

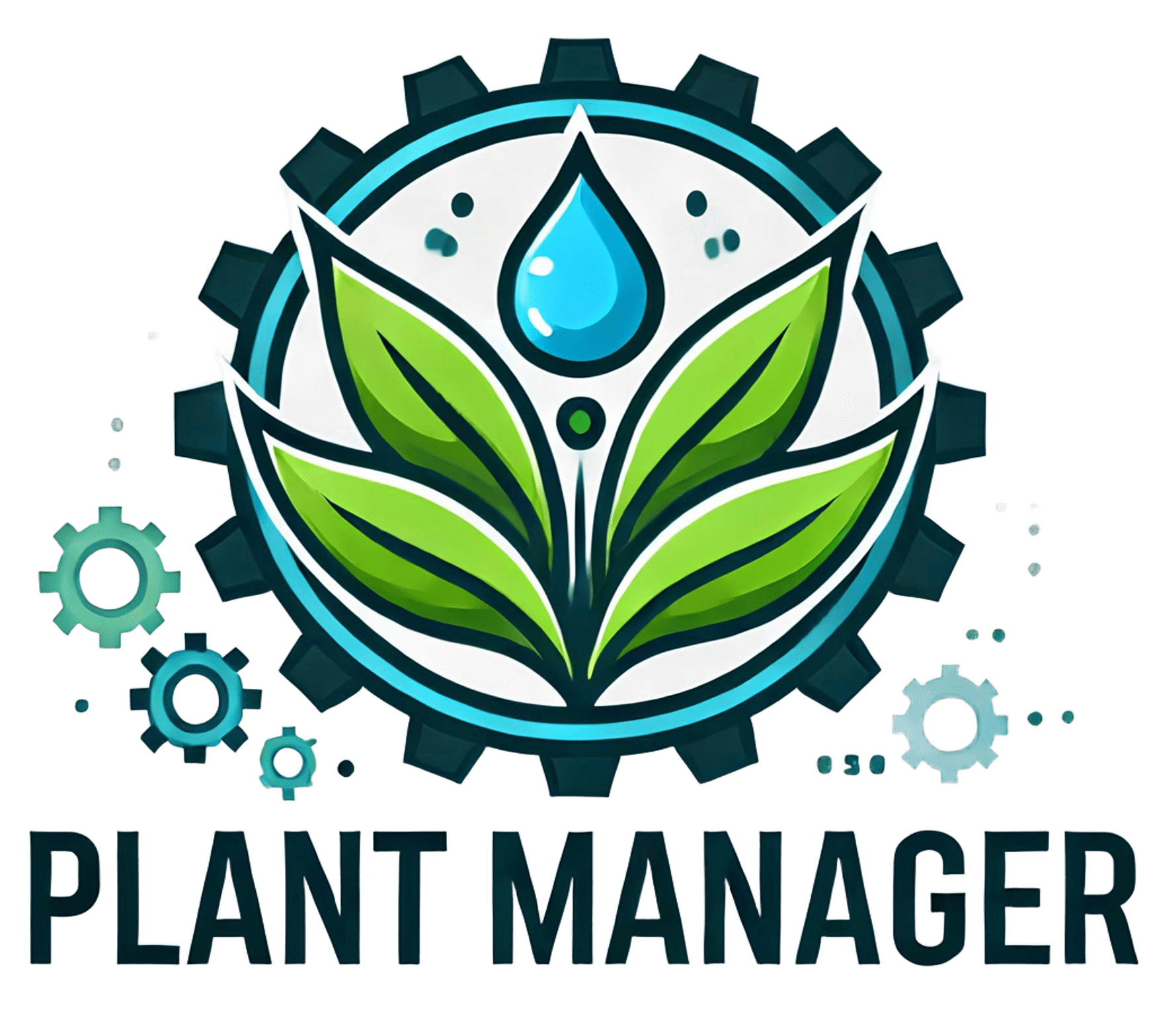
BUT3 - GEII - ESE

Thomas Gomes

IUT NICE CöTE d’AZUR

Plant Manager

RAPPORT DE TACHES

Une image contenant plante d’intérieur, pot de fleurs, spathiphyllum, plante

Le contenu généré par l’IA peut être incorrect.

# Introduction

Dans le cadre de la SAE de BUT3 ESE, le projet Plant Manager est une solution de gestion des plantes de manière automatisée, assurant un suivi en temps réel ainsi qu’une optimisation des conditions environnementales propices à leur croissance. Ce système repose sur un Raspberry Pi Zéro 2 W, et est équipé de divers de capteurs, permettant de mesurer des valeurs essentielles au développement des plantes, à savoir l’humidité du sol, la température, l’humidité de l’air, le niveau de luminosité et la concentration de CO₂.

Construit avec une architecture propre et efficace, Plant Manager permet un contrôle des besoins des plantes, en déclenchant automatiquement des actions comme l’arrosage via une pompe à eau. Le système est visualisé et piloté via une interface web permettant aux utilisateurs d’accéder aux valeurs des capteurs, d’ajuster les paramètres de l’automatisation, d’intervenir à distance en cas de nécessité ainsi que de voir l’évolution des mesures effectuer au cours du temps.

Table des matières

[I. Introduction 1](#_Toc191222487)

[II. Objectif du projet 4](#_Toc191222488)

[III. Architecture globale 5](#_Toc191222489)

[I. Matériel : 5](#_Toc191222490)

[II. Logiciel : 5](#_Toc191222491)

[IV. Taches 6](#_Toc191222492)

[1. Programmation capteur & actionneur 7](#_Toc191222493)

[Capteur de luminosité 7](#_Toc191222494)

[Capteur de température & humidité de l’air 7](#_Toc191222495)

[Capteur d’humidité du sol 8](#_Toc191222496)

[Actionneur : Buzzer 8](#_Toc191222497)

[2. Interface web ( Annexe1) 9](#_Toc191222498)

[Arborescences du projet 10](#_Toc191222499)

[Serveur 11](#_Toc191222500)

[Dépendance Node JS 11](#_Toc191222501)

[Fonctionnalité : 11](#_Toc191222502)

[Liste des API de Plant Manager 12](#_Toc191222503)

[Base de données 13](#_Toc191222504)

[Table : niveau\_eau 13](#_Toc191222505)

[Table : configurations 13](#_Toc191222506)

[Table : capteurs\_configuration 14](#_Toc191222507)

[Table : data\_capteur 14](#_Toc191222508)

[Arborescence de la base de données 15](#_Toc191222509)

[Interface client 16](#_Toc191222510)

[INDEX 16](#_Toc191222511)

[CONNEXION 17](#_Toc191222512)

[DASHBOARD 19](#_Toc191222513)

[CONFIGURATION 22](#_Toc191222514)

[DATA-EVOLUTION 24](#_Toc191222515)

[III. Intégration du système sur Raspberry 26](#_Toc191222516)

[26](#_Toc191222517)

[V. CONCLUSION 27](#_Toc191222518)

[VI. Annexes 28](#_Toc191222519)

# Objectif du projet

* Informations des plantes : Mesure en temps réel via des capteurs
* Surveillance des plantes : Visualisation des données via une page web
* Automatisation de l’arrosage : Activation en fonction de l’humidité du sol ou via page web
* Respect des conditions environnementales : Configuration spécifique au besoin de chaque plante
* Alertes et notifications : En cas de valeurs en dehors d’une plage spécifique : alertes par mail

# Architecture globale

Pour mener à bien PlantManager, le système est composé des éléments suivants :

## Matériel :

* + Raspberry Pi Zero 2 W
  + Capteurs :
    - **Humidité du sol** : Détection du besoin d’arrosage
    - **Température & humidité de l’air** : Surveillance du climat ambiant
    - **Luminosité**  : Surveillance de l’exposition lumineuse
    - **CO₂** : Surveillance de la concentration en CO2
    - **Flotteur** : Vérification du réservoir d’arrosage
    - **Bouton poussoir :** activer la pompe de manière manuelle
  + Actionneurs
    - **Pompe à eau** : Surveillance du niveau d’eau
    - **Ruban LED** : Indicateur visuel des conditions de la plante.
    - **Buzzer**: alarme sonore en cas de réservoir vide
    - **Divers leds :** affiche des états du système

## Logiciel :

* + **Interface Web :** 
    - Lancement du système de mesures
    - Affichage des valeurs de capteurs (tableau, jauges graphiques)
    - Configuration de plante pour l’automatisation
    - Système de session
    - Pilotage de la pompe
    - Notification par mail en cas d’alertes
    - Visualisation de l’évolution des données
  + **Script python** : Permet de lire les capteurs et de gérer les actionneurs ainsi que l’envoyer les données vers une base de données
  + **Base de données SQLite** : Stocke les valeurs des capteurs et des configurations
  + **Serveur Node.js** : Gère la communication entre la base de données et l’interface web.
  + **GitHub** : Utilisation d’un dépôt pour le développement

# Taches

Pour réaliser ce projet dans un temps dédié de 40H (10x4H) en projet de SAE,

Alexandre Pichot (binôme) et moi-même avons décider de se scinder le travail selon les différentes missions à réaliser.

Les missions réaliser pour ma part sur ce projet sont :

1. La programmation de capteurs
   * Capteur de luminosité
   * Capteur de température
   * Capteur d’humidité
   * Capteur d’humidité du sol
2. La programmation d’actionneur
   * Le buzzer
3. Le développement de toutes la partie logicielle
   * Script global python
   * Interface Web
   * Base de données
   * Serveur Node.js
   * Programmation python permettant de lire l’ensemble des valeurs des capteurs.
4. Intégration du système sur Raspberry

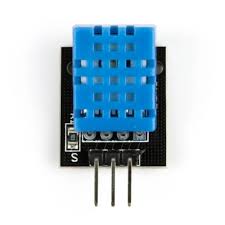
## Programmation capteur & actionneur

### Capteur de lumière VEML7700 ADA4162 - GotronicCapteur de luminosité

Afin de développer notre gamme de capteur permettant l’exploitation leurs données, j’ai programmé un script python permettant de lire les valeurs d’un capteur de luminosité (Réf : VEML7700). Celui-ci pour de mesurer l’intensité lumineuse ambiante et ajuster les conditions de croissance des plantes de manière automatisée.

Coder en python et connecté au système en I2C, il permet d’envoyer ces valeurs (en lux) en temps réel à la base de données

### Capteur de température & humidité de l’air

Par la suite, pour un système efficace, la température et l’humidité de l’air sont des données importantes pour une plante, c’est pour cela que j’ai programmer un capteur de température et d’humidité (Réf : DHT11). Le capteur est connecté à un GPIO du Raspberry Pi Zero 2 W et permet de donner une plage de température entre 0°C à 50°C et une plage d’humidité : 20% à 90%. L’utilisation d’une bibliothèque Adafruit\_DHT est nécessaire pour lire les valeurs du capteur via Python.

Cependant dans des contraintes de réalisation de la carte et de temps ce capteur ne sera pas d’utiliser dans ce projet car son utilisation en gpio et non en I2C cause une gêne pour le reste du système.

Cette valeur est transmise à la base de données

### Capteur d’humidité du sol

De plus, une des données fondamentales du projet est l’humidité du sol. En effet, il permet par la lecture de sa valeur d’adapter l’arrosage automatique en fonction des besoins réels des plantes. Donner en tensions convertit en %, il est un chiffre clé pour le projet.

J’ai développé le script de ce capteur avec python en connectant entre le capteur et la raspberry, un ads115 qui est un convertisseur analogique-numérique (ADC) pour lire précisément la valeur du capteur d’humidité du sol.

Cette valeur est transmise à la base de données

### Actionneur : Buzzer



Enfin, dans le but d’avoir un moyen physique de prévenir l’utilisateur du système en cas de problème, l’emploie d’un buzzer comme moyen alerte sonore est fondamental pour le système. En effet, ce buzzer de type TOR (0 ou 1) permet de faire un bruit lorsque le réservoir d’eau est vide. Ceci permet donc de prévenir l’utilisateur autre que par le web que le système a besoin d’une action humaine pour continuer à fonctionner. Le script de ce buzzer à été fait en python est lit la valeur du flotteur (détection de niveau d’eau). Ainsi lorsque le niveau d’eau est à 0 alors le buzzer passe à 1 et fait du bruit

L’ensemble de ces capteurs et actionneurs ont été développer et tester indépendant pour des phases de test puis assembler sur un PCB et piloter et lu depuis un script général regroupant l’ensemble des capteurs et actionneur. Ce script global permet aussi la sauvegarde des données dans la base de données du système plant manager

## Interface web ( Annexe1)

L’interface web du projet PlantManager, est composé de fichier se trouvant à la racine pour le côté serveur et d’un dossier public où se trouve l’ensemble des fichiers pour le client.

PlantManager sur web est un système aux multiples fonctionnalités dont j’ai défini la structure suivante :

* Accueil sur une page d’index
* Espace utilisateur pour l’inscription et la connexion au compte PlantManager
* Visualisation & Pilotage sur le tableau de bord regroupant l’ensemble des informations nécessaire en temps réel du système
* Un onglet configuration permettant à l’utilisateur de paramétrer son compte ainsi qu’un environnement de configuration pour des valeurs idéales de gestion d’une plante avec aide en ligne
* Un onglet de visualisation des données au cours du temps

Logo de PlantManager

Une image contenant texte, Graphique, graphisme, affiche

Le contenu généré par l’IA peut être incorrect.

### Arborescences du projet

* PlantManager/
* .git/ > Dossier Git
* node\_modules/ > Dépendances Node.js installées
* public/ > Contient les fichiers vue par le client
  + css/ > Styles CSS
    - index.css
    - connexion.css
    - dashboard.css
    - configuration.css
    - data-evolution.css
  + html/ > Fichiers HTML pour l'interface
    - index.html
    - connexion.html
    - dashboard.html
    - configuration.html
    - data-evolution.html
  + javascript/ > Scripts JS pour gérer les interactions
    - index.js
    - connexion. js
    - dashboard. js
    - configuration. js
    - data-evolution. js
  + media/ > Images
* controle.json > Fichier JSON de configuration de contrôle
* script\_plantmanager.py > Script pour capteurs physiques
* package.json > Déclaration des dépendances Node.js
* package-lock.json > Fichier de verrouillage des dépendances
* plant\_manager.db > BDD stockant les valeurs des capteurs
* README.md > Documentation du projet
* serveur.js > Serveur Node.js principal
* simu-data\_capteur.py > Script de simulation de capteurs
* utilisateurs.json > Liste des utilisateurs en JSON

### Script Python

Après avoir valider indépendamment l'

Le script python du système Plant Manager fonctionne sur un Raspberry Pi pour gérer un système d'arrosage automatique et lire les valeurs des divers capteurs présents.

Chaque capteur à son environnement dédie pour être utilisé correctement via I²C et GPIO tel que :

* Le capteur de luminosité *(VEML7700)*
* Le capteur de CO₂ *(via ADS1115 - ADC)*
* Une image contenant clipart, symbole, conception

  Le contenu généré par l’IA peut être incorrect.Le capteur d'humidité du sol *(via ADS1115 - ADC)*
* Le capteur de niveau d'eau *(flotteur sur GPIO)*
* Le bouton poussoir *(pour activation manuelle de la pompe)*
* La pompe à eau *(contrôlée via GPIO)*
* Les LED indicatrice *(indique si le réservoir d’eau est vide ou plein)*

De plus chaque capteur possède sa propre fonction de lecture de données :

* lire\_luminosite() : Renvoie la luminosité en lux.
* lire\_co2() : Convertit la tension mesurée en concentration de CO₂ (ppm).
* lire\_humidite\_sol() : Convertit la tension mesurée en humidité du sol (%).
* lire\_flotteur() : Vérifie si le réservoir est vide ou plein.
* lire\_bouton\_poussoir() : Vérifie si l'utilisateur appuie sur le bouton.

Nous retrouvons aussi une fonction d’enregistrement des données dans la base de données : enregistrer\_donnees() :

Ainsi que les fonction permettant le contrôle de la pompe à eau

set\_pump\_state(state) : Permet de mettre à jour l’état de la pompe dans controle.json et active/désactive la pompe.

gerer\_pompe() : Permet d’activer la pompe si l’humidité est trop basse (<30%) ou si le bouton est pressé et de désactiver la pompe si l’humidité dépasse 60%.

Toutes ces fonctionnalités permettant de réaliser plusieurs missions comme la création des tables dans la base de données si elles n’existent pas, la lecture des capteurs et enregistrement des données toutes les 5 secondes et le pilotage d’actionneur et le nettoyage des GPIO à l’arret du programme.

### Serveur

La conception de l’interface web de PlantManager s’appuie sur un serveur basé sur Node.js et Express.js permettant la gestion de l’acquisition et de l’affichage des données des capteurs, mais aussi le contrôle de composant du système.

#### Dépendance Node JS

Le serveur repose sur plusieurs bibliothèques importées avec la commande require(). Ces bibliothèques sont essentielles pour assurer le bon fonctionnement des différentes fonctionnalités du projet. Elles sont installer depuis le fichier package.json et donc contrôler via le serveur Node js

Nous utilisons pour Plant Manager, les dépendances suivantes :

* express : Gestion des routes et API.
* express-session : Gestion des sessions utilisateur.
* sqlite3 : Gestion de la base de données pour stocker les données des capteurs.
* fs : Gestion des fichiers de configuration.
* nodemailer : Gestion des envois d’alertes par e-mail.
* onoff : Gestion des GPIO pour la pompe.
* ws : Gestion des WebSockets pour mise à jour en temps réel.

#### Fonctionnalité :

Afin de réaliser les différentes fonctionnalités du projet, le serveur web ( serveur.js) utilise les méthode GET et POST afin de gérer l’envoie et la réception d’informations afin de mener à bien l’utilisation de la page web. Ces 2 méthodes complémentaires permettent fournir les informations attendues à l’interface web et aux autres clients du système sans modifier l’état du serveur.

* **Méthode GET**

La méthode GET est utilisée pour récupérer des données du serveur sans modifier l'état du système. Elle est principalement utilisée pour afficher les informations des capteurs, l'état des composants et les sessions utilisateur.

* **Méthode POST**

La méthode POST est utilisée pour envoyer des données au serveur afin de modifier son état. Elle permet par exemple d'inscrire un utilisateur, d'activer la pompe ou d'envoyer des alertes.

#### Liste des API de Plant Manager

|  |  |  |
| --- | --- | --- |
| **Méthode** | **Endpoint** | **Objectif** |
| POST | /api/inscription | Inscription d'un nouvel utilisateur. Vérifie si l'email existe déjà, sinon ajoute l'utilisateur à `utilisateurs.json`. |
| POST | /api/connexion | Connexion d'un utilisateur. Vérifie les identifiants dans `utilisateurs.json` et crée une session. |
| GET | /api/verifier-session | Vérifie si l'utilisateur est connecté et retourne ses informations (nom, email, mot de passe). |
| POST | /api/deconnexion | Déconnecte l'utilisateur en détruisant la session. |
| POST | /api/modifier-utilisateur | Modifie les informations du compte utilisateur connecté (nom, email, mot de passe). |
| DELETE | /api/reinitialiser-donnees | Vide les tables `data\_capteur` et `niveau\_eau` dans la base SQLite. |
| GET | /api/donnees-capteurs | Récupère les dernières valeurs des capteurs stockées dans `data\_capteur`. |
| GET | /api/statut-systeme | Récupère la dernière valeur du niveau d'eau dans `niveau\_eau`. |
| GET | /api/historique-capteurs?limite=X | Récupère l'historique des valeurs des capteurs (limite paramétrable). |
| GET | /api/historique-eau?limite=X | Récupère l'historique du niveau d'eau (limite paramétrable). |
| POST | /api/enregistrer-configuration | Enregistre ou met à jour une configuration de plante avec ses capteurs idéaux. |
| GET | /api/recuperer-configurations | Récupère la liste de toutes les configurations enregistrées. |
| GET | /api/recuperer-config/:nom\_configuration | Récupère une configuration spécifique avec ses capteurs. |
| DELETE | /api/supprimer-configuration/:nom\_configuration | Supprime une configuration de plante et ses capteurs associés. |
| POST | /api/envoyer-alertes | Envoie un e-mail d'alerte si des anomalies sont détectées dans les capteurs (avec un délai de 10 minutes entre chaque alerte). |
| GET | /api/statut-pompe | Récupère l'état actuel de la pompe (on/off) depuis `control.json`. |
| POST | /api/definir-pompe | Active ou désactive la pompe (écrit dans `control.json`). |
| GET | /toggle-script?state=on/off | Démarre ou arrête le script `simu\_data\_capteur.py` qui lit les valeurs des capteurs. |

### Base de données

Dans le cadre du projet PlantManager, une base de données (plant\_manager.db) est utilisée pour stocker et gérer les données des capteurs en temps réel ainsi que les configurations des valeurs idéales de chaque plante ajouter par l’utilisateur. L'objectif est d'assurer un suivi précis des paramètres environnementaux et d'optimiser la gestion automatique des plantes via l’interface web

La base de données est donc composée de plusieurs tables stockant ainsi diverses informations du système PlantManager

#### Table : niveau\_eau

* Permet de stocker l’état du réservoir d’eau du système ainsi l’horaire de la mesure.
* Envoie des données depuis le script python
* Lecture des données sur l’interface web (dashboard)

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | INTEGER | Identifiant unique |
| valeur | INTEGER | Niveau d'eau |
| horaire | DATETIME | Date et heure de la mesure |

#### Table : configurations

* Permet de définir des configurations spécifiques pour chaque type de plante.
* Associer un nom de plante à une configuration prédéfinie
* Envoie des données depuis l’interface web (configuration)
* Lecture des données depuis l’interface web (configuration, dashboard)

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | INTEGER | Identifiant unique |
| nom\_configuration | TEXT | Nom de la configuration (ex: "Configuration Plante #1", |
| nom\_plante | TEXT | Nom de la plante associée (ex: "Cactus", "Tomates") |

#### Table : capteurs\_configuration

* Associer des valeurs idéales aux capteurs pour chaque configuration
* Déterminer les seuils optimaux (ex: humidité idéale (en %) = [30, 60])
* Envoie des données depuis l’interface web (configuration)
* Lecture des données depuis l’interface web (configuration, dashboard)

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | INTEGER | Identifiant unique |
| id\_configuration | INTEGER | Référence à la table configurations |
| type | TEXT | Type de capteur (ex: température, humidité) |
| valeur\_ideale | TEXT | Valeur idéale attendue |
| unite | TEXT | Unité de la valeur |

#### Table : data\_capteur

* Archiver et datés les mesures des capteurs pour l’analyse de l’environnement
* Suivi des variations pour une meilleure gestion
* Envoie des données depuis le script python (valeurs réelles des capteurs)
* Lecture des données sur l’interface web (dashboard, data-evolution)

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | INTEGER | Identifiant unique |
| type | TEXT | Type de capteur (température, humidité, etc.) |
| valeur | REAL | Valeur mesurée |
| unite | TEXT | Unité de mesure |
| horaire | DATETIME | Date et heure de la mesure |

#### Arborescence de la base de données

Une image contenant texte, capture d’écran, nombre, Police

Le contenu généré par l’IA peut être incorrect.

### Interface client

L'interface web partie client de PlantManager est conçue pour offrir aux utilisateurs un environnement de surveille simple et efficace afin de bien gérer leurs plantes

#### INDEX

La page index.html est la première page que l’utilisateur voit lorsqu’il accède à l’interface web de Plant Manager. Elle a pour objectif de fournir une vue d’ensemble du projet.

C cette page permet d'informer l'utilisateur sur les capacités principales de PlantManager tout en lui donnant un accès direct à l'espace utilisateur.

Ainsi elle permet de :

* Présenter PlantManager et son rôle dans la gestion automatisée des plantes.
* Expliquer les fonctionnalités principales du système (suivi des données, automatisation, personnalisation).
* Accès à l'espace utilisateur pour permettre la connexion et l’inscription pour accéder par la suite au système.

Une image contenant texte, capture d’écran, logo, conception

Le contenu généré par l’IA peut être incorrect.Image de index.html

#### CONNEXION

La page de connexion de PlantManager est une interface permettant aux utilisateurs de se connecter ou de s’inscrire afin accéder à PlantManager avec un espace personnel et pouvoir interagir avec le système. Seul une connexion permet d’accéder au système. Si un client essaye d’aller sur d’autre page du système sans être connecté il sera automatiquement renvoyé ici.

Cette page assure une gestion sécurisée et personnalisé des accès pour l’identification des utilisateurs.

Nous demandons le nom, un mail et un mot de passe permettant de reconnaitre chacun des utilisateurs.

Le mail fournit servira par la suite à envoyer les alertes directement par mail

Pour cela, le fichier connexion.js gère l’interactivité et l'authentification de l'utilisateur avec les fonctionnalités suivantes :

* **Basculement entre connexion et inscription** :
  + Lorsqu’un utilisateur clique sur Connexion, la section de connexion s’affiche et l'inscription est masquée.
  + Lorsqu’un utilisateur clique sur Inscription, la section d'inscription apparaît à la place.
* **Gestion de la connexion** :
  + Envoi des informations (email, mot de passe) via une requête POST à */api/connexion*
  + Vérification de la réponse du serveur et redirection vers le tableau de bord en si OK via la requête API /api/vérifier-session (serveur.js)
  + Affichage d’alerte en cas d’erreur
* **Gestion de l'inscription** :
  + Envoi des informations (nom, email, mot de passe) via une requête POST a l’API */api/inscription*
  + Vérification de la création du compte et basculement automatique vers la section connexion.
  + Affichage d’alerte en cas d’erreur
* **Serveur**

Le fichier serveur.js assure la gestion des utilisateurs via un fichier JSON (utilisateurs.json) stockant les comptes utilisateur.

Image de connexion.html (section connexion)

Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.

Image de connexion.html (section inscription)

Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.

#### DASHBOARD

Le Tableau de Bord de PlantManager constitue l'interface principale permettant aux utilisateurs de suivre en temps réel l’état de leurs plantes.

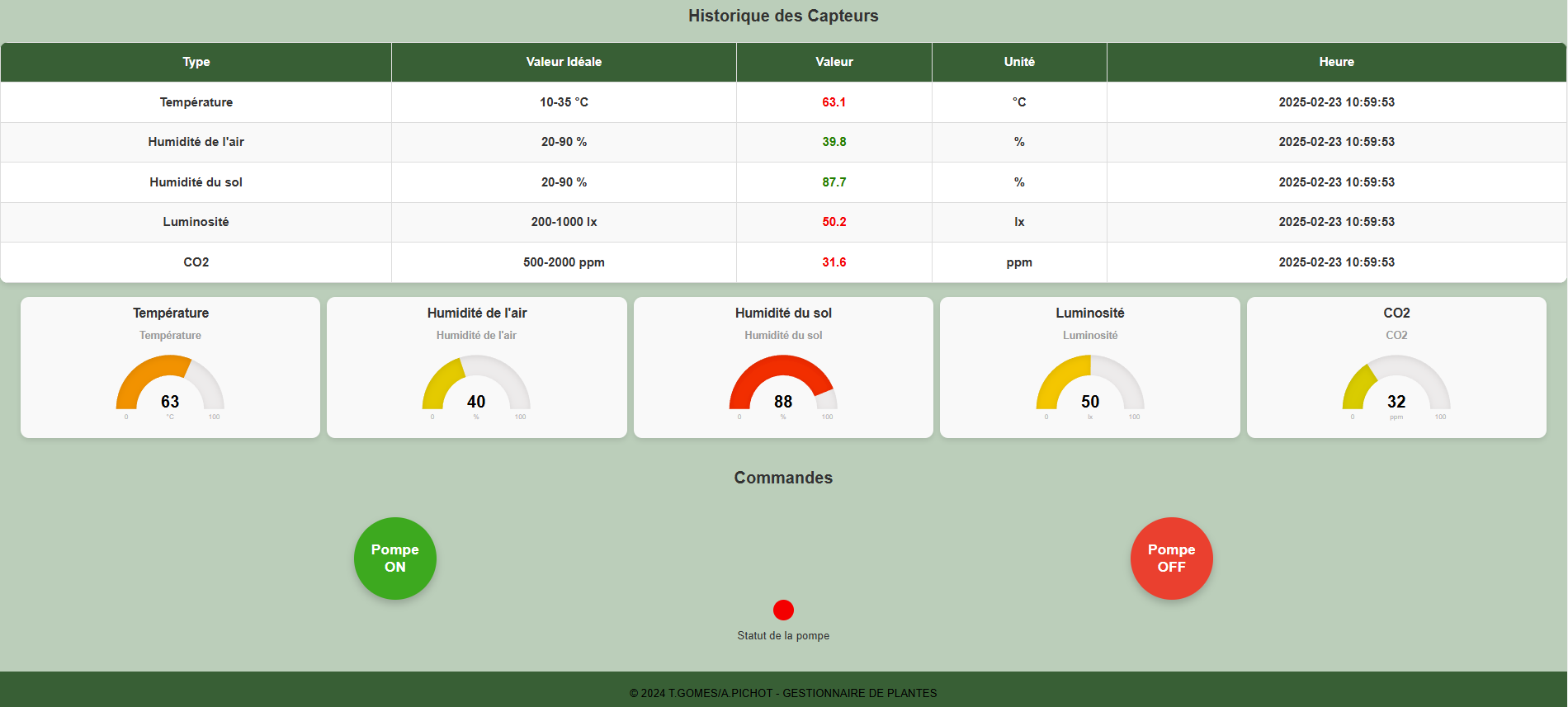
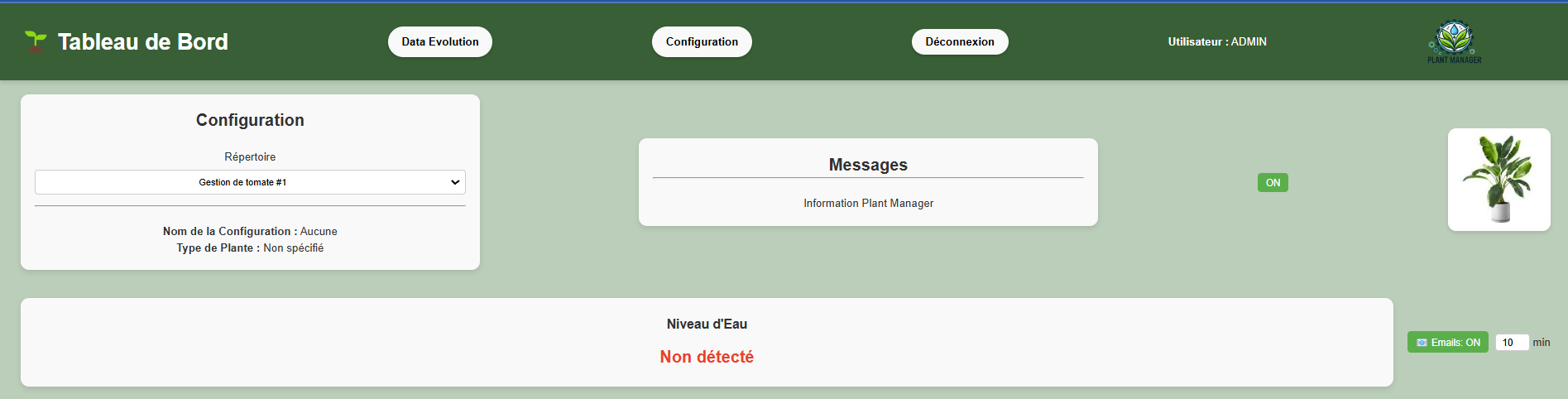
Il permet de réunir de manière efficace (tableau et jauges) les informations des capteurs, et possède des commandes d’action afin de piloter les actionneurs du système (ex : pompe à eau) et permet de sélectionner la configuration souhaite par l’utilisateur

De plus, il permet de lancer le système de mesure de PlantManager (script python) et permet aussi d’activer ou de désactiver le système d’envoi d’alerte par mail en choisissant son intervalle de réception de message par mail

Pour cela le fichier dashboard.js s’occupe de la :

* Vérification de la connexion utilisateur :
  + Redirection vers la page de connexion si aucun utilisateur n’est authentifié.
  + Affichage dynamique du nom de l’utilisateur connecté.
* Gestion des configurations :
  + Récupération et affichage des configurations via /api/récupération-configurations.
  + Chargement des valeurs des capteurs associées à la configuration sélectionnée.
* Surveillance des capteurs :
  + Mise à jour automatique des jauges interactives (JustGage).
  + Vérification de l’adéquation avec les valeurs idéales de la configuration sélectionnée.
  + Affichage d’alertes en cas de dépassement des seuils définis.
* Indicateurs système :
  + Surveillance du niveau d’eau et affichage en vert (haut) ou rouge (bas).
  + Gestion de l’envoi des alertes par email à intervalles définis.
* Contrôle des équipements :
  + Commande de la pompe avec activation/désactivation (/api/définir-pompe).
  + Mise à jour des indicateurs visuels en fonction des actions.
* Activation du système de collecte des données :
  + Envoi de requêtes à /toggle-script pour démarrer ou arrêter la récupération des données.
* Déconnexion :
  + Gestion de la déconnexion et redirection vers la page d’accueil.

Image du tableau de bord



#### CONFIGURATION

La page de configuration de PlantManager permet aux utilisateurs de créer, modifier et supprimer les paramètres de configuration de leurs plantes en fonction des conditions idéales d’environnement. De plus aller permet aussi de modifier les informations de compte d’un utilisateur. Enfin elle offre un lien vers un site de recherche de données de cultivation de plantes nommé permapeople.org.

Pour cela le fichier configuration.js gère les interactions à la demande de l’utilisateur :

* **Vérification de la connexion utilisateur** :
  + Si aucun utilisateur n'est connecté : redirection vers page de connexion
  + Affichage du nom, email et mot de passe si utilisateur connecté.
* **Déconnexion**:
  + Si bp appuyé : envoi la requête POST via /*api/deconnexion*
  + Redirection vers page d'accueil après déconnexion
* **Gestion des configurations de plantes** :
  + Chargement des configurations déjà existantes via la requête API à */api/recuperer-configurations* en interrogeant la base de données
  + Récupération et affichage des détails dans le formulaire d'une configuration sélectionnée via la requête API /*api/recuperer-config/:nom\_configuration* en interrogeant la base de données
  + Enregistrement d’une nouvelle configuration en envoyant les paramètres à la base de données via la requête API /*api/enregistrer-configuration*
  + Suppression d’une configuration avec demande de confirmation via la requête API */api/supprimer-configuration/:nom\_configuration* en interrogeant la base de données
* **Gestion du compte utilisateur** :
  + Modification du nom, email ou mot de passe via la requête API /api/modifier-utilisateur en interrogeant le fichier *utilisateur.json*
  + Mise à jour des informations utilisateur affichées.
* **Serveur**

Le fichier serveur.js assure la gestion des configurations via la base de données et la mise à jour des informations de l’utilisateur dans le fichier utilisateur.json

Image de la partie configuration de l’interface web

Une image contenant texte, capture d’écran, Police, logo

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, nombre, Police

Le contenu généré par l’IA peut être incorrect.

#### DATA-EVOLUTION

La page data-evolution permet aux utilisateurs de visualiser en temps réel l’évolution des mesures captées par les capteurs. Grâce à un graphique dédié par capteur, ceci permet d’analyser les mesures stocker dans la base de données en réglant le nombre de point de mesure à visualisation (10, 50, 100 ou toutes les valeurs de la base de données pour capteur)

De plus un bouton de suppression de valeurs est présent, elle permet à l’utilisateur de vider la base de données si trop de valeur y sont stocker (graphes illisibles)

Pour cela le fichier data-evolution.js gère la mise à jour des données pour les graphiques :

* **Vérification de la connexion utilisateur** :
  + Si aucun utilisateur n'est connecté : redirection vers page de connexion
  + Affichage du nom, email et mot de passe si utilisateur connecté.
* **Déconnexion**:
  + Si bp appuyé : envoi la requête POST via /*api/deconnexion*
  + Redirection vers page d'accueil après déconnexion
* **Sélection du nombre des données** :
  + L’utilisateur peut choisir d’afficher toutes les données ou 10, 50 ou 100 dernières valeurs des capteurs présent de la base de données.
  + Les graphiques se mettent à jour en fonction du nombre sélectionné.
* **Affichage des graphiques dynamiques** :
  + Graphiques pour chaque capteur : Température, Humidité de l'air, Humidité du sol, Luminosité, CO2, Niveau d'eau
  + Graphiques via Chart.js
* **Reset des données** :
  + Un bouton permet de vider les données stockées en envoyant une requête DELETE à /api/reinitialiser-donnees.
  + Une confirmation utilisateur est demandé avant que ce soit supprimer.
* **Serveur**

Le fichier serveur.js gère les requêtes liées aux données des capteurs présent dans la base de données afin d’afficher correctement ce que l’utilisateur attend

Une image contenant texte, diagramme, capture d’écran, ligne

Le contenu généré par l’IA peut être incorrect.Affichage de toutes les valeurs de température stocker dans la base de données

Une image contenant texte, diagramme, capture d’écran, Tracé

Le contenu généré par l’IA peut être incorrect.Affichage des 10 dernières valeurs de température stocker dans la base de données

### Intégration du système sur Raspberry

Pour finir, une fois l’ensemble de la partie logiciel terminé, la dernière étape est L’intégration de sur Raspberry Pi du système PlantManage. Ceci permet de rendre le système totalement autonome et fonctionnel en conditions réelles.

En effet, la Raspberry Pi joue un rôle central dans la gestion des capteurs, l’automatisation des actions et l’interface web.

Pour se faire, j’ai du mettre en place sur la raspberry, un environnement virtuel Python qui permet d’isoler les dépendances et d’exécuter le script de collecte des données de PlantManager sans conflits avec d’autres logiciels installés sur le Raspberry Pi

### Raspberry Pi Zero 2 W : Amazon.fr: Informatique

### CONCLUSION

Ainsi, pour le bon développement du projet PlantManager permettant d’offrir une gestion automatisée des plantes à travers un système de suivi et de contrôle. J’ai conçu différentes parties d’un système complet en réalisant différentes sections : Partie capteur & actionneur, avec quelques capteurs et un actionneur ; Une fois tous les capteurs programmés : création du script final regroupant l’intégralité des capteurs et actionneurs ; Partie Logiciel avec le serveur Node JS et la base de données ; ainsi que la partie Interface web avec l’Accueil, la connexion Tableau de bord, Configuration et l’Évolution des données. Pour finir une fois tout cela terminé j’ai intégré tout cette ensemble logiciel du système PlantManager à la Raspberry Pi Zero 2W

Tout cela permet aux utilisateurs de surveiller en temps réel les paramètres environnementaux de leurs plantes, d'ajuster les configurations selon les besoins spécifiques, et d'interagir avec les équipements connectés.

Bien sûr comme tout projet, des changements d’idée et des décisions modifiant le projet de base ont été faites, ainsi que des retards de temps de développement à la suite d’imprévu au sein de l’équipe, ne rendant pas opérationnelle l’ensemble de nos idées. Mais bien que des contraintes soit présentes, nous avons, moi comme mon binôme, réaliser quelques choses de fonctionnel et efficace pour un projet innovant et simple d’utilisation

# Annexes

**Annexe 1**

Une image contenant capture d’écran, diagramme, texte, ligne

Le contenu généré par l’IA peut être incorrect.