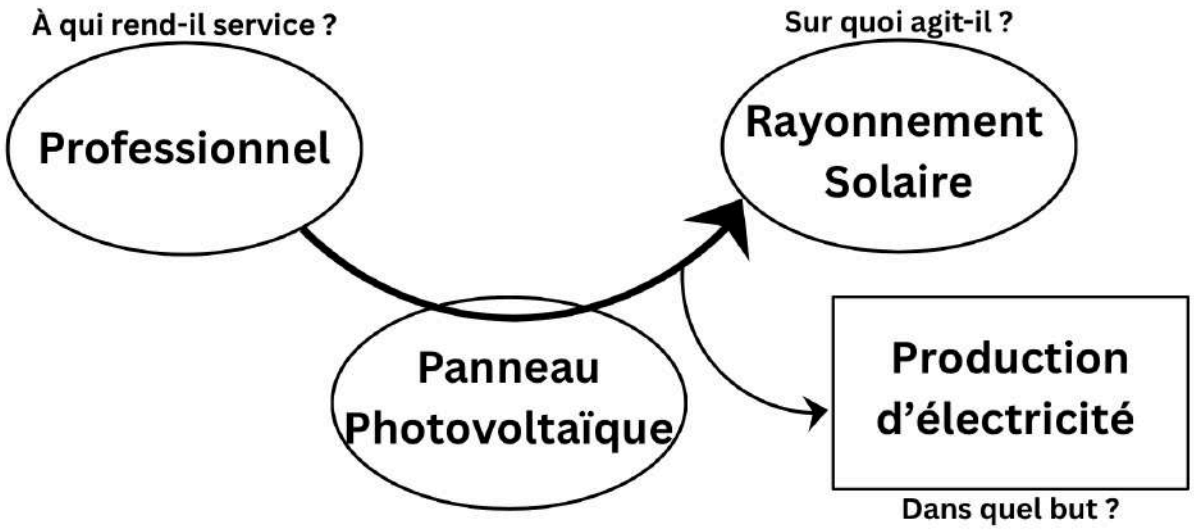
Elle vise également à traduire les besoins des utilisateurs en fonctions clairement définies que le produit doit remplir, indépendamment des solutions techniques envisagées. Cette démarche facilite la conception, l'amélioration et l'optimisation d'un système en le rendant plus lisible et maîtrisable.

Le principe fondamental de l'analyse fonctionnelle repose sur l'idée que chaque composant du système remplit une fonction précise et que ces fonctions sont interdépendantes. Leur identification permet d'évaluer leur contribution au fonctionnement global et d'améliorer les performances du système.

Pour synthétiser notre démarche d'analyse fonctionnelle, on a présenté nos résultats à l'aide de deux outils graphiques bien connus: une bête à corne mettant en lumière les acteurs touchés par le système, la matière d'œuvre du système et son rôle. Enfin, on a également utilisé un diagramme pieuvre ou diagramme des interacteurs au sein duquel figurent les principales fonctions que l'on a jugé indispensable afin que le système réponde parfaitement au besoin client.

Bête à corne

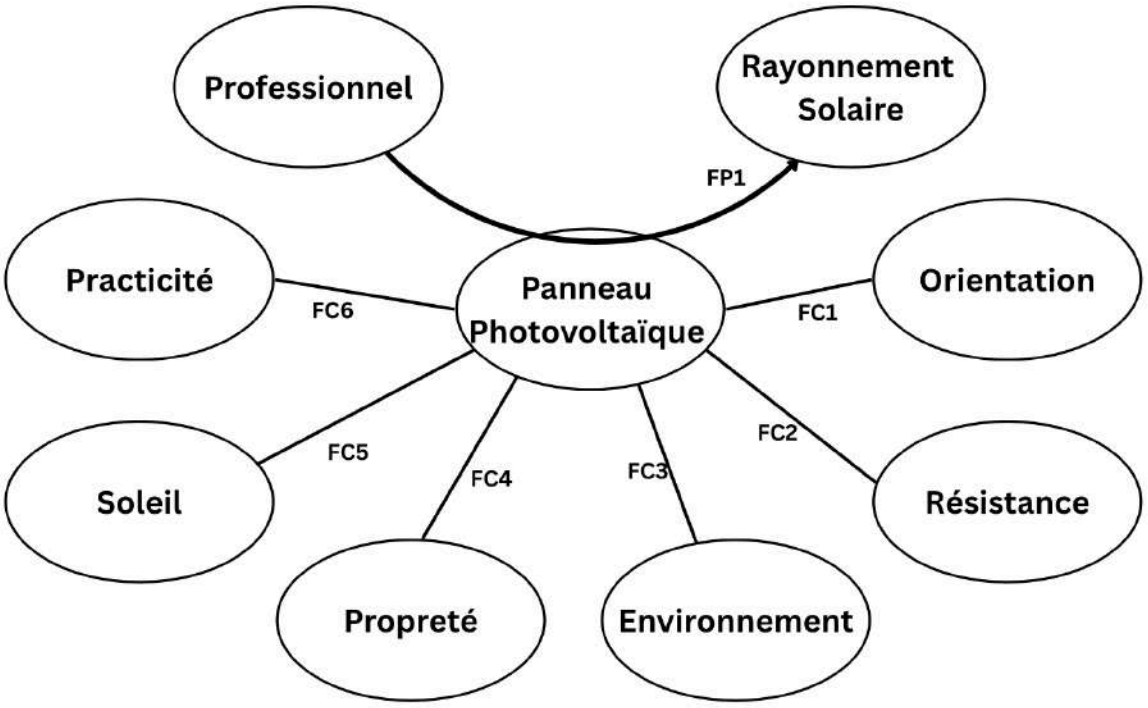
Le panneau photovoltaïque est placé au centre du diagramme de la bête à cornes.



Il rend service à un professionnel, ici à une entreprise, en agissant sur le rayonnement solaire. Plus concrètement, son but est de capter les photons de la lumière du soleil pour les convertir en énergie électrique.

Cette énergie sera majoritairement consommée par l'entreprise elle-même mais elle pourra aussi être revendue puis injectée dans le réseau électrique afin de pouvoir en tirer un bénéfice.

Diagramme Pieuvre



Voici les fonctions contraintes que l'on a réussi à établir à l'issue de cette étape d'analyse fonctionnelle:

FP1 : Convertir l'énergie lumineuse en énergie électrique

FC1: Le système doit orienter automatiquement le panneau vers le soleil afin qu'il reçoive un maximum de lumière

FC2 : Le panneau doit résister aux vents violents sans se déformer ni se casser en se mettant en position de sécurité (à plat)

FC3 : Le système doit se protéger contre les intempéries en limitant les dégâts qu'elles pourraient occasionner au panneau et à ses composants

FC4 : Le panneau doit rester propre car trop de saleté pourrait obstruer ses cellules et donc réduire drastiquement l'efficacité de la production

FC5 : Le système doit mesurer l'ensoleillement afin d'optimiser le rendement de la production électrique

FC6 : Le système doit être pratique d'utilisation en centralisant, puis en affichant toutes les informations recensées par les capteurs

Enfin, pour achever notre démarche d'analyse du système, on va se pencher sur un autre aspect: l'aspect opérationnel, que l'on va détailler dans la partie suivante.

Analyse Opérationnelle

L'analyse opérationnelle se définit comme l'étude détaillée des processus requis pour mener à bien une mission ou une activité spécifique. Si l'analyse fonctionnelle définit les objectifs attendus (le "quoi"), l'approche opérationnelle se focalise sur la mise en œuvre concrète (le "comment"). Elle décortique ainsi la chaîne des ressources, des gestes et des procédures indispensables pour parvenir au résultat visé.

Pour valider la pertinence de ce POC, le dispositif est étudié en situation réelle : installé en toiture, il est soumis aux mêmes contraintes climatiques qu'une installation photovoltaïque standard.

L'objectif est de vérifier si le système répond aux exigences du cahier des charges en s'appuyant sur des indicateurs concrets. L'analyse opérationnelle repose ici sur la collecte de données environnementales précises, traitées par une série de capteurs stratégiques:

- Thermique : Suivi des variations saisonnières et nycthémérales pour évaluer l'impact de la chaleur sur le rendement.
- Hygrométrie: Identification des précipitations (pluie, neige) pouvant influencer la production ou nécessiter une mise en sécurité.
- Luminosité: Quantification du flux lumineux pour piloter l'orientation dynamique du panneau et maximiser l'apport solaire.
- Maintenance: Évaluation de l'encrassement de la surface pour optimiser les cycles de nettoyage et limiter les pertes énergétiques.

Cahier des charges

Pour récapituler l'ensemble des fonctions et performances que doit posséder notre dispositif de panneau solaire autonome, on les a présentés au sein d'un outil très classique et indispensable à toute démarche d'innovation en ingénierie: le cahier des charges fonctionnel. Celui-ci est d'une aide précieuse, car il indique non pas la marche à suivre pour innover, mais plutôt les conditions qui vont stipuler que l'innovation apportée au système est valide et respecte donc les attentes du client. Nous avons établi notre cahier des charges à partir de nos deux analyses du système: l'analyse fonctionnelle et l'analyse opérationnelle.

F0 : Flexibilité nulle

F1 : Flexibilité faible

F2 : Flexibilité moyenne

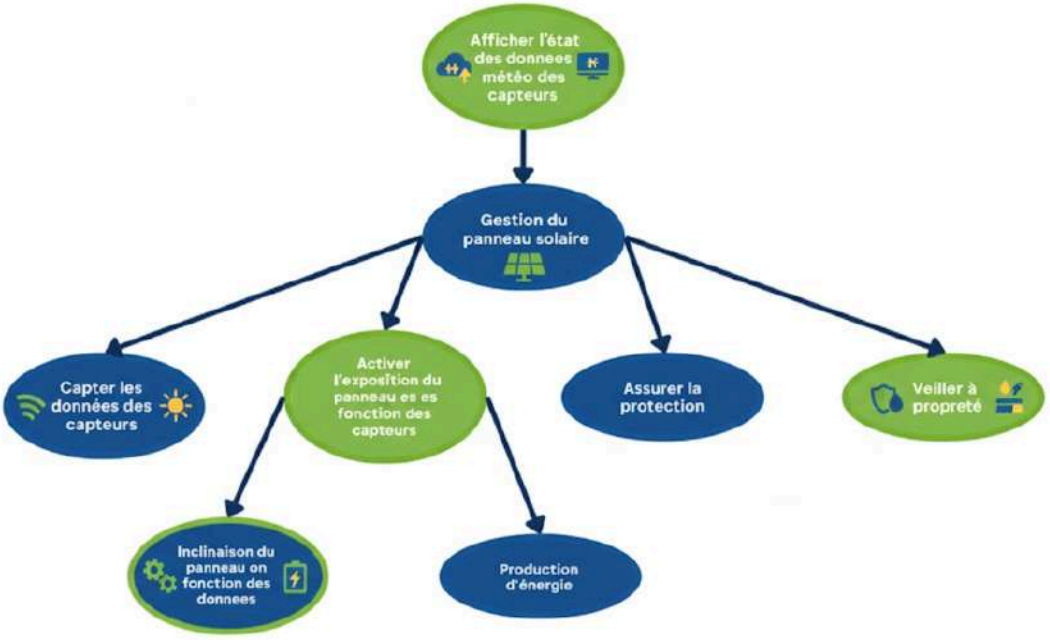
CAHIER DES CHARGES FONCTIONNEL (CDCF)			
FONCTION PRINCIPALE			
Fonction de service		**Justification**	
Convertir l'énergie lumineuse en énergie électrique		C'est la principale valeur ajoutée pour un panneau photovoltaïque: produire de l'électricité à partir de rayonnement solaire	
FONCTION SECONDAIRES			
Fonction de service	**Critère**	**Niveau**	**Flexibilité**
Maximiser le rayonnement solaire reçu par le panneau en suivant le soleil	Orientation précise	0° (horizontal) 90/-90° (vertical)	F0
	Vitesse de réponse rapide	≤ 30 secondes (pour un ajustement complet)	F1
Mesurer l'ensoleillement	Précision	+/- 5%	F1

CAHIER DES CHARGES FONCTIONNEL (CDCF)			
Résister aux contraintes mécaniques dues au vent	Seuil de déclenchement	Vent > 80km/h	F0
	Temps de mise en sécurité	<30 s	F0
	Mise à plat	à l'horizontal	F0
	Absence de déformation	Aucun dommage jusqu'à 120 km/h	F0
Résister aux intempéries	Résistance à la poussière, au sel	norme IP65	F0
	Tolérance aux variations climatiques	+/-50°	F0
Préserver la propreté de la surface active du panneau	Sensibilité à l'encrassement	Pression dépassant une valeur limite	F1
	Inclinaison si le panneau est obstrué	45°	F2
Détecter et agir face à la pluie	Seuil de déclenchement	Valeur de taux d'humidité	F1
	Inclinaison si pluie détectée	45°	F1
	Temps d'inclinaison	30 s max	F2
Permettre la transmission et la consultation des données au site web	Temps de traitement	1 minute max	F1
	Type de réseau	Wifi	F0
Assurer la transmission des données entre station météo et panneau solaire	Fiabilité des données mesurées (pluviométrie, ensoleillement, vent)	Données exploitables en temps réel	F0

CAHIER DES CHARGES FONCTIONNEL (CDCF)			
	Temps de traitement	1 minute max	F1
	Type de réseau	Wifi	F0

Diagramme des cas d'utilisation (DCU)

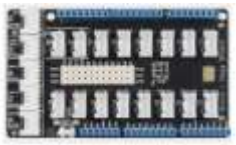
Le diagramme de commande d'utilisation modélise l'interaction entre les entrées de l'opérateur et le comportement interne du dispositif. En cartographiant les commandes disponibles et les changements d'états associés, il offre une vue d'ensemble du pilotage du système. Cet outil est indispensable pour valider la logique de contrôle et la réactivité du système face aux actions humaines.



Les acteurs du système sont : Utilisateurs, technicien de maintenance, réseau électrique, soleil, vent, pluie et pression

Description du système

Nous avons répertorié dans le tableau ci-dessous tous les éléments qui nous ont été nécessaires pour réaliser le projet.

Éléments	Quantité	Photo
Carte Arduino Mega. Elle simplifie le branchement des capteurs et actionneurs	1	
Shield Grove pour MEGA	1	
Carte NodeMCU avec shield Note. Elle permet de traiter les données d'un capteur et de les transmettre par wifi	1	
Servomoteur angulaire. Il permet le mouvement du panneau solaire en fonction de la luminosité et de la pluie et du vent et permet à la photorésistance de balayer la luminosité	2	
Capteur de température et humidité (DHT11 I2C) Grove. Il détermine la température et le taux d'humidité	1	

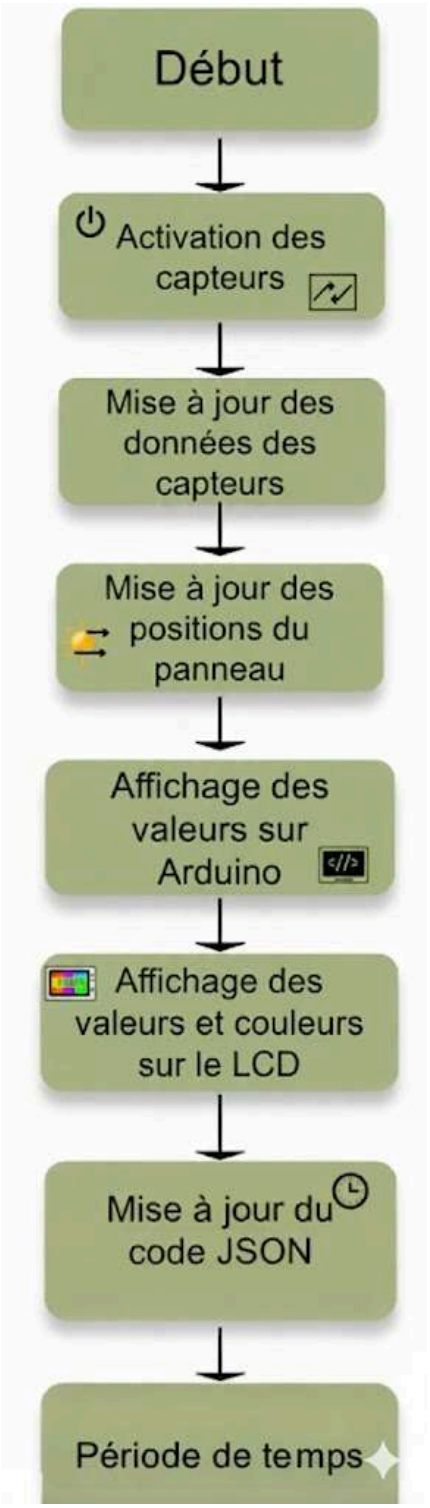
Capteur de lumière I2C Grove (photorésistance). Il mesure l'intensité lumineuse et en donne une valeur numérique	1	
Potentiomètre Grove. Il simule la vitesse du vent	1	
Ecran LCD I2C Grove Il affiche les informations renvoyées par les capteurs.	1	

Nous avons répertorié dans le tableau suivant, tous les branchements des capteurs présents ci dessus.

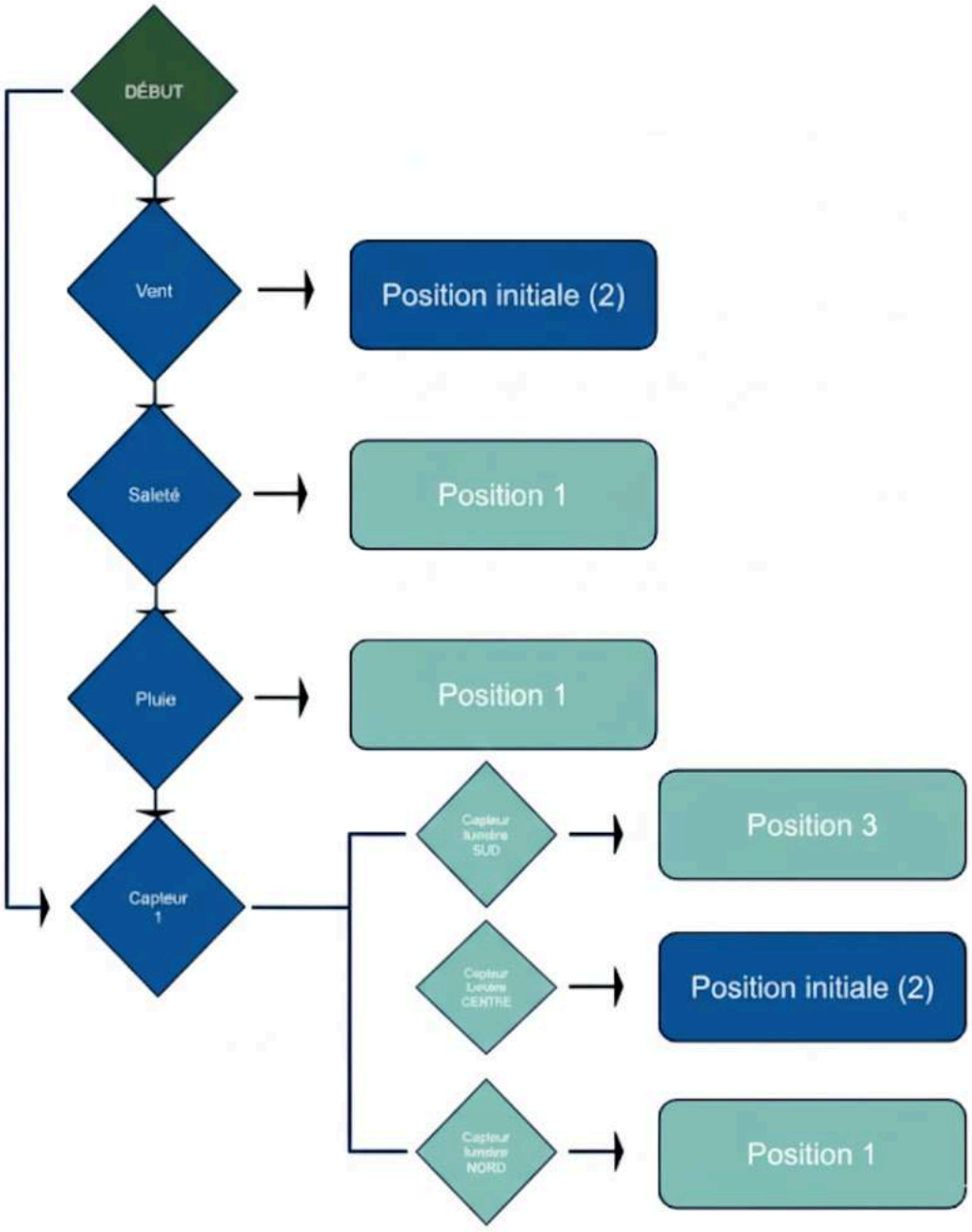
Éléments	**PIN branchés**
Servomoteur angulaire	D3
Capteur de température et humidité (DHT11 I2C) Grove	D4 D6
Capteur de lumière I2C Grove (photorésistance).	A1
Potentiomètre Grove	A2
Ecran LCD I2C Grove	SDA : SDA de la carte SCL : SCL de la carte

ALGORIGRAMMES

Programme Arduino:



Algorigramme de la position et de l'angle du moteur



Liste des solutions technologiques qu'on a réalisé

Analyse des Choix Technologiques du Système Solaire

Pour transformer les fonctions de service en réalité technique, nous avons sélectionné des solutions robustes adaptées à un environnement extérieur. Cette approche garantit l'interopérabilité entre les capteurs et le microcontrôleur.

Orientation automatique du panneau solaire :

Fonction : (**Orientation**) Piloter l'inclinaison de la structure pour maintenir un angle d'incidence optimal face au soleil.

Solution technologique : Un dispositif de suivi (tracker) utilisant des photorésistances pour la détection différentielle et un servomoteur pour la gestion de la partie mécanique.

Mesure de l'ensoleillement :

Fonction : (**Soleil)** Quantifier le flux lumineux reçu afin de déclencher les stratégies de positionnement.

Solution technologique : Intégration de capteurs de luminosité analogiques (LDR) ou numériques via bus I2C pour une lecture haute précision de l'éclairement énergétique.

Détection des conditions météorologiques :

Fonctions :(**Propreté**) Détecter la pluie : Anticiper les risques électriques et modifier l'assiette du panneau pour favoriser l'évacuation de l'eau.

(**Résistance**) Détecter les vents violents : Protéger l'intégrité structurelle en plaçant le panneau en position de sécurité (mise en drapeau).

Solutions technologiques : * Sonde d'humidité superficielle (écosystème Grove).

Centrale inertielle (accéléromètre) ou capteur magnétique à effet Hall pour évaluer la vitesse cinétique du vent.

Propreté du panneau solaire :

Fonction : (**Propreté**) Analyser le taux d'occultation de la surface (poussière, dépôts) pour prévenir la chute de rendement.

Solution technologique : Utilisation de capteurs de pression piézoélectriques sous la structure ou de cellules émettrices/réceptrices infrarouges pour mesurer l'opacité de la surface.

Affichage des données en temps réel :

Fonction : (**Praticité**) Restituer les paramètres du système sur une interface physique pour

l'opérateur local.

Solution technologique : Afficheur LCD alphanumérique piloté en I2C, relié à l'unité de traitement centrale.

Transmission et centralisation des données :

Fonction : (**Praticité**) Exporter les relevés vers une base de données distante ou un tableau de bord numérique.

Solution technologique : Microcontrôleur de type NodeMCU pour assurer la connectivité sans fil via les protocoles Wi-Fi ou Bluetooth.

Les solutions technologiques retenues

Dans le cadre de ce projet, la conception de notre panneau solaire intelligent repose sur une sélection rigoureuse d'équipements, dictée par les impératifs de notre cahier des charges. L'objectif principal était de créer un système capable de réagir de manière autonome à son environnement grâce à une architecture de capteurs diversifiée.

Instrumentation et traitement des données

Le cœur de la station météo assure le suivi des paramètres fondamentaux que sont la température et la pression atmosphérique. Pour ce faire, nous avons intégré des capteurs dédiés qui communiquent leurs relevés en continu au microcontrôleur. Ce dernier traite les informations avant de les acheminer vers l'écran d'affichage, permettant ainsi un suivi en temps réel des conditions ambiantes.

Stratégie d'optimisation de l'ensoleillement

Pour garantir une production énergétique maximale, nous avons développé une méthode d'orientation basée sur la comparaison lumineuse. Trois capteurs de luminosité (combinant technologie I2C, photorésistances et LED) sont répartis sur les faces de la station. Le microcontrôleur analyse les flux reçus : le panneau reste en position initiale ou s'incline de 45° vers la source la plus intense, à condition que l'intensité lumineuse soit suffisante. En cas de faible luminosité, le système privilégie la position centrale pour économiser l'énergie des moteurs.

Protocoles de sécurité et protection climatique

La robustesse du système est assurée par des capteurs de vent et d'humidité. La détection des vents violents repose sur un capteur à effet Hall dont les données sont comparées à un seuil critique préétabli. Si une rafale menace l'intégrité de la structure, le panneau se place automatiquement à plat pour offrir le moins de prise au vent possible, tandis qu'un message d'alerte s'affiche sur la console. Parallèlement, la détection de la pluie permet d'anticiper les risques de stagnation d'eau ou de courts-circuits.

Choix technique pour le contrôle de la propreté

L'efficacité des cellules photovoltaïques dépendant de leur clarté, nous avons exploré trois pistes pour mesurer l'encrassement : le différentiel de luminosité, l'utilisation de rayons infrarouges, ou la mesure de pression pondérale.

Après analyse, nous avons retenu la solution du capteur de pression installé sous le panneau. Cette méthode consiste à définir une limite de poids maximale ; si la pression mesurée excède ce seuil, le système conclut à une accumulation excessive de poussière ou de débris. Ce choix a été privilégié pour sa fiabilité et sa faisabilité technique dans les délais impartis pour notre projet.

Réalisation de la maquette

Pour réaliser la maquette, nous avons fait le choix de représenter un bâtiment quelconque, avec un toit plat. On a placé le panneau solaire en hauteur pour pas qu'il ne bloque en raison de sa rotation. Pour placer la photorésistance il faut qu'elle soit exactement le même axe de rotation que le panneau solaire, sinon le résultat serait incorrect. De plus, le panneau solaire ne doit pas lui faire d'ombre. Nous avons également mis une batterie externe pour alimenter les deux cartes présentes. Les cartes sont à l'intérieur du bâtiment, ce qui est assez réaliste en soi.

Conclusion

Nous avons trouvé que ce projet sur les panneaux solaires a été une expérience enrichissante, tant sur le plan technique que pratique. La bonne compréhension du projet grâce aux slides et l'utilisation de codes prédéfinis pour chaque capteur ont facilité l'intégration des composants, tout en nous permettant de nous concentrer sur leur mise en œuvre et leur optimisation.

L'association de la carte NodeMCU et de la carte Arduino Mega a représenté un défi supplémentaire. La gestion de la communication entre ces deux cartes a nécessité une attention particulière et une coordination précise, ce qui nous a permis de renforcer nos compétences en résolution de problèmes et en adaptabilité face à des architectures techniques complexes. La réalisation de preuves de concept (POC) a été une étape clé qui nous a permis de suivre toutes les étapes de la conception technique, de la définition des besoins à la validation des solutions, en passant par les ajustements nécessaires au fur et à mesure du développement.

Par ailleurs, ce projet nous a permis de travailler en équipe, en répartissant les tâches selon les points forts et les compétences de chacun, et de nous familiariser avec différents langages de programmation. Nous avons notamment programmé les capteurs pour mesurer la luminosité, la température et l'humidité en C++ (sur Arduino), et utilisé le langage HTML, CSS et JAVASCRIPT pour la création d'une interface web de suivi des données. Enfin, ce système s'inscrit dans une démarche écologique et représente une solution énergétique durable. Il ouvre des perspectives d'évolution intéressantes pour l'avenir, tant sur le plan technique que pour son impact environnemental.

Retours personnels

Ce projet a été une véritable opportunité pour notre équipe d'élargir ses compétences et d'enrichir ses connaissances. Nous avons particulièrement apprécié la découverte et l'utilisation de différents capteurs, comme le capteur à effet Hall (ou potentiomètre) utilisé pour simuler la vitesse du vent. Ces composants ont apporté une dimension pratique au projet, illustrant concrètement leur utilité dans un système intégré. Cette expérience nous a permis de prendre conscience de l'importance d'une bonne organisation et d'une répartition efficace des tâches au sein de l'équipe.

Avec un groupe de six, il a été essentiel de s'écouter et de coordonner nos efforts pour mener le projet à son terme. La structure claire du projet, les étapes définies et l'utilisation d'outils comme le diagramme de Gantt nous ont aidés à planifier le travail et à respecter les délais. Les professeurs nous ont apporté un accompagnement précieux, en nous guidant lors des séances de TD et de TP et en nous orientant dans les choix techniques à effectuer. La réalisation d'une maquette avec différents matériaux a également permis de visualiser concrètement le fonctionnement du système et d'illustrer nos concepts.

En résumé, ce projet a été à la fois un défi technique et une expérience collective enrichissante. Il nous a permis d'apprendre à travailler en équipe, à partager nos connaissances et à progresser individuellement et collectivement. Nous sommes pleinement satisfaits du travail accompli et de l'expérience acquise.

Annexes

CODE Pour la Mega :

```
#include "rgb_lcd.h"
#include <DHT.h>
#include <Servo.h>
#include <Arduino_JSON.h>

// Définir les pins
#define DHT_PIN 2
#define WIND_POT_PIN A0    // Potentiomètre pour simulation du vent
#define LIGHT_SENSOR_PIN A2 // Photorésistance pour la lumière
#define SERVO_SCAN_PIN 4   // Servo qui scanne
#define SERVO_TRACK_PIN 6  // Servo qui suit la position optimale

// Configurer le capteur DHT
#define DHT_TYPE DHT11
DHT dht(DHT_PIN, DHT_TYPE);

// Configurer le LCD RGB
rgb_lcd lcd;

// Configurer les servos
Servo servoScan;   // Servo qui scanne (pin 9)
Servo servoTrack;  // Servo qui suit (pin 8)

// Tableau pour stocker les positions du servo
int positions[] = {10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160,170,180};
int nbPositions = 18;

// Variables pour la recherche de luminosité maximale
int maxLightValue = 0;      // Valeur maximale de luminosité
int maxLightPosition = 0;   // Position correspondant à la luminosité max
bool scanningComplete = false;
int indexPos = 0;

// Variables pour la commande du servo track
int servoAngle = 90;        // Angle du servo track
String commandeMoteur = "Initialisation";

// Variables globales météo
float currentTemp = 0.0;      // Température courante
float currentHumidity = 0.0;  // Humidité courante
int windSpeed = 0;            // Vitesse du vent (0-100 km/h)
```

```
int lightValue = 0;          // Valeur de luminosité (0-1023)
int lightPercent = 0;        // Luminosité en pourcentage

// Variables pour calibration lumière
int lightLow = 1023;
int lightHigh = 0;

// Écran actuel (pour alterner l'affichage)
int currentScreen = 0;
unsigned long lastScreenChange = 0;
const unsigned long screenInterval = 3000; // Changer d'écran toutes les 3 secondes

// Variables pour le timing du scan servo
unsigned long lastScan = 0;
const unsigned long scanInterval = 20000; // Scanner toutes les 5 minutes (300000 ms)

// Variables pour la commande du servo track
unsigned long lastServoCommand = 0;
const unsigned long servoCommandInterval = 1000; // Mettre à jour la commande servo
toutes les secondes

//========== OBJETS ET VARABLES GLOBAUX
  JSONVar readings;

//========== REGLAGES
  uint32_t timerDelay     = 1000;       // Fréquence d'envoi vers le NODE
  uint32_t speedSerial    = 115200;     // Vitesse moniteur série
  uint32_t speedSerial1   = 115200;     // Vitesse envoi des données vers le NODE

//======== FACILITATEURS
  // MACRO : Limiteur de fréquence
    #define LIMIT_FREQ_BEGIN(varName, duration) static uint32_t varName = 0;
if(millis()>varName+duration){
    #define LIMIT_FREQ_END(varName) varName = millis();}

  // MACRO : Limiteur de répétition
    #define LIMIT_REP_BEGIN(varName, nbrOfRep) static uint8_t varName = 0;
if(varName<nbrOfRep){
    #define LIMIT_REP_END(varName) varName++;}

  // MACRO : Setup virtuel
    #define SETUP_VIR_BEGIN(varName) static uint8_t varName = 0; if(varName<1){
    #define SETUP_VIR_END(varName) varName++;}


//========== PROTOTYPES DE FONCTIONS
String getSensorReadings();
```

```
void setup() {

  //Pour le JSON

  Serial.begin(speedSerial);  while(!Serial);
  Serial1.begin(speedSerial1);while(!Serial1);
  Serial.println("Serials:ok");

  // Initialiser le port série
  Serial.begin(115200);

  // Initialiser le capteur DHT
  dht.begin();

  // Initialiser les servos
  servoScan.attach(SERVO_SCAN_PIN);   // Servo qui scanne sur pin 9
  servoTrack.attach(SERVO_TRACK_PIN); // Servo qui suit sur pin 8
  servoScan.write(90);   // Position initiale servo scan
  servoTrack.write(90);  // Position initiale servo track

  // Initialiser le LCD RGB
  lcd.begin(16, 2);
  lcd.setRGB(0, 255, 255);  // Couleur cyan au démarrage
  lcd.clear();

  // Afficher le message initial
  lcd.setCursor(0, 0);
  lcd.print("Station Meteo");
  lcd.setCursor(0, 1);
  lcd.print("Calibration...");

  // Calibration du capteur de lumière pendant 5 secondes
  unsigned long startTime = millis();
  while (millis() - startTime < 5000) {
    lightValue = analogRead(LIGHT_SENSOR_PIN);

    if (lightValue > lightHigh) {
      lightHigh = lightValue;
    }
    if (lightValue < lightLow) {
      lightLow = lightValue;
    }

    delay(100);
  }

  lcd.clear();
  lcd.setCursor(0, 0);
```

```
  lcd.print("Calibration OK!");
  lcd.setRGB(0, 255, 0);  // Vert pour succès
  delay(1000);
  lcd.clear();

  Serial.println("=== STATION MÉTÉO + SERVOS ===");
  Serial.println("Capteurs: DHT11 + Vent + Lumière + 2 Servos");
  Serial.println("Servo Scan (pin 9) | Servo Track (pin 8)");
  Serial.print("Lumière Min: ");
  Serial.print(lightLow);
  Serial.print(" | Max: ");
  Serial.println(lightHigh);
  Serial.println("Scan toutes les 5 minutes");
  Serial.println("==============================");
}

void loop() {
  // 1. Vérifier si il faut faire un scan avec le servo
  if (millis() - lastScan > scanInterval || !scanningComplete) {
    scanForMaxLight();
    lastScan = millis();
    scanningComplete = true;
  }

  // 2. Lire la température et l'humidité
  readDHT();

  // 3. Lire la vitesse du vent (potentiomètre)
  readWindSpeed();

  // 4. Lire la luminosité
  readLight();

  // 5. Commander le servo track en fonction des capteurs (vent/humidité/lumière)
  if (millis() - lastServoCommand > servoCommandInterval) {
    commandServoTrack();
    lastServoCommand = millis();
  }

  // 6. Afficher sur LCD (alternance entre écrans)
  displayLCD();

  // 7. Afficher sur port série
  displaySerial();

  // 8. Envoi JSON
  LIMIT_FREQ_BEGIN(sendingNodeData, timerDelay)
    Serial.println(getSensorReadings());
```

```
    Serial1.println(getSensorReadings());
  LIMIT_FREQ_END(sendingNodeData)

  delay(500); // Attendre 0.5 seconde
}

void scanForMaxLight() {
  Serial.println(">>> DÉBUT DU SCAN DE LUMINOSITÉ <<<");

  // Changer la couleur LCD pendant le scan
  lcd.setRGB(255, 255, 0);  // Jaune pendant le scan

  // Réinitialisation des valeurs max
  maxLightValue = 0;
  maxLightPosition = 0;

  // Balayage : parcourt toutes les positions du tableau avec le servo de scan
  for (indexPos = 0; indexPos < nbPositions; indexPos++) {
    servoScan.write(positions[indexPos]);
    delay(300);  // attente pour que le servo se stabilise

    lightValue = analogRead(LIGHT_SENSOR_PIN);  // Lecture de la photorésistance

    Serial.print("Position: ");
    Serial.print(positions[indexPos]);
    Serial.print("° | Luminosité: ");
    Serial.println(lightValue);

    // Vérifier si c'est la luminosité max
    if (lightValue > maxLightValue) {
      maxLightValue = lightValue;
      maxLightPosition = positions[indexPos];
    }

    delay(200);  // attente supplémentaire à chaque position
  }

  // Affichage de la position avec luminosité maximale
  Serial.println("=========================================");
  Serial.print("POSITION AVEC LUMINOSITÉ MAX: ");
  Serial.print(maxLightPosition);
  Serial.print("° | Valeur: ");
  Serial.println(maxLightValue);
  Serial.println("=========================================");

  // Déplacement des deux servos vers la position optimale
  Serial.println("Déplacement des servos vers la position optimale...");
  servoScan.write(maxLightPosition);   // Servo scan prend la position
```

```
  delay(500);

  // Note: Le servo track (pin 8) est géré par commandServoTrack()
  // qui prend en compte vent/humidité en priorité

  Serial.print("Servo Scan (pin 9): ");
  Serial.print(maxLightPosition);
  Serial.println("°");

  delay(1000);
}

void commandServoTrack() {
  // Priorité: vent > humidité > lumière
  if (windSpeed > 50) {              // Vent fort
    commandeMoteur = "Plat (vent)";
    servoAngle = 90; // 90 pour horizontal (protection contre le vent)
  } else if (currentHumidity > 90) {  // Humidité élevée
    commandeMoteur = "Incline (humidite)";
    servoAngle = 45; // Inclinaison pour évacuation/nettoyage
  } else {                           // Conditions normales: suivre la lumière
    commandeMoteur = "Suivi lumiere";
    servoAngle = maxLightPosition; // Suivre la position optimale trouvée par le scan
  }

  // Appliquer la commande au servo track
  servoTrack.write(servoAngle);
}

void readDHT() {
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();

  if (!isnan(temp)) {
    currentTemp = temp;
  } else {
    Serial.println("Erreur lecture température DHT11!");
  }

  if (!isnan(hum)) {
    currentHumidity = hum;
  } else {
    Serial.println("Erreur lecture humidité DHT11!");
  }
}

void readWindSpeed() {
  // Lire le potentiomètre et convertir en vitesse du vent (0-100 km/h)
```

```
  int potValue = analogRead(WIND_POT_PIN);
  windSpeed = map(potValue, 0, 1023, 0, 100);
}

void readLight() {
  // Lire la photorésistance
  lightValue = analogRead(LIGHT_SENSOR_PIN);

  // Calculer le pourcentage
  lightPercent = map(lightValue, lightLow, lightHigh, 0, 100);
  lightPercent = constrain(lightPercent, 0, 100);
}

void displayLCD() {
  // Alterner entre quatre écrans toutes les 3 secondes
  if (millis() - lastScreenChange > screenInterval) {
    currentScreen = (currentScreen + 1) % 4;
    lastScreenChange = millis();
  }

  lcd.clear();

  if (currentScreen == 0) {
    // ÉCRAN 1 : Température, Humidité et Vent
    lcd.setRGB(0, 255, 255);  // Cyan pour météo générale
    lcd.setCursor(0, 0);
    lcd.print("T:");
    lcd.print(currentTemp, 1);
    lcd.print("C");

    lcd.setCursor(9, 0);
    lcd.print("H:");
    lcd.print(currentHumidity, 0);
    lcd.print("%");

    lcd.setCursor(0, 1);
    lcd.print("Vent:");
    lcd.print(windSpeed);
    lcd.print(" km/h");

  } else if (currentScreen == 1) {
    // ÉCRAN 2 : Luminosité avec barre
    lcd.setRGB(255, 255, 0);  // Jaune pour luminosité
    lcd.setCursor(0, 0);
    lcd.print("Luminosite:");
    lcd.print(lightPercent);
    lcd.print("%");
```

```
    lcd.setCursor(0, 1);
    int bars = map(lightPercent, 0, 100, 0, 16);
    for (int i = 0; i < bars; i++) {
      lcd.write(0xFF); // Caractère bloc plein
    }

  } else if (currentScreen == 2) {
    // ÉCRAN 3 : Position optimale du servo scan
    lcd.setRGB(255, 165, 0);  // Orange pour servo scan
    lcd.setCursor(0, 0);
    lcd.print("Scan Pos:");
    lcd.print(maxLightPosition);
    lcd.print("deg");

    lcd.setCursor(0, 1);
    lcd.print("Lum max:");
    lcd.print(maxLightValue);

  } else {
    // ÉCRAN 4 : Commande du servo track avec couleur selon mode
    lcd.setCursor(0, 0);
    lcd.print("Track:");
    lcd.print(servoAngle);
    lcd.print("deg");

    lcd.setCursor(0, 1);
    if (commandeMoteur == "Plat (vent)") {
      lcd.setRGB(255, 0, 0);  // Rouge pour mode vent
      lcd.print("Mode: Vent");
    } else if (commandeMoteur == "Incline (humidite)") {
      lcd.setRGB(0, 0, 255);  // Bleu pour mode humidité
      lcd.print("Mode: Humidite");
    } else {
      lcd.setRGB(0, 255, 0);  // Vert pour mode lumière
      lcd.print("Mode: Lumiere");
    }
  }
}

void displaySerial() {
  static unsigned long lastSerialUpdate = 0;

  if (millis() - lastSerialUpdate > 2000) { // Toutes les 2 secondes
    Serial.println("=== DONNÉES MÉTÉO ===");

    Serial.print("Température: ");
    Serial.print(currentTemp, 1);
    Serial.println(" °C");
```

```
    Serial.print("Humidité: ");
    Serial.print(currentHumidity, 1);
    Serial.println(" %");

    Serial.print("Vitesse du vent: ");
    Serial.print(windSpeed);
    Serial.println(" km/h");

    Serial.print("Luminosité: ");
    Serial.print(lightValue);
    Serial.print(" (");
    Serial.print(lightPercent);
    Serial.println("%)");

    Serial.print("Position Servo Scan (pin 9): ");
    Serial.print(maxLightPosition);
    Serial.println("°");

    Serial.print("Position Servo Track (pin 8): ");
    Serial.print(servoAngle);
    Serial.print("° - ");
    Serial.println(commandeMoteur);

    Serial.println("========================");

    lastSerialUpdate = millis();
  }
}

String getSensorReadings() {
  readings["DATA-CARD1-1"] = String(currentTemp);
  readings["DATA-CARD1-2"] = String(currentHumidity);
  readings["DATA-CARD1-3"] = String(windSpeed);
  readings["DATA-CARD1-4"] = String(lightValue);
  readings["DATA-CARD1-5"] = String(lightPercent);
  readings["DATA-CARD1-6"] = String(servoAngle);

  String jsonString = JSON.stringify(readings);
  return jsonString;
}
```

CODE Pour la ESP8266 :

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <ESPAsyncTCP.h>
```

```
#include <ESPAsyncWebServer.h>
#include "LittleFS.h"
#include <Arduino_JSON.h>

//========== TODO PROF
//OpenData
//QR-code

//========== QUOI RÉGLER ?
//0-Veillez à avoir un point d'accès avec la bande de fréquences 2.4Ghz active (l'encryption PSK n'est pas compatible)
//1-Dans REGLAGE_01 : Remplacez les SSID et PASS par ceux disponibles (vous pouvez laisser des lignes vides)
//2-Dans REGLAGE_02 : Ouvrez le moniteur série et ajustez la vitesse à la valeur indiquée (115200 bauds ici)
//3-Dans REGLAGE_03 : Décommenter la ligne si vous souhaitez héberger les pages dans le Node
//4-Dans REGLAGE_04 : Décommenter la ligne et donnez un nom à votre Node si vous pouvez utiliser mDns


//========== REGLAGES_00
//===== REGLAGES_00 : Point d'accès
// Il doit avoir la bande de fréquence 2,4Ghz active
// Il ne doit pas utiliser le PSK !



//===== REGLAGES_01 : Modifiez les SSID et PASS ci-dessous en fonction des points d'accès disponibles
  //---- Si vous n'avez qu'un point d'accès, mettre 3 fois le même SVP

  //---- Point d'accès disponible 1
  #define AP1_SSID "PanneauSolaire"
  #define AP1_PASS "88888888"

  //---- Point d'accès disponible 2
  #define AP2_SSID ""
  #define AP2_PASS ""

  //---- Point d'accès disponible 3
  #define AP3_SSID ""
  #define AP3_PASS ""



//===== REGLAGES_02 : Vitesse moniteur série
uint32_t speedSerial    = 115200;
```

```
//===== REGLAGES_03 : littleFs (permet d'héberger les fichiers)
#define ENABLE_LITTLE_FS


//===== REGLAGES_04 : mDNS (permet de pouvoir charger la page par http://nom.local) - Pas compatible avec tout !
//#define ENABLE_MDNS
//#define NAME_MDNS "esp" // Le nom auquel doit répondre le CARD2 (ici, c'est http://esp.local)


//===== Réglage de la fréquence de rafraîchissement (pas trop vite SVP)
unsigned long timerDelay = 2000; // Fréquence de rafraîchissement imposée par le Node



//========== OBJETS ET VARIABLES
//----- Création d'un objet server sur le port 80 pour répondre à la requête http://192.168.x.x
AsyncWebServer server(80);

//----- Création d'un objet Websocket
AsyncWebSocket ws("/ws");

//----- Création d'un objet WifiMulti pour gérer plusieurs point d'accès
ESP8266WiFiMulti wifiMulti;


//========== BUFFER et autres
//----- Chaînes JSON
String jsonStringCard1    = "";  // Contiendra la chaîne JSON reçue de la CARD1
String jsonStringAll      = "";  // Contiendra la chaîne JSON complète envoyée à la page web

//----- Variables devant contenir les valeurs de capteurs (variables à mettre à jour dans le programme)
 /*
 String   data_card2_1;          // Exemple
 int      data_card2_2;          // Exemple : Entier 16bits
 float    data_card2_3;          // Exemple : Nbr à virgule
 int32_t  data_card2_4;          // Exemple : Entier 32bits
 uint32_t data_card2_5;          // Exemple : Entier 32bits non signé
 String   data_card2_6;          // Exemple
 */

uint32_t data_card2_1 = 0;                              // Exemple
uint32_t data_card2_2 = 0;                              // Exemple
```

```
uint32_t data_card2_3 = 0;                    // Exemple
uint32_t data_card2_4 = 0;                    // Exemple
uint32_t data_card2_5 = 0;                    // Exemple
uint32_t data_card2_6 = 0;                    // Exemple


//======== FACILITATEURS (ne pas changer SVP)
// MACRO : Limiteur de fréquence
#define LIMIT_FREQ_BEGIN(varName, duration) static uint32_t varName = 0;
if(millis()>varName+duration)
#define LIMIT_FREQ_END(varName) varName = millis()

// MACRO : Limiteur de répétition
#define LIMIT_REP_BEGIN(varName, nbrOfRep) static uint8_t varName = 0;
if(varName<nbrOfRep)
#define LIMIT_REP_END(varName) varName++

// MACRO : Setup virtuel
#define SETUP_VIR_BEGIN(varName) static uint8_t varName = 0; if(varName<1)
#define SETUP_VIR_END(varName) varName++


//========== PROTOTYPES DE FONCTIONS (pour que VSCode soit contant)
String  getSensorReadings();
String  getSensorReadingsCard1();
void    initWifiMulti();
void    notifyClients(String sensorReadings);
void    handleWebSocketMessage(void *arg, uint8_t *data, size_t len);
void    onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType
type, void *arg, uint8_t *data, size_t len);
void    initWebSocket();
void    initFS();
void    manageMdns();
String  checkAndReadsCard1Data();


//======================================== SETUP()
=====================================
void setup() {
  //---------- Initialisation des liaisons séries
  Serial.begin(speedSerial); while (!Serial); // Utiliser pour débogage & réception données de
la CARD1
  Serial.println(); Serial.println("Serial:ok");

  //---------- Initialisation du Wifi (ancienne méthode)
  //initWiFi();

  //---------- Initialisation du wifi-multi (nouvelle méthode)
```

```
  wifiMulti.addAP(AP1_SSID, AP1_PASS);
  wifiMulti.addAP(AP2_SSID, AP2_PASS);
  wifiMulti.addAP(AP3_SSID, AP3_PASS);
  initWifiMulti();

  //---------- Initialisation du websocket
  initWebSocket();

  //---------- Initialisation du SPIFF (appelé maintenant Little FS sur les ESP8266)
  #if defined ENABLE_LITTLE_FS
  initFS();

  //---- Initialisation du serveur
  server.on("/", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send(LittleFS, "/index.html", "text/html");
  });
  server.serveStatic("/", LittleFS, "/");
  server.begin();
  #endif
}



//======================================== LOOP()
======================================
void loop() {

  //========== OPTION : MDNS : ne fonctionne pas toujours - (il faut un navigateur
compatible hélas)
  #if defined ENABLE_MDNS
    manageMdns();
  #endif

  //========== Récupération des données de la CARD1 (capteurs distants)
  jsonStringCard1 = checkAndReadsCard1Data();

  //========== Récupération des données des capteurs de l'ESP8266 (mise à jour des
valeurs de capteurs locaux)
  /* Exemple
    data_card2_1  = "Hello CARD2!";                  // Exemple chaîne simple
    data_card2_2  = random(0,1000);                  // Exemple nbr 2 octets
    data_card2_3  = 0.12;                          // Exemple nbr float
    data_card2_4  = 123456789;                       // Exemple nbr 4 octets
    data_card2_5  = random(0,10000);                 // Exemple nbr 4 octets non signés (au
hasard)
    data_card2_6  = "<span style='color:red;'>RED<span>";  // Exemple String avec injection
html
    //*/
```

```
  //========== Envoie les données complètes à la page web
  LIMIT_FREQ_BEGIN(sendingWebData, timerDelay) {
    String sensorReadings = getSensorReadings();
    Serial.print("CARD2>");
    Serial.println(sensorReadings);
    notifyClients(sensorReadings);
    LIMIT_FREQ_END(sendingWebData);
  }

  //========== Gestion des clients
  ws.cleanupClients();

}



////////////////////////////////////////////////////////////////////////////////
//                          IMPLÉMENTATIONS DE FONCTIONS                      //
////////////////////////////////////////////////////////////////////////////////

String getSensorReadings() {
  // Utilité : récupère les données de la carte2 et de la carte1 et retourne une chaîne JSON

  //========== Récupération des données locales à la carte 2
  // Les noms des étiquettes DATA-X-X doivent être les mêmes que dans le code html !
  // On peut injecter de l'html de manière dynamique ici(à utiliser avec parcimoni car envoyé
régulièrement !)
  JSONVar jsonObjectAll;

  //========== Ajout des données de la CARD1
  // Les données sont dans jsonStringCARD1 qui est une chaîne au format JSON
  JSONVar jsonObjectCard1 = JSON.parse(jsonStringCard1);
  jsonObjectAll["DATA-CARD1-DISTANT-1"]   =   String(jsonObjectCard1["DATA-CARD1-1"]);
  jsonObjectAll["DATA-CARD1-DISTANT-2"]   =   String(jsonObjectCard1["DATA-CARD1-2"]);
  jsonObjectAll["DATA-CARD1-DISTANT-3"]   =   String(jsonObjectCard1["DATA-CARD1-3"]);
  jsonObjectAll["DATA-CARD1-DISTANT-4"]   =   String(jsonObjectCard1["DATA-CARD1-4"]);
  jsonObjectAll["DATA-CARD1-DISTANT-5"]   =   String(jsonObjectCard1["DATA-CARD1-5"]);
  jsonObjectAll["DATA-CARD1-DISTANT-6"]   =   String(jsonObjectCard1["DATA-CARD1-6"]);

  jsonStringAll = JSON.stringify(jsonObjectAll);
  return jsonStringAll;
}

void initWifiMulti() {
  Serial.println("Tentative de connexion à un point d'accès disponible... ");
```

```
  //while (wifiMulti.run() != WL_CONNECTED) {
  //}

  tryWifi:
  if (wifiMulti.run() != WL_CONNECTED) {
    delay(500);
    Serial.print('.');
    goto tryWifi;
  }
  Serial.println('\n');
  Serial.print("Connecté à:\t\t\t");
  Serial.println(WiFi.SSID());
  Serial.print("Adresse IP:\t\t\t");
  Serial.println(WiFi.localIP());
  Serial.print("Force du signal(dBm):\t");
  Serial.println(WiFi.RSSI());
  Serial.print("Adresse MAC du Node:\t");
  Serial.println(WiFi.macAddress());
}

void notifyClients(String sensorReadings) {
  ws.textAll(sensorReadings);
}

void handleWebSocketMessage(void *arg, uint8_t *data, size_t len) {
  AwsFrameInfo *info = (AwsFrameInfo*)arg;
  if (info->final && info->index == 0 && info->len == len && info->opcode == WS_TEXT) {
    //data[len] = 0;
    //String message = (char*)data;
    // Check if the message is "getReadings"
    //if (strcmp((char*)data, "getReadings") == 0) {
    //if it is, send current sensor readings
    String sensorReadings = getSensorReadings();
    //Serial.print(sensorReadings);
    notifyClients(sensorReadings);
    //}
  }
}

void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType type,
void *arg, uint8_t *data, size_t len) {
  switch (type) {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u connected from %s\n", client->id(),
client->remoteIP().toString().c_str());
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u disconnected\n", client->id());
```

```
      break;
    case WS_EVT_DATA:
      handleWebSocketMessage(arg, data, len);
      break;
    case WS_EVT_PONG:
    case WS_EVT_ERROR:
      break;
  }
}

void initWebSocket() {
  ws.onEvent(onEvent);
  server.addHandler(&ws);
}

String checkAndReadsCard1Data() {
  static String result = ""; // Evite de créer une variable globale en dehors de cette fonction
  if (Serial.available()) {
    result = Serial.readStringUntil('\n'); // Lecture d'une chaîne JSON complète
    Serial.print("CARD1>");
    Serial.println(result);
    Serial.flush();
    // A ce stade, la chaîne JSON envoyée par la CARD1 est dans result
  }
  return result;
}

//============================== OPION : LITTLE-FS (gestion des fichiers web)
#if defined ENABLE_LITTLE_FS
void initFS() {
  if (!LittleFS.begin()) {
    Serial.println("LittleFS a rencontré un problème");
  }
  else {
    Serial.println("LittleFS est fonctionnel");
  }
}
#endif

//============================== OPTION : MDNS (uniquement utile si vous voulez
accéder à la page par http://nomMdns.local)
#if defined ENABLE_MDNS
#include <ESP8266mDNS.h>
void manageMdns() {
  const char* hostName  = NAME_MDNS;
  SETUP_VIR_BEGIN(mDns) {
    if (MDNS.begin(hostName)) {
      Serial.print("mDNS est actif. ");
```

```
    Serial.print("Votre CARD2 répondra à http://");
    Serial.print(hostName);
    Serial.println(".local (si navigateur compatible)");
    //MDNS.addService("http", "tcp", mdnsPort);
   } else {
    Serial.println("Erreur lors de l'activation de mDNS.");
   }
   SETUP_VIR_END(mDns);
  }
  MDNS.update();
}
#endif
```

Code HTML :

```
<!--Todo : https://www.chartjs.org/docs/latest/getting-started/ -->
<!DOCTYPE html>
<html>
    <head>
        <title>Panneau de controle</title>
        <meta name="viewport" content="width=device-width,
initial-scale=1">
        <link rel="icon" type="image/png" href="img/epf.jpeg">
        <link rel="stylesheet" type="text/css" href="style.css">
    </head>
    <body>
        <div class="topnav">
            <h1>INFORMATIONS DU PANNEAU SOLAIRE</h1>
            <br />
            <hr class="hr">
        </div>



        <div class="content">
            <div class="card-grid-1">
                <div class="card1">
                    <p class="card-title"> Temperature (en degres
Celsius) </p>
                    <p class="reading"><span
id="DATA-CARD1-DISTANT-1"></span></p>
                </div>
```

```
                <div class="card2">
                    <p class="card-title"> Humidite (en %)</p>
                    <p class="reading"><span
id="DATA-CARD1-DISTANT-2"></span></p>
                </div>

                <div class="card3">
                    <p class="card-title"> Vitesse du vent ( en m/s)
</p>
                    <p class="reading"><span
id="DATA-CARD1-DISTANT-3"></span></p>
                </div>

                <div class="card4">
                    <p class="card-title"> Luminosite (en Lux) </p>
                    <p class="reading"><span
id="DATA-CARD1-DISTANT-4"></span></p>
                </div>

                <div class="card5">
                    <p class="card-title"> Luminosite (en %) </p>
                    <p class="reading"><span
id="DATA-CARD1-DISTANT-5"></span></p>
                </div>

                <div class="card6">
                    <p class="card-title"> Angle du panneau (en degres)
</p>
                    <p class="reading"><span
id="DATA-CARD1-DISTANT-6"></span></p>
                </div>

            </div>
        </div>
        <script src="script-data-reception.js"></script>
    </body>
</html>
```

Code CSS :

```
/*
```

```
Sélecteur de couleur en ligne :
https://developer.mozilla.org/fr/docs/Web/CSS/CSS_colors/Color_picker_t
ool


*/
html {
    font-family: Arial, Helvetica, sans-serif;
    display: inline-block;
    text-align: center;
}
h1 {
    font-size: 1.6rem;
}
.topnav {
    overflow: hidden;
    background-color: #FFFFFF;
    padding-bottom: 0px;
}
.hr {
    margin-top: 20px;
    color:#FFF;
}

body {
    margin: 0;
    position: relative;
    min-height: 100vh;

    /* Image de fond */
    background-image: url('img/epf.jpeg');
    background-size: contain;       /*on s'assure d'avoir toute
l'image*/
    background-position: center;    /* centrée */
    background-repeat: no-repeat;   /* pas de répétition sinon c'est
horrible */
    background-attachment: fixed;   /* reste fixe lors du scroll */
    opacity: 0,8;
}

.content {
    padding-top:    30px;
```

```
    padding-left:   40px;
    padding-right:  40px;
}

.card-grid-1{
    max-width:      1600px;
    margin:         0 auto;
    display:        grid;
}

.card-grid-1 {
    grid-gap:       2rem;
    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
}



.card1, .card2, .card3, .card4, .card5, .card6{
    background-color: rgba(196, 224, 251, 0.6);
    box-shadow: 2px 2px 6px 1px rgba(230,230,230,.9);
    border-radius: 0px 20px 20px 20px;
    padding:        20px;
}


.card-title {
    font-size: 1.2rem;
    font-weight: bold;
    color: #0b3153
}
.reading {
    font-size: 1.2rem;
    color: #32889f;
}
```

Code JavaScript :

```
var gateway = `ws://${window.location.hostname}/ws`; // Si page web
dans le SOC
//var gateway = `ws://192.168.1.99/ws`; // Si page web sur ordi.
var websocket;

// Init web socket when the page loads
window.addEventListener('load', onload);

function onload(event) {
    initWebSocket();
}

function getReadings(){
    websocket.send("getReadings");
}

function initWebSocket() {
    console.log('Trying to open a WebSocket connection…');
    websocket = new WebSocket(gateway);
    websocket.onopen = onOpen;
    websocket.onclose = onClose;
    websocket.onmessage = onMessage;
}

// When websocket is established, call the getReadings() function
function onOpen(event) {
    console.log('Connection opened');
    getReadings();
}

function onClose(event) {
    console.log('Connection closed');
    setTimeout(initWebSocket, 2000);
}

// Function that receives the message from the ESP8266 with the
readings
// Les noms des ◆tiquettes JSON doivent ◆tre les m◆mes que les id du
fichier html
function onMessage(event) {
    console.log(event.data);
    var myObj = JSON.parse(event.data);
    var keys = Object.keys(myObj);
```

```
    for (var i = 0; i < keys.length; i++){
        var key = keys[i];
        document.getElementById(key).innerHTML = myObj[key];
    }
}
```