

UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

TX - DÉVELOPPEMENT D'UNE APPLICATION DE GESTION DE POTAGERS

Responsable

Mr. BOUABDALLAH Abdelmadjid

API POTAGERCREATOR



Remerciement

Nous souhaitons tout d'abord exprimer notre gratitude envers l'équipe enseignante de l'UV TX pour avoir mis en place les structures et le soutien nécessaires permettant aux étudiants de mener à bien des projets personnels en parallèle de leurs cours. Ces opportunités enrichissent significativement notre parcours et encouragent l'apprentissage pratique.

Nous remercions tout particulièrement Monsieur Abdelmadjid BOUABDALLAH, notre référent TX, pour son encadrement et sa communication claire concernant les modalités de l'UV. Son soutien, même face à nos retards, a été précieux pour l'aboutissement de ce projet.

Nous adressons également nos remerciements à Messieurs Etienne ARNOULT, Claude-Olivier SARDE, Mehdi SERAIRI et Ahmed LOUNIS pour leur compréhension et leur bienveillance, nous ayant permis de mener ce projet à son terme malgré les délais supplémentaires nécessaires.

supervisé par
Mr. Abdelmadjid BOUABDALLAH

Sommaire

| | | |
|----------|--|-----------|
| 1 | Introduction | 5 |
| 2 | Cahier des Charges | 6 |
| 2.1 | SUJET | 6 |
| 2.2 | BESOIN | 6 |
| 2.3 | DESCRIPTION FONCTIONNELLE | 6 |
| 2.4 | ORGANISATION DE L'INTERFACE UTILISATEUR | 7 |
| 3 | Developpement de l'API | 8 |
| 3.1 | ARCHITECTURE UML ET BASE DE DONNÉE | 8 |
| 3.1.1 | ENTITÉ UTILISATEUR | 9 |
| 3.1.2 | ENTITÉ JARDIN | 9 |
| 3.1.3 | ENTITÉ VEGETAUX | 9 |
| 3.1.4 | ENTITÉ ADMIN | 10 |
| 3.1.5 | ENTITÉ COMPO__JARDIN | 10 |
| 3.1.6 | RELATIONS | 10 |
| 3.1.7 | IMPLÉMENTATION DE LA BASE DE DONNÉE | 10 |
| 3.2 | ACHITECTURE WEB | 14 |
| 3.2.1 | PAGE DE CONNEXION/INSCRIPTION | 14 |
| 3.2.2 | ESPACE UTILISATEUR | 16 |
| 3.2.3 | CRÉATION JARDIN | 16 |
| 3.2.4 | AJOUT D'UN VÉGÉTAUX | 17 |
| 3.3 | MOTEUR DE CALCUL PRÉVISIONNEL | 18 |
| 4 | Sécurisation de l'API | 21 |
| 4.1 | GESTION ET SÉCURITÉ DES ROUTES | 21 |
| 4.1.1 | TECHNIQUES UTILISÉES : VÉRIFICATION DES RÔLES ET SESSIONS | 21 |
| 4.2 | CRYPTAGE DES MOTS DE PASSE | 21 |
| 4.2.1 | TECHNIQUE UTILISÉE : MIDDLEWARE BCRYPT | 21 |
| 4.3 | PRÉVENTION DES ATTAQUES PAR INJECTION | 21 |
| 4.3.1 | TECHNIQUE UTILISÉE : REQUÊTES PARAMÉTRÉES | 21 |

| | | |
|----------|---|-----------|
| 5 | Ressentis et Axes d'amélioration | 23 |
| 5.1 | RESSENTIS | 23 |
| 5.2 | AXES D'AMÉLIORATION | 23 |
| 6 | Conclusion | 24 |
| 7 | Annexe - Vidéo Présentation | 25 |

Table des figures

| | | |
|---|-------------------------------------|----|
| 1 | <i>Diagramme UML</i> | 8 |
| 2 | <i>Accueil</i> | 14 |
| 3 | <i>Inscription</i> | 15 |
| 4 | <i>Connexion</i> | 15 |
| 5 | <i>Espace utilisateur</i> | 16 |
| 6 | <i>Création Jardin</i> | 16 |
| 7 | <i>Ajout végétaux</i> | 17 |

1 Introduction

L'agriculture et le jardinage sont des domaines où la technologie peut apporter une valeur ajoutée significative en fournissant des informations précises et en temps réel pour optimiser la croissance des plantes. Dans ce contexte, notre projet vise à développer une API de gestion de potager intelligent, qui utilise les données météorologiques pour recommander les zones les plus favorables à la culture de légumes et de fruits en France.

L'objectif principal de cette API est de fournir des informations précieuses aux jardiniers amateurs et professionnels, leur permettant de prendre des décisions éclairées sur les cultures à planter, en fonction des conditions météorologiques locales. En utilisant les données météorologiques, l'API peut prédire selon les zones et les périodes les plus favorables à la croissance de différents types de légumes et de fruits, en tenant compte des facteurs tels que la température, la géolocalisation, etc...

L'API de gestion de potager intelligent est conçue pour être facilement intégrée dans les applications mobiles et web, offrant ainsi une expérience utilisateur transparente et conviviale. En fournissant des informations précises et en temps réel, l'API peut aider les utilisateurs à optimiser la croissance de leurs plantes, à réduire les déchets et à améliorer leur rendement global.

Dans ce rapport, nous présenterons les différentes étapes de la conception et du développement de l'API, ainsi que les résultats obtenus lors des tests et des validations. Nous décrirons également les défis rencontrés lors du développement de l'API et les solutions mises en œuvre pour les surmonter. Enfin, nous discuterons des perspectives d'avenir pour l'API de gestion de potager intelligent et les améliorations potentielles qui pourraient être apportées pour en améliorer encore les performances et la fonctionnalité.

2 Cahier des Charges

2.1 Sujet

Cette TX a pour objectif le développement d'une application web dédiée à la gestion de potagers permettant la contribution à la démocratisation de cette pratique chez les particuliers. Son objectif principal est de fournir aux utilisateurs une plateforme conviviale et intuitive pour planifier, suivre et optimiser la gestion de leurs potagers. Les fonctionnalités prévues comprennent la planification des cultures, la gestion des semis, le suivi des arrosages et des traitements, ainsi que la tenue d'un journal de bord permettant d'enregistrer les observations et les résultats de chaque culture. En outre, l'application vise à aider les utilisateurs à optimiser leurs plantations en exploitant des données de qualité sur le sol et les conditions météorologiques. En intégrant ces informations, les utilisateurs pourront prendre des décisions éclairées pour améliorer leurs pratiques de jardinage et obtenir des récoltes plus abondantes et de meilleure qualité.

2.2 Besoin

Une application web interactive de conseils en potager pour aider les utilisateurs à planifier et gérer leur jardin potager en fonction de leurs préférences alimentaires, de leur emplacement géographique et de la taille de leur terrain. Cette plateforme fournira des recommandations sur les périodes de plantation des légumes, basées sur les données météorologiques de l'année précédente de la localisation du jardin de l'utilisateur.

- Fonctionnalités de l'application :
 - Ajouter un profil utilisateur
 - Sélectionner les légumes à cultiver et la quantité
 - Afficher les recommandations de plantation en fonction de la localisation, de la météo.
 - Accéder à une base de données de légumes avec des informations détaillées sur chaque variété
 - Recevoir des notifications sur les recommandations de plantation et d'entretien du potager

2.3 Description fonctionnelle

- Acteurs :
 - Utilisateur (candidat)
 - Administrateur (gère les utilisateurs et leurs informations)
- Authentification :
 - Les utilisateurs doivent créer un compte en fournissant leur nom, prénom, adresse e-mail, mot de passe sécurisé, emplacement géographique et taille du terrain.
 - L'administrateur possède un accès privilégié pour gérer les données de la plateforme.

- Fonctionnalités pour l'utilisateur :
 - Ajouter des informations sur leur potager (emplacement, taille, etc.)
 - Sélectionner les légumes qu'ils souhaitent cultiver
 - Recevoir des recommandations de plantation en fonction de leur localisation et des données météorologiques
- Fonctionnalités pour l'administrateur :
 - Gérer la base de données des légumes et de leurs conditions de croissance
 - Mettre à jour les recommandations de plantation en fonction des données météorologiques

2.4 Organisation de l'interface utilisateur

- Écran d'accueil :
 - Page de connexion/inscription pour les utilisateurs
 - Accès direct aux fonctionnalités après authentification
- Écran utilisateur :
 - Ajouter/modifier les informations du potager
 - Sélectionner les légumes à cultiver
 - Afficher les recommandations de plantation
- Écran administrateur :
 - Gérer la base de données des légumes
 - Mettre à jour les recommandations de plantation
- Livrables attendus :
 - Spécifications détaillées des fonctionnalités de l'application
 - Conception du modèle de données
 - Prototype de l'interface utilisateur (HMI)

3 Développement de l'API

Pour le développement de notre API de gestion de potager, nous avons choisi d'utiliser comme langages JavaScript pour le développement pur de l'architecture Web, Python pour le moteur de calcul prévisionnel, ainsi que la base de données PostgreSQL pour stocker les données localement dans un premier temps.

3.1 Architecture UML et Base de donnée

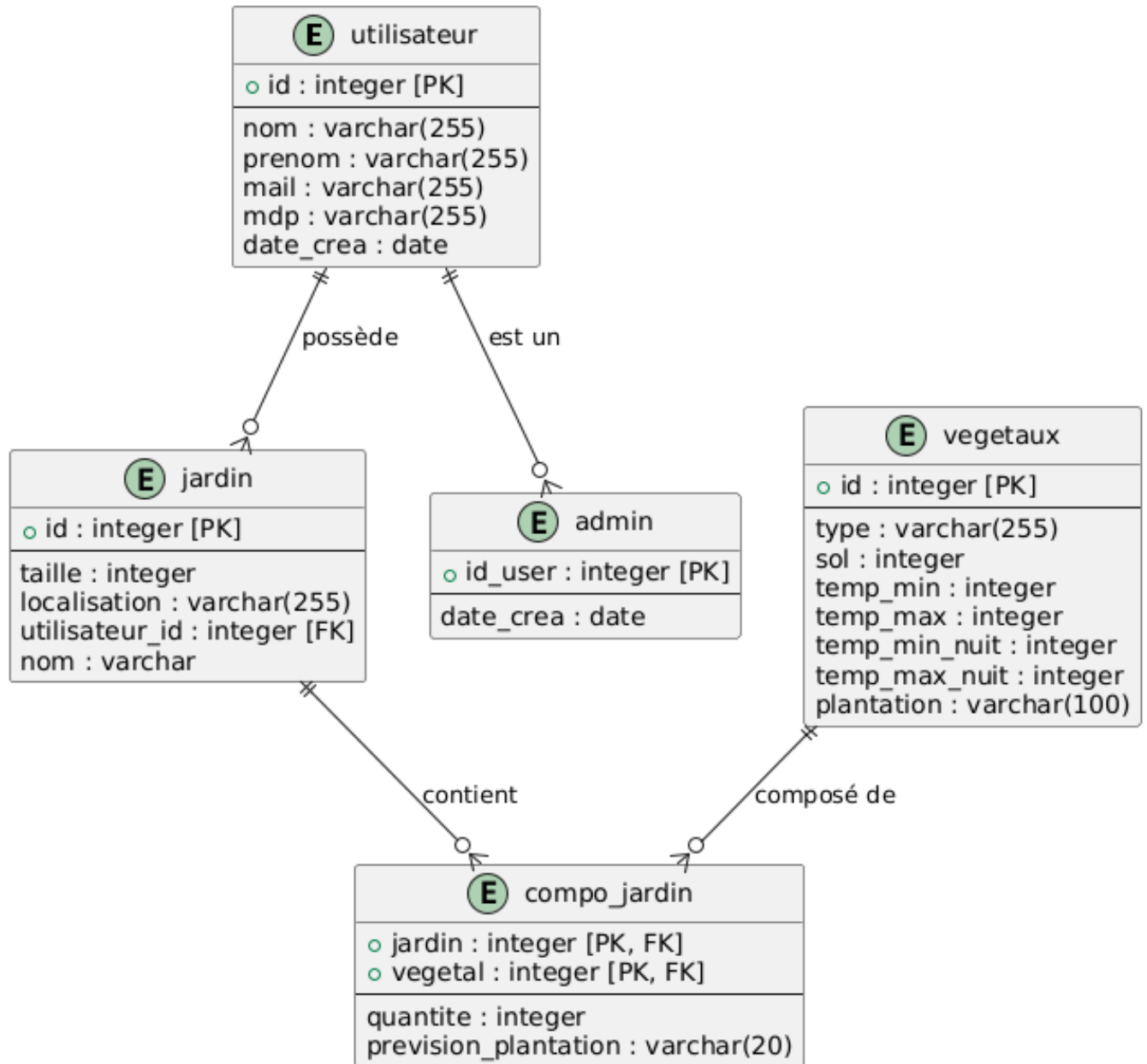


FIGURE 1 – *Diagramme UML*

3.1.1 Entité utilisateur

L'entité utilisateur représente les utilisateurs de l'application. Chaque utilisateur peut s'inscrire, gérer ses jardins et potentiellement accéder à des fonctionnalités administratives s'il est désigné comme administrateur. Cette entité contient les attributs suivants :

- id : Identifiant unique de l'utilisateur, utilisé comme clé primaire.
- nom : Nom de l'utilisateur.
- prenom : Prénom de l'utilisateur.
- mail : Adresse e-mail de l'utilisateur, qui peut également être utilisée pour l'authentification.
- mdp : Mot de passe de l'utilisateur, probablement stocké de manière sécurisée (haché).
- date_crea : Date de création du compte de l'utilisateur.

3.1.2 Entité jardin

L'entité jardin représente un jardin individuel qu'un utilisateur peut gérer. Elle contient les informations sur le jardin et le relie à un utilisateur spécifique :

- id : Identifiant unique du jardin, utilisé comme clé primaire.
- taille : Taille du jardin en mètres carrés ou dans une autre unité définie.
- localisation : Localisation ou emplacement du jardin. Cela pourrait être une adresse ou une description textuelle.
- utilisateur_id : Identifiant de l'utilisateur qui possède le jardin, définissant ainsi une relation entre un jardin et un utilisateur.
- nom : Nom du jardin, permettant de le différencier s'il y a plusieurs jardins par utilisateur.

3.1.3 Entité vegetaux

L'entité vegetaux représente les types de végétaux disponibles pour la plantation dans un jardin. Ces végétaux peuvent inclure des fruits, des légumes ou d'autres plantes utiles dans un jardin potager :

- id : Identifiant unique du végétal, utilisé comme clé primaire.
- type : Type de végétal, par exemple "tomate", "carotte", "fraises", etc.
- sol : Indication sur le type de sol requis pour ce végétal, exprimée sous forme numérique ou de code.
- temp_min : Température minimale à laquelle ce végétal peut pousser en journée (à midi).
- temp_max : Température maximale supportée par ce végétal en journée (à midi).
- temp_min_nuit : Température minimale à laquelle ce végétal peut pousser en pleine nuit (à 4h).
- temp_max_nuit : Température maximale supportée par ce végétal en pleine nuit (à 4h).
- plantation : Saison ou période de plantation recommandée pour ce végétal, par exemple "printemps", "été", etc.

3.1.4 Entité admin

L'entité admin identifie les utilisateurs ayant des privilèges d'administration dans l'application. Un administrateur est un utilisateur qui a des capacités de gestion supplémentaires pour superviser l'application :

- `id_user` : Référence à l'identifiant de l'utilisateur qui possède le rôle d'administrateur. Il agit ici comme une clé primaire, liée à l'entité utilisateur.
- `date_crea` : Date à laquelle l'utilisateur a été désigné comme administrateur.

3.1.5 Entité compo_jardin

L'entité compo_jardin est une table de liaison qui définit la composition d'un jardin. Elle relie un jardin à plusieurs végétaux et contient des informations spécifiques sur chaque association :

- `jardin` : Référence à l'identifiant du jardin. Cette clé agit à la fois comme une clé primaire et une clé étrangère.
- `vegetal` : Référence à l'identifiant du végétal planté dans le jardin. Cette clé est également une clé primaire et une clé étrangère.
- `quantite` : Quantité de ce type de végétal planté dans le jardin.
- `prevision_plantation` : Période prévue pour la plantation de ce végétal dans le jardin sous forme de chaîne de caractères.

3.1.6 Relations

Les relations entre les entités de ce modèle permettent de structurer l'application de manière cohérente :

- utilisateur "possède" jardin : Chaque utilisateur peut posséder un ou plusieurs jardins, ce qui est représenté par une relation un-à-plusieurs entre utilisateur et jardin.
- jardin "contient" compo_jardin : Un jardin peut contenir plusieurs types de végétaux, et cette association est représentée par la relation entre jardin et compo_jardin.
- compo_jardin "composé de" vegetaux : Cette relation illustre que chaque compo_jardin est composée de plusieurs végétaux, permettant de relier un jardin à ses plantes spécifiques.
- utilisateur "est un" admin : Cette relation montre qu'un utilisateur peut être un administrateur. Un administrateur reste un utilisateur de l'application avec des privilèges supplémentaires.

3.1.7 Implémentation de la base de donnée

Nous avons commencé par implémenter la base de donnée sur Postgres en *localhost*. En effet, il aurait été préférable d'implémenter notre base de donnée sur un SGBD qui fonctionne sur un cloud comme par exemple *PhpMyAdmin* mais malheureusement après avoir demandé des codes à Mr. Lounis nous n'avons pas eu de réponse. Notre base de donnée fonctionne donc en local sur nos propres machines mais est complètement fonctionnelle.

Pour implémenter la base de donnée, après avoir fait le diagramme UML mettant en relation les différentes classes, nous avons fait le modèle logique de données visant à définir la construction finale des tables avec leur(s) clef(s) primaires et les potentielles clefs étrangères ainsi que les différentes méthodes.

utilisateur (int id {PK}, string nom, string prenom, string mail,
string mdp, date date_crea)

jardin (int id {PK}, int taille, string localisation,
int utilisateur_id {FK -> utilisateur(id)}, string nom)

vegetaux (int id {PK}, string type, int sol, int temp_min,
int temp_max, string plantation)

admin (int id_user {PK, FK -> utilisateur(id)}, date date_crea)

compo_jardin (int jardin {PK, FK -> jardin(id)},
int vegetal {PK, FK -> vegetaux(id)}, int quantite,
string prevision_plantation {CHECK : 'janvier' à 'décembre'})

Enfin, nous avons traduis ce MLD en un script SQL pour être crée sur PostGreSQL.

```
-- Recreate the utilisateur table
CREATE TABLE IF NOT EXISTS public.utilisateur
(
    id integer NOT NULL GENERATED ALWAYS AS IDENTITY
    ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),
    nom character varying(255) COLLATE pg_catalog."default",
    prenom character varying(255) COLLATE pg_catalog."default",
    mail character varying(255) COLLATE pg_catalog."default",
    mdp character varying(255) COLLATE pg_catalog."default",
    date_crea date,
    CONSTRAINT utilisateur_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.utilisateur
    OWNER to postgres;

-- Recreate the jardin table
CREATE TABLE IF NOT EXISTS public.jardin
(
    id integer NOT NULL DEFAULT nextval('jardin_id_seq'::regclass),
    taille integer,
    localisation character varying(255) COLLATE pg_catalog."default",
    utilisateur_id integer,
    nom character varying COLLATE pg_catalog."default",
    CONSTRAINT jardin_pkey PRIMARY KEY (id),
    CONSTRAINT jardin_utilisateur_id_fkey FOREIGN KEY (utilisateur_id)
        REFERENCES public.utilisateur (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

```

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.jardin
    OWNER to postgres;

-- Recreate the vegetaux table
CREATE TABLE IF NOT EXISTS public.vegetaux
(
    id integer NOT NULL,
    type character varying(255) COLLATE pg_catalog."default",
    sol integer,
    temp_min integer,
    temp_max integer,
    plantation character varying(100) COLLATE pg_catalog."default",
    CONSTRAINT vegetaux_pkey PRIMARY KEY (id)
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.vegetaux
    OWNER to postgres;

-- Recreate the admin table
CREATE TABLE IF NOT EXISTS public.admin
(
    id_user integer NOT NULL,
    date_crea date,
    CONSTRAINT admin_pkey PRIMARY KEY (id_user),
    CONSTRAINT admin_id_user_fkey FOREIGN KEY (id_user)
        REFERENCES public.utilisateur (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.admin
    OWNER to postgres;

-- Recreate the compo_jardin table
CREATE TABLE IF NOT EXISTS public.compo_jardin
(
    jardin integer NOT NULL,
    vegetal integer NOT NULL,
    quantite integer,
    prevision_plantation character varying(20) COLLATE
pg_catalog."default",
    CONSTRAINT compo_vegetaux_pkey PRIMARY KEY (jardin, vegetal),
    CONSTRAINT fk_jardin FOREIGN KEY (jardin)
        REFERENCES public.jardin (id) MATCH SIMPLE
        ON UPDATE NO ACTION

```

```

        ON DELETE CASCADE,
CONSTRAINT fk_vegetal FOREIGN KEY (vegetal)
    REFERENCES public.vegetaux (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE CASCADE,
CONSTRAINT compo_jardin_prevision_plantation_check CHECK
(prevision_plantation::text = ANY
(ARRAY['janvier'::character varying,
'fevrier'::character varying, 'mars'::character varying,
'avril'::character varying, 'mai'::character varying,
'juin'::character varying, 'juillet'::character varying,
'aout'::character varying, 'septembre'::character varying,
'octobre'::character varying, 'novembre'::character varying,
'decembre'::character varying]::text[]))
)
TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.compo_jardin
    OWNER to postgres;

```

3.2 Architecture WEB

Nous avons choisi d'utiliser JavaScript pour le développement de notre API en raison de sa popularité et de sa flexibilité. Pour la gestion des routes et des requêtes HTTP, nous avons utilisé le framework Express pour Node.js. Nous avons également utilisé le moteur de template EJS pour la génération de vues.

3.2.1 Page de connexion/inscription

Lorsque nous lançons l'API, nous avons conçu une page de connexion et d'inscription qui permet aux utilisateurs de s'enregistrer en fournissant toutes les données nécessaires ou de se connecter.

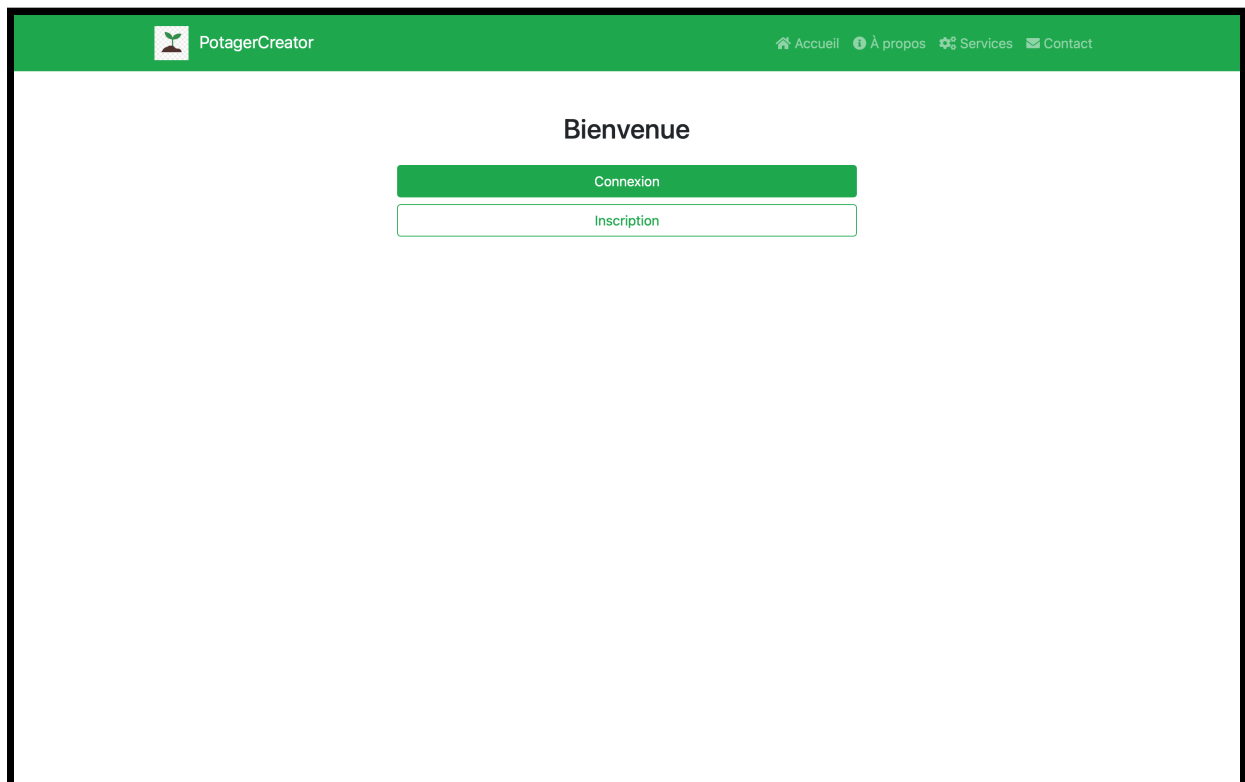
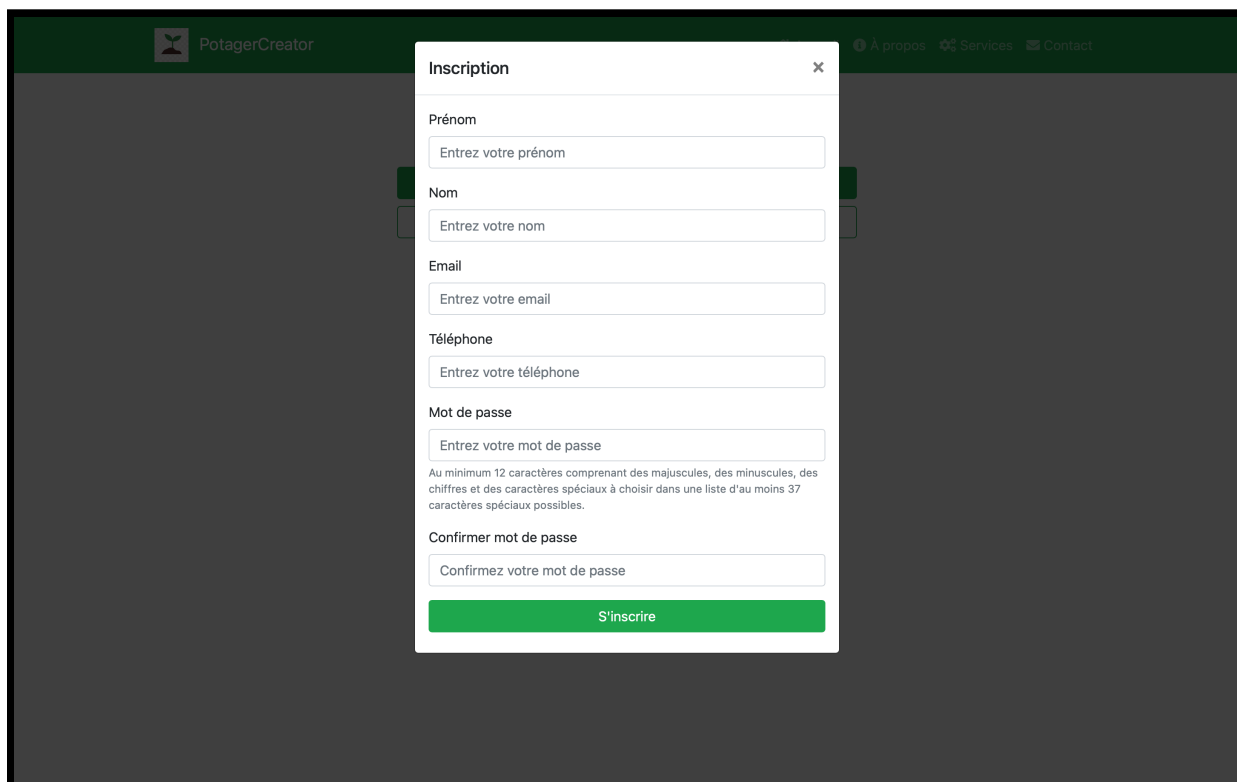
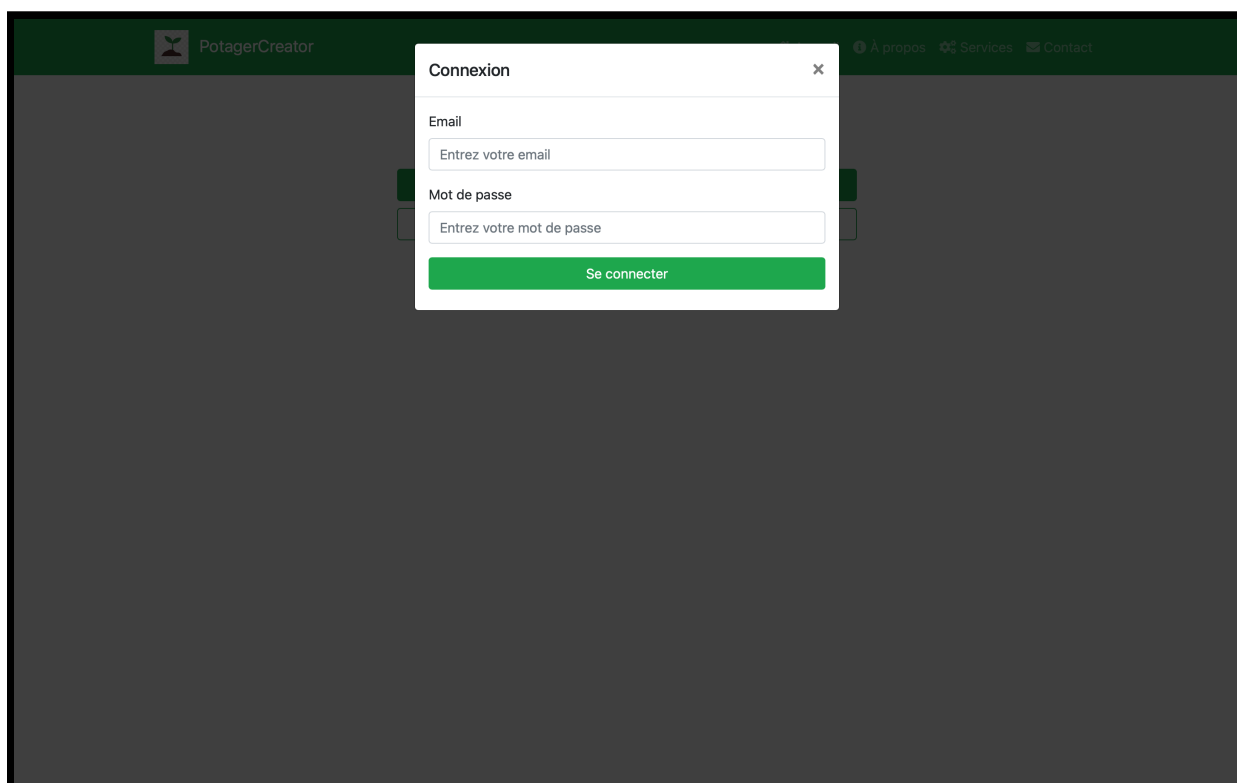


FIGURE 2 – *Accueil*



The image shows a web browser window with a dark green header. On the left is the 'PotagerCreator' logo. On the right are links: 'À propos', 'Services', and 'Contact'. A white modal window titled 'Inscription' is centered. It contains the following fields: 'Prénom' (Entrez votre prénom), 'Nom' (Entrez votre nom), 'Email' (Entrez votre email), 'Téléphone' (Entrez votre téléphone), 'Mot de passe' (Entrez votre mot de passe), and 'Confirmer mot de passe' (Confirmez votre mot de passe). Below the password fields is a green button labeled 'S'inscrire'. A note below the password field states: 'Au minimum 12 caractères comprenant des majuscules, des minuscules, des chiffres et des caractères spéciaux à choisir dans une liste d'au moins 37 caractères spéciaux possibles.'

FIGURE 3 – *Inscription*



The image shows the same web browser window as Figure 3. A white modal window titled 'Connexion' is centered. It contains two fields: 'Email' (Entrez votre email) and 'Mot de passe' (Entrez votre mot de passe). Below these fields is a green button labeled 'Se connecter'.

FIGURE 4 – *Connexion*

3.2.2 Espace utilisateur

Lorsqu'un utilisateur se connecte/s'inscrit, son espace utilisateur s'affiche. Celui-ci contient tous les jardins qu'il a enregistré sur la plateforme. C'est ici qu'il peut gérer tous ses jardins : en créer des nouveaux, ajouter des végétaux sur certains, supprimer un jardin, ou encore modifier un jardin (modifier ses végétaux, sa surface, sa localisation, etc...).

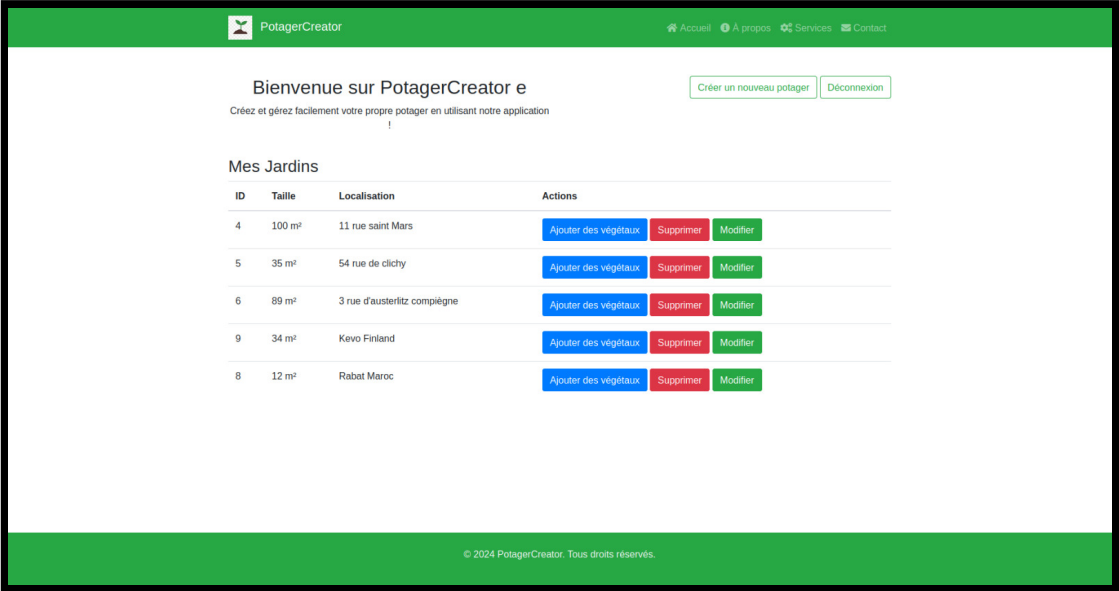


FIGURE 5 – Espace utilisateur

3.2.3 Création Jardin

Comme nous l'avons vu précédemment, une fois connecté/inscrit, si un utilisateur souhaite créer un nouveau jardin, il devra renseigner plusieurs champs obligatoire et nécessaires à son identification. Il doit en effet fournir, son nom, son adresse et sa taille en m^2 .

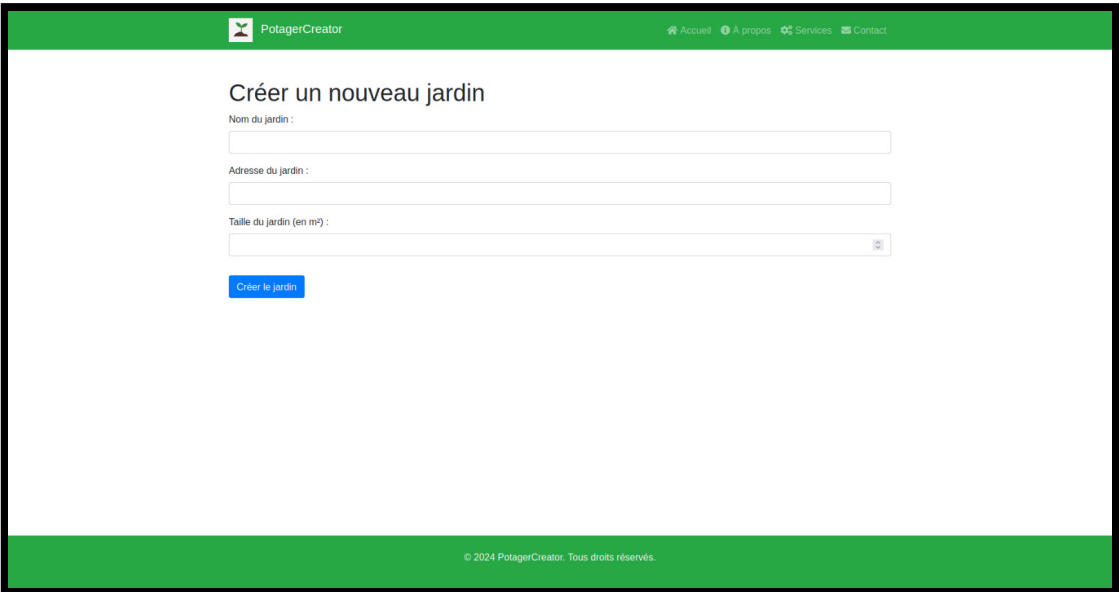


FIGURE 6 – Création Jardin

3.2.4 Ajout d'un végétaux

Lorsqu'un utilisateur souhaite voir les végétaux d'un jardin ou éventuellement d'en ajouter, il lui suffit de cliquer sur "Ajouter des végétaux". Une fois cliqué, il suffit à l'utilisateur de choisir parmi les végétaux disponible dans la liste et de renseigner une quantité.

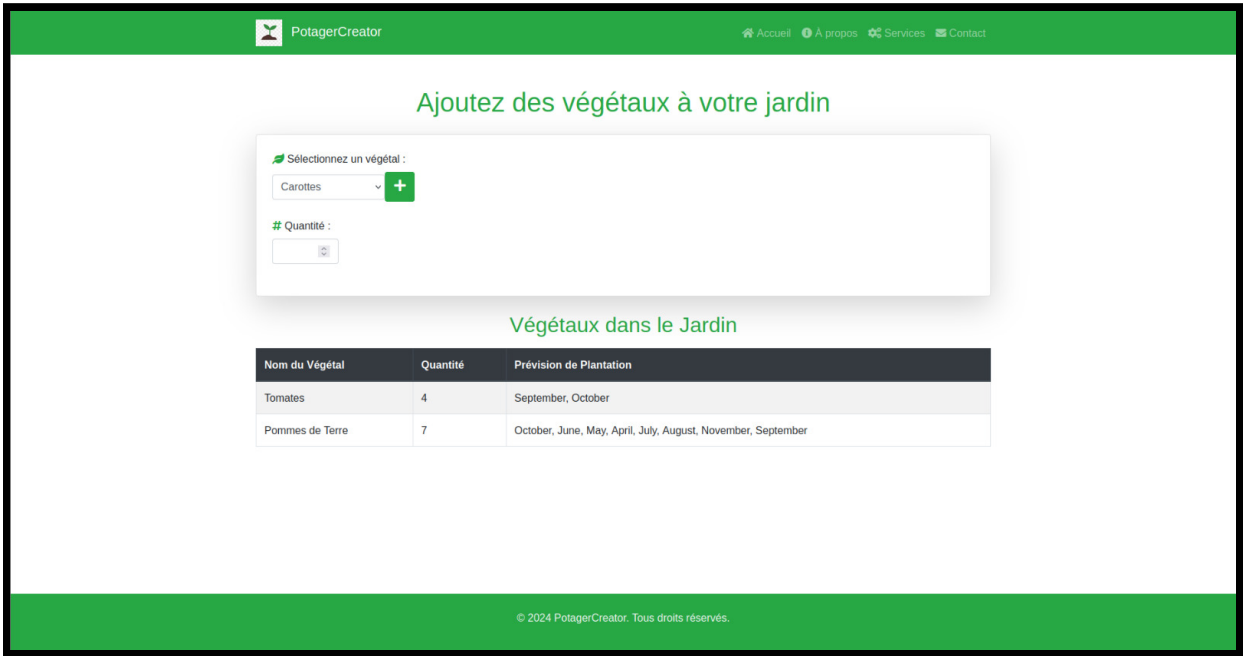


FIGURE 7 – *Ajout végétaux*

Lorsqu'on ajoute un végétal, on peut avoir une ligne d'un tableau qui se crée, contenant les informations choisies par l'utilisateur mais contenant également une colonne "Prévision de plantation".

Cette attribut permet, grâce à un moteur de calcule prévisionnel codé en python, à l'utilisateur de savoir quel(s) est/sont les mois les plus favorable à la plantation du végétal.

3.3 Moteur de calcul prévisionnel

Comme dit précédemment, dans le but de définir à l'utilisateur quels sont les mois les plus favorables à la plantation de chaque végétal choisi, nous avons conçu un moteur de calcul prévisionnel.

Ce moteur de calcul commence par vérifier si l'utilisateur a fourni trois arguments nécessaire pour calculer le(s) moi(s) favorables : une adresse, une température minimale et une température maximale à la fois pour midi et pour 4h du matin (pour la nuit) . Si l'un de ces éléments manque, il affiche un message d'erreur et se termine.

```
import sys

if len(sys.argv) < 6:
    print("Veuillez fournir une adresse, une température min et max pour midi, \
    puis une température min et max pour la nuit en arguments.")
    sys.exit(1)

adresse = sys.argv[1]
temp_min_midi = float(sys.argv[2])
temp_max_midi = float(sys.argv[3])
temp_min_nuit = float(sys.argv[4])
temp_max_nuit = float(sys.argv[5])
```

A partir de cette adresse, le moteur de calcul récupère la *longitude* et *latitude* du jardin à l'aide du géocodeur *Nominatim* (de la bibliothèque *geopy*) qui, une fois initialisé, convertis l'adresse en coordonnées GPS. En effet, ces données sont nécessaires pour obtenir des données météo précises pour un emplacement. Si l'adresse n'est pas trouvée, le script affiche un message d'erreur et s'arrête.

```
from geopy.geocoders import Nominatim

# Initialiser le géocodeur Nominatim
geolocator = Nominatim(user_agent="my_unique_app_name")

# Géocodage pour obtenir la latitude et la longitude
location = geolocator.geocode(adresse)
```

À l'aide des coordonnées obtenues, un objet *Point* est créé pour représenter la localisation. La période de données météo est définie pour toute l'année 2023, avec *start* et *end* spécifiant les dates de début et de fin.

```
from datetime import datetime
from meteostat import Point, Hourly

# Définir la localisation
point = Point(location.latitude, location.longitude)
```

```
# Définir la période pour les données historiques
start = datetime(2023, 1, 1)
end = datetime(2023, 12, 31)
```

Les données météorologiques horaires sont ensuite téléchargées pour la période spécifiée. Les températures sont ensuite filtrées pour récupérer uniquement : Les températures à midi (12h) et les températures nocturnes à 4h du matin. Ces températures ont été entré à la main dans la base de donnée après de long travaux de recherches étant donné qu'il était compliqué de trouver une base de donnée toute faite gratuite.

```
import pandas as pd
```

```
# Filtrer les données pour obtenir uniquement les températures à 12h
 #(midi) et à 4h du matin
temps_at_noon = data[data.index.hour == 12]['temp']
temps_at_4am = data[data.index.hour == 4]['temp']
```

Les températures sont regroupées par mois et moyennées, créant deux séries :
temps_monthly_avg_at_noon : température moyenne mensuelle à midi et
temps_monthly_avg_at_4am : température moyenne mensuelle à 4h du matin.

```
import matplotlib.pyplot as plt
```

```
# Calculer la température moyenne pour chaque mois à 12h et à 4h du matin
temps_monthly_avg_at_noon = temps_at_noon.resample('M').mean()
temps_monthly_avg_at_4am = temps_at_4am.resample('M').mean()
```

Le programme analyse ensuite quels mois remplissent les conditions de températures spécifiées. Pour la journée ; mois dont la température moyenne à midi est comprise entre *temp_min_midi* et *temp_max_midi*. Pour la nuit ; mois dont la température moyenne à 4h du matin est comprise entre *temp_min_nuit* et *temp_max_nuit*. Il effectue ensuite une intersection entre les mois des deux plages pour déterminer les mois satisfaisant les deux critères.

```
# Préparer les données pour le diagramme à barres
months = temps_monthly_avg_at_noon.index.strftime('%B')
# Obtenir les noms des mois
temps_values_noon = temps_monthly_avg_at_noon.values
# Valeurs des températures à midi
temps_values_4am = temps_monthly_avg_at_4am.values
# Valeurs des températures à 4h du matin

# Trouver les mois qui sont dans la plage donnée pour midi et pour la nuit
months_in_range_noon = months[(temps_values_noon >= temp_min_midi) &
(temps_values_noon <= temp_max_midi)]
```

```

months_in_range_4am = months[(temps_values_4am >= temp_min_nuit) &
(temps_values_4am <= temp_max_nuit)]

# Trouver les mois qui répondent aux deux critères
months_in_range_both = set(months_in_range_noon).intersection(
set(months_in_range_4am))

```

Le script renvoie une chaîne de caractères contenant la liste des mois durant lesquels les températures moyennes, à la fois à midi et à 4h du matin, sont comprises dans les plages spécifiées. Si aucun mois ne satisfait les deux critères, un message indique qu'aucun mois correspondant n'a été trouvé. Cette chaîne de caractère obtenue est ajoutée à la table `compo_jardin` en tant qu'attribue `prevision_plantation`.

```

if len(months_in_range_both) > 0:
    print(", ".join(months_in_range_both))
else:
    print("Aucun mois trouvé avec des températures moyennes à midi et à \
4h du matin dans les plages données.")

```

En résumé, ce moteur utilise des données météo horaires et des plages de températures spécifiques pour identifier les mois les plus favorables à la plantation, en prenant en compte les températures moyennes de jour et de nuit.

4 Sécurisation de l'API

4.1 Gestion et sécurité des routes

4.1.1 Techniques utilisées : Vérification des Rôles et Sessions

Pour sécuriser les accès aux différentes routes de l'application, nous avons implémenté des middlewares qui vérifient les rôles des utilisateurs et la validité de leur session. Ces vérifications permettent de s'assurer que seuls les utilisateurs autorisés peuvent accéder à certaines fonctionnalités. Voici des exemples de notre implémentation :

```
// Middleware pour vérifier si l'utilisateur est un administrateur
function checkAdmin(req, res, next) {
  if (!req.session.user) {
    res.redirect('/');
  } else {
    adminModel.isAdmin(req.session.user)
      .then(isAdmin => {
        if (isAdmin) {
          next();
        } else {
          res.redirect('/');
        }
      })
      .catch(err => {
        console.error('Erreur lors de la vérification du statut
d\'administrateur:', err);
        res.status(500).send('Erreur lors de la vérification du statut
d\'administrateur');
      });
  }
}
```

4.2 Cryptage des Mots de Passe

4.2.1 Technique utilisée : Middleware bcrypt

Pour protéger les mots de passe des utilisateurs, nous utilisons le middleware bcrypt pour les hacher avant de les stocker dans la base de données. Le hachage rend les mots de passe illisibles et sécurise les informations sensibles en cas de compromission de la base de données.

4.3 Prévention des Attaques par Injection

4.3.1 Technique utilisée : Requêtes Paramétrées

Les attaques par injection SQL représentent une menace importante pour les applications web, permettant à des attaquants d'exécuter des commandes SQL arbitraires. Pour prévenir ces attaques, nous avons utilisé des requêtes paramétrées, qui séparent les données des commandes SQL. Voici un exemple de notre implémentation :

```

create: function(nom, prenom, mail, mdp) {
  return new Promise((resolve, reject) => {
    const sql = "INSERT INTO Utilisateur (nom, prenom, mail, mdp, date_crea)
    VALUES ($1, $2, $3, $4, NOW()) RETURNING id";
    console.log("SQL query:", sql);
    db.query(sql, [nom, prenom, mail, mdp], (err, result) => {
      if (err) {
        console.error("Erreur lors de la création de l'utilisateur :", err);
        reject(err);
      } else {
        // Récupérer l'ID de la ligne créée
        const userId = result.rows[0].id;
        resolve(userId);
      }
    });
  });
}

```

5 Ressentis et Axes d'amélioration

5.1 Ressentis

Ce projet a été à la fois intéressant et instructif. Travailler sur une API de gestion de potager nous a permis d'explorer divers aspects techniques, de l'architecture logicielle à la manipulation des données météo, tout en passant par la conception de la base de donnée et tout intégrant des calculs prévisionnels pour optimiser la gestion d'un jardin. Nous avons également pu expérimenter avec des bibliothèques de géolocalisation et d'analyse de données, ce qui a grandement enrichi notre compréhension de ces technologies.

Cependant, nous avons rencontré plusieurs défis au niveau de l'organisation et de la gestion du projet. La planification initiale a rapidement été dépassée, ce qui nous a contraints à réajuster constamment nos objectifs. Les réunions et la coordination n'ont pas toujours été optimales, ce qui a contribué à des retards dans l'avancement du projet. En conséquence, ce rapport est rendu avec quelques mois de retard par rapport au planning initial, ce qui souligne les difficultés que nous avons eues à gérer efficacement le temps et les priorités.

5.2 Axes d'amélioration

Pour améliorer notre projet d'un point de vu technique nous pouvons transférer la base de donnée dans un **cloud** afin de permettre à n'importe quel utilisateur d'utiliser l'API depuis n'importe quel endroit. De plus, nous pouvons encore **améliorer notre moteur de calcul prévisionnel** en prenant en compte plus de paramètre tels que la qualité des sols, l'humidité, la pluviométrie, l'ensoleillement, etc...afin de le rendre plus précis. Conscient que certains paramètres restent difficile à collecter à distance, une **solution IOT** peut également être introduit afin de collecter tous ces paramètres à l'aide de capteurs nécessitant cependant une mise en place physique.

Pour améliorer cette fois ci la gestion de notre projet et même la celle des projets futurs, nous pourrions tirer parti des leçons apprises ici et mettre en place plusieurs changements :

- **Meilleure planification et définition des objectifs** : Prendre le temps de définir des objectifs clairs dès le début et les diviser en étapes réalisables. Une planification en sprints (par exemple, inspirée des méthodes agiles) pourrait nous permettre de mieux suivre notre progression et d'anticiper les retards.
- **Utilisation d'outils de gestion de projet** : L'adoption d'outils collaboratifs, aurait pu améliorer la transparence et la communication entre les membres de l'équipe. Ces outils offrent des fonctionnalités de suivi de tâches et facilitent la répartition du travail.
- **Rythme de réunions régulières** : Organiser des réunions hebdomadaires de mise à jour aurait aidé à assurer que chacun reste aligné sur les priorités et les deadlines. Cela aurait également permis de réagir plus rapidement aux éventuels problèmes rencontrés.
- **Suivi de l'avancement et des retards** : En suivant de manière rigoureuse l'avancement des tâches et en identifiant rapidement les retards, nous aurions pu anticiper les ajustements nécessaires pour limiter l'accumulation de retard.

6 Conclusion

Ce projet de développement d'une API de gestion de potager nous a permis d'acquérir des compétences techniques variées, notamment en conception de bases de données, développement d'API, géolocalisation et intégration de données météorologiques. En combinant ces éléments, nous avons créé un outil qui conseille les utilisateurs sur les périodes optimales de plantation selon leur localisation et les conditions climatiques.

Bien que nous ayons rencontré des défis dans la gestion du projet, aboutissant à des retards significatifs, cette expérience nous a offert des leçons précieuses en matière de planification et d'organisation. Les améliorations futures pourraient inclure l'élargissement de la base de données, l'intégration de données météo en temps réel et le développement d'une interface utilisateur plus interactive.

En conclusion, ce projet a été à la fois formateur et stimulant, et nous sommes fiers du produit final, qui a le potentiel de se développer encore pour répondre aux besoins des jardiniers amateurs et passionnés.

7 Annexe - Vidéo Présentation

Vous trouverez ci-dessous un lien vers une overview de l'API et de ses possibilités d'utilisation.

- [PotagerCreator Overview](#)