

# Dossier de Projet

En vue de l'obtention du Titre Professionnel

## Développeur Web et Web mobile



Réalisé par

**Guillaume Debeire**

Centre de formation

**O'clock**

# Remerciements

2020 fût une année pleine de rebondissements. Elle a été le terrain d'une grande réflexion pour moi, qui s'est conclue par un projet de réorientation professionnelle.

Ce projet a pris la forme de O'clock, chez qui j'ai pu apprendre le développement web et grâce à qui j'ai pu rentrer dans l'univers de l'informatique, univers dans lequel je baigne depuis toujours mais pour lequel je ne me suis jamais impliqué professionnellement, jusqu'à aujourd'hui. Mes premiers remerciement vont donc à l'organisme O'clock qui m'a permis de découvrir ce monde.

L'ensemble des enseignants m'a permis d'approcher le développement avec beaucoup d'assurance, l'intensité de la formation et la régularité des exercices m'ont permis de vite renforcer mon expérience et de valider mes connaissances.

# Sommaire

<b>Compétences du référentiel couvertes par le projet</b>	<b>5</b>
Développer la partie front-end d'une application en intégrant les recommandations de sécurité	5
Développer la partie back-end d'une application en intégrant les recommandations de sécurité	5
<b>Résumé du projet</b>	<b>6</b>
<b>Cahier des charges</b>	<b>7</b>
Présentation du projet	7
Définition des besoins et objectifs	7
Public visé	7
Liste des technologies utilisées	8
Fonctionnalités	8
Navigateurs / compatibilités	8
Arborescence de l'application web	9
User Stories	11
<b>Spécifications techniques</b>	<b>12</b>
Versioning	12
Technologies côté front-end	12
Technologies côté back-end	12
<b>La réalisation du projet</b>	<b>14</b>
Introduction	14
L'équipe	14
Gestion du projet	14
Les outils collaboratifs	14
Les Sprints	15
MCD et wireframes	15
Charte graphique	17

Front-end	17
Back-end	18
Installation	18
API	18
Sécurité	19
Services	19
Difficultés rencontrées	19
<b>Réalisations personnelles</b>	<b>21</b>
Mon rôle au quotidien	21
Requête customisée, MVC et Voters	21
CSS Customisé	24
Formulaire d'ajout de plante	25
<b>Présentation du jeu d'essai</b>	<b>27</b>
<b>Veille technologique sur la sécurité</b>	<b>28</b>
<b>Extrait d'un site anglophone et traduction</b>	<b>30</b>
<b>Conclusion</b>	<b>32</b>
<b>Annexes</b>	<b>33</b>
<b>Glossaire</b>	<b>48</b>

# Compétences du référentiel couvertes par le projet

Le projet Plantopia est constitué de 2 applications : un front-office réalisé avec la librairie JavaScript React, et un back-office réalisé par le framework Symfony. Le front-office va consommer la base de données qui est gérée par le back-office qui est une application à part entière, constituée d'une partie back-end et front-end.

## Développer la partie front-end d'une application en intégrant les recommandations de sécurité

### 1. Maquetter une application

La phase de conception nous a permis de mettre en place *wireframes*, MCD et *user stories* (ces documents se retrouvent en Annexe).

### 2. Réaliser une interface utilisateur web statique et adaptable

Nous avons réalisé plusieurs interfaces utilisateur, que ce soit pour le front-office ou le back-office. Celles-ci (principalement pour le front-office) sont adaptables et *responsives* afin de s'adapter au *mobile* comme au *desktop*.

### 3. Développer une interface utilisateur web dynamique

La partie front-end et le front-office ont été réalisées avec la librairie Javascript React afin d'élargir les possibilités en matière d'interface utilisateur et de faire fonctionner l'application en *single-page*.

## Développer la partie back-end d'une application en intégrant les recommandations de sécurité

### 5. Créer une base de données

Pour les besoins du projet, nous avons procédé à la création d'une base de données, suivant le dictionnaire de données et le MCD. Nous avons ensuite réalisé nos tables et colonnes grâce à l'ORM Doctrine.

### 6. Développer les composants d'accès aux données

Pour la consultation des données de la base, nous nous sommes servis de Doctrine, l'ORM (Object Relational Mapping) par défaut de Symfony.

### 7. Développer la partie back-end d'une application web ou web mobile

Plantopia est une application qui dispose d'une partie back-end pour son front-office comme son back-office. Nous avons donc mis en place un MVC pour le back-office, avec un système de controllers de models pour l'affichage et la manipulation des données. Pour le front-office, les données étaient consommées grâce à l'API mise en place.

# Résumé du projet

Plantopia est un projet réalisé durant un mois, dans le cadre de la formation de Développeur Web et Web Mobile chez O'clock, par quatre développeurs n'ayant encore jamais travaillé ensemble mais impatients de mettre en pratique les connaissances rassemblées pendant cinq mois.

Mélanie, notre Product Owner, passionnée de plantes et désireuse de créer, à terme, un shop en ligne ouvert sur la communauté et l'échange, a porté ce projet, rejointe par trois amateurs de végétaux et désireux de découvrir l'alliance de React et de Symfony et l'API.

Il est donc entendu que ce projet est la première pierre d'un ensemble plus large : ce Plantopia V1 a pour vocation de devenir une appli web gratuite, exhaustive, pratique et simple pour les utilisateurs, en ménageant des ouvertures vers des fonctionnalités futures.

Notre projet a ainsi pour but, dans la version qui nous intéresse aujourd'hui, de permettre à un utilisateur de visionner notre bibliothèque des plantes, de créer son compte, d'ajouter ses propres plantes et de renseigner les données personnalisées relatives à leur entretien. Les administrateurs de l'application doivent également avoir accès à une interface dédiée à la gestion des données.

Ce dossier va à présent vous raconter la conception de Plantopia, son déroulement collectif et mes réalisations personnelles, et son avenir.

# Cahier des charges

## Présentation du projet

Plantopia est une application web - mobile first - qui permet à ses utilisateurs de répertorier, afin de gérer plus facilement leurs plantes, en fonction de leurs besoins.

Elle leur permet d'enregistrer leurs plantes à partir d'une base de données puis de paramétriser des alertes afin de répondre aux besoins d'arrosage, de fertilisation ou de rempotage des plantes de l'utilisateur.

## Définition des besoins et objectifs

La naissance du projet est partie d'une problématique : chaque plante a des besoins différents. Et mémoriser les besoins de chacune peut s'avérer compliqué. Il existe des applications permettant de les enregistrer et de les répertorier sous la forme d'une to-do list mais nous voulions en créer une sans publicité, et disposant d'une interface claire et attractive.

Notre objectif était donc de créer Plantopia, une application permettant de programmer des alertes personnelles pour chaque besoins de chacune de nos plantes, les partager avec la communauté, et favoriser l'échange de plantes et boutures entre membres de la communauté.

## Public visé

Pour cette application, le public visé est constitué de tous les propriétaires de plantes, du plus débutant au plus expérimenté.

Pour ce faire, un des objectifs primaires du développement va être l'accessibilité, afin de rendre l'application la plus ouverte et agréable possible.

Avec un nombre de plantes qui augmente dans nos intérieurs il est indispensable d'avoir une appli qui permette d'avoir des rappels afin de ne pas s'y perdre.

# Technologies utilisées

## Back en Symfony

- Base de données MySQL gérée avec Adminer
- ORM Doctrine
- Identification avec JWT
- Back office avec twig et CSS

## Front en React

- Redux
- Requêtes API avec Axios
- Routage avec React Router

## Fonctionnalités

### MVP :

**En tant que visiteur** : consulter la bibliothèque des plantes (déjà créées en BDD)

**En tant qu'utilisateur** :

- Lister et gérer ses plantes (CRUD)
- Paramétrier des rappels/alertes

**En tant qu'administrateur** : gérer la bibliothèque des plantes et les users

### Améliorations :

- Plateforme d'échanges de plantes avec chat
- Reconnaissance d'une plante à partir d'un upload d'une photo

## Navigateurs / Compatibilités

Pour le bon fonctionnement de l'application pour le plus d'utilisateurs possibles, l'application sera compatible avec Chrome - Firefox - Edge - Safari – Opéra.

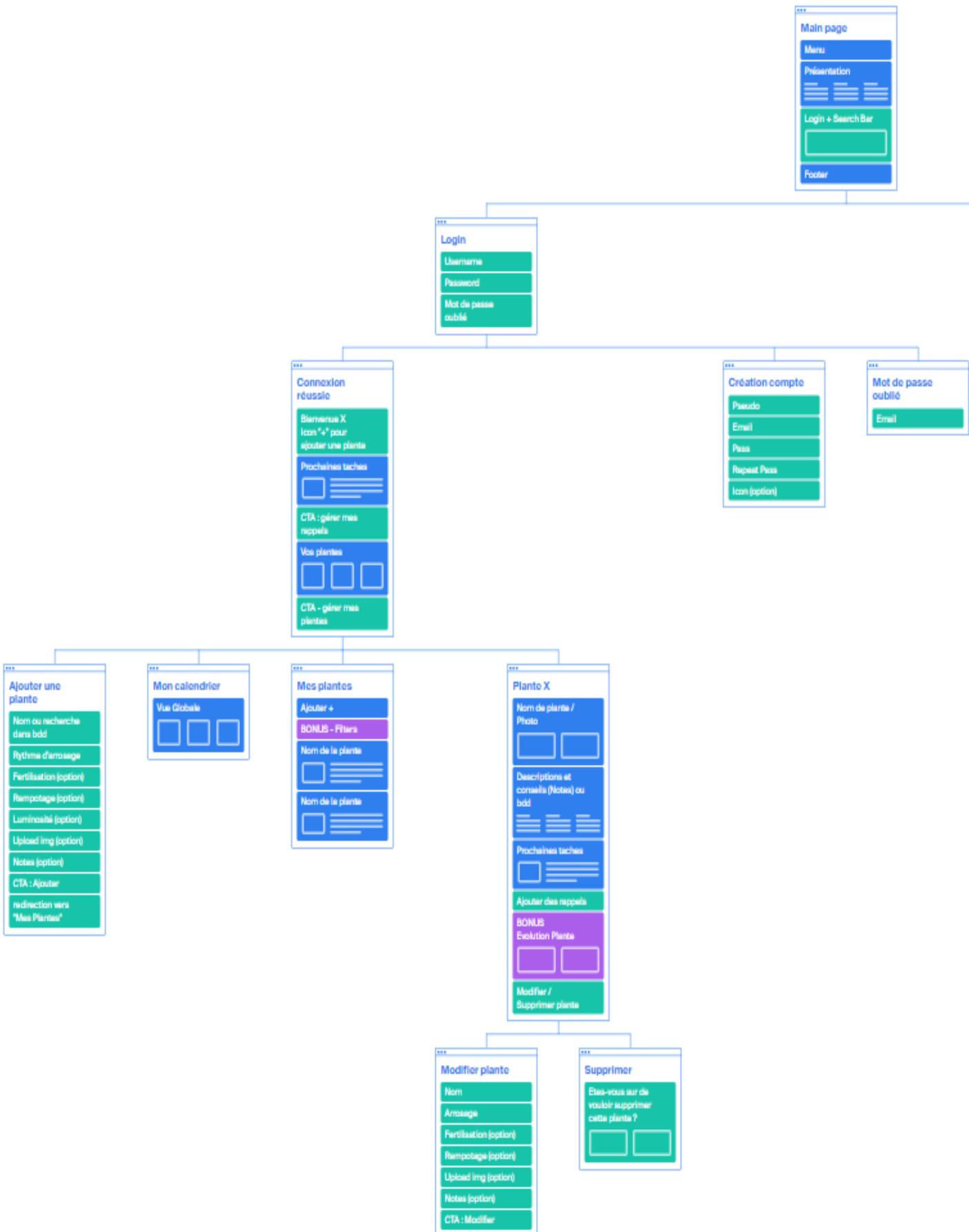
## Liste des rôles

**Mélanie:** Owner, web designer

**Yanis:** Lead dev front

**Guillaume:** Lead dev back, Git master

**Christophe:** Scrum master





# User Stories

## En tant que visiteur

Je veux pouvoir accéder à une page d'accueil référent plusieurs exemples de plantes et un résumé de leur entretien.

Je veux pouvoir accéder aux détails d'une plante spécifique, comprenant son nom, rythme d'arrosage, de fertilisation et de rempotage, et des images.

## En tant qu'utilisateur connecté

Je veux pouvoir accéder à une page répertoriant l'ensemble de mes plantes et quand les arroser.

Je veux pouvoir créer une nouvelle plante pour ma liste.

Je veux pouvoir ajouter une plante à ma liste depuis la base de données.

Je veux pouvoir indiquer le rythme d'arrosage de mes plantes (obligatoire).

Je veux pouvoir indiquer le rythme de fertilisation et de rempotage (optionnel).

Je veux pouvoir ajouter des images à ma plante (optionnel).

Je veux pouvoir accéder aux détails d'une de mes plantes, avec des images et un calendrier qui décrit quand l'arroser/fertiliser/rempoter.

Je veux pouvoir informer quand j'ai effectué un(e) arrosage/fertilisation/rempotage d'une plante.

Je veux pouvoir modifier les informations ci-dessus.

Je veux pouvoir activer une alerte (email) me rappelant d'arroser/fertiliser/rempoter.

## En tant qu'administrateur

Je veux pouvoir accéder à une page listant l'ensemble des utilisateurs de la plateforme.

Je veux pouvoir accéder aux détails d'un utilisateur.

Je veux pouvoir accéder à une page listant l'ensemble des plantes de la base de données.

Je veux pouvoir accéder aux détails d'une plante (nom, photo, rythme d'arrosage/fertilisation/rempotage) et les modifier.

# Spécifications Techniques

## Versioning

Afin d'éviter les conflits, de bien avoir le contrôle sur notre code et de pas se mélanger les pinceaux, nous avons décidé de travailler sur deux repositories différents, un pour la partie front-end et React, et un pour la partie back-end et Symfony. Chaque repositories travaillaient de la manière suivante :

Une branche Master servait de base et de version stable et fonctionnelle de l'application. Nous poussions notre travail sur cette branche en fin de sprint, lorsque nous avions fini de travailler et de tester le bon fonctionnement d'une fonctionnalité.

La branche du Sprint courant servait de base pendant le sprint concerné. A chaque nouvel apport au code, à partir du moment que cet apport était vérifié et conservait la stabilité de l'application, notre travail était sauvegardé sur cette branche Sprint.

Enfin, pour chaque fonctionnalité à coder et à tester, une branche portant le nom de la fonctionnalité était créée.

## Technologies côté front-end

Nous avons utilisé la librairie JavaScript React afin de concevoir la partie front-end de l'application.

Redux : la librairie Redux nous a permis l'enregistrement des informations de l'utilisateur pour faciliter la connexion et la transmission du token.

Axios : la librairie Axios nous a permis d'effectuer des requêtes vers l'API mise en place en back-office.

Sass et SCSS : Nous avons utilisé le SCSS pour l'apparence de l'application côté front.

Babel : Nous nous sommes servis de Babel pour la compilation des dépendances.

React Router : la gestion des routes et de la redirection s'est faite grâce à React Router.

## Technologies côté back-end

Je me suis servi du framework PHP Symfony pour concevoir la partie back-end de l'application, ainsi que le back-office qui nous a permis un accès plus pratique à la base de données.

MySQL : Nous avons utilisé MySQL pour créer la base de données.

**API REST** : Nous avons rendus accessibles la base de données pour la partie front-end via une API respectant les contraintes de l'architecture REST (*Representational State Transfer* ou Etat de Transfert Représentationnel).

**Lexik JWT** : Pour l'authentification des utilisateurs nous avons installé le composant d'authentification Lexik JWT (JSON Web Token).

**Twig** : Nous nous sommes servis de Twig, un moteur de template pour PHP intégré à Symfony, pour l'affichage du back-office. Twig permet le rendu de code HTML en plus d'une longue liste de tags, filtres et fonctions qui permettent de créer des conditions, des boucles, des variables.

**ORM Doctrine** : Doctrine est l'ORM (Object-Relational Mapping) par défaut utilisé par Symfony.

**Composer** : Un gestionnaire de dépendances qui permet l'installation de bibliothèques requises par le projet.

# La réalisation du projet

## Introduction

Plantopia est un projet de fin de formation chez O'clock. Celle qui fût à l'origine de cette idée est Mélanie Riché, qui a été le *product owner* du projet. Etant elle-même une grande amatrice de plantes à entretenir à la maison, le besoin de façonner une application de gestions des plantes personnelles est venu très naturellement.

## L'équipe

Mélanie, Yanis, Christophe et moi sommes des membres de la promotion Marty. Mélanie et Yanis se sont spécialisés dans React, Christophe et moi avons fait la spécialisation Symfony. Nous avons effectué la répartition des rôles suivante :

Mélanie : Product Owner/ Front.

Yanis : Lead Developper Front.

Christophe : Scrum Master/ Back.

Guillaume : Lead Deavelopper Back, Git Master.

## Gestion du Projet

### Les outils collaboratifs

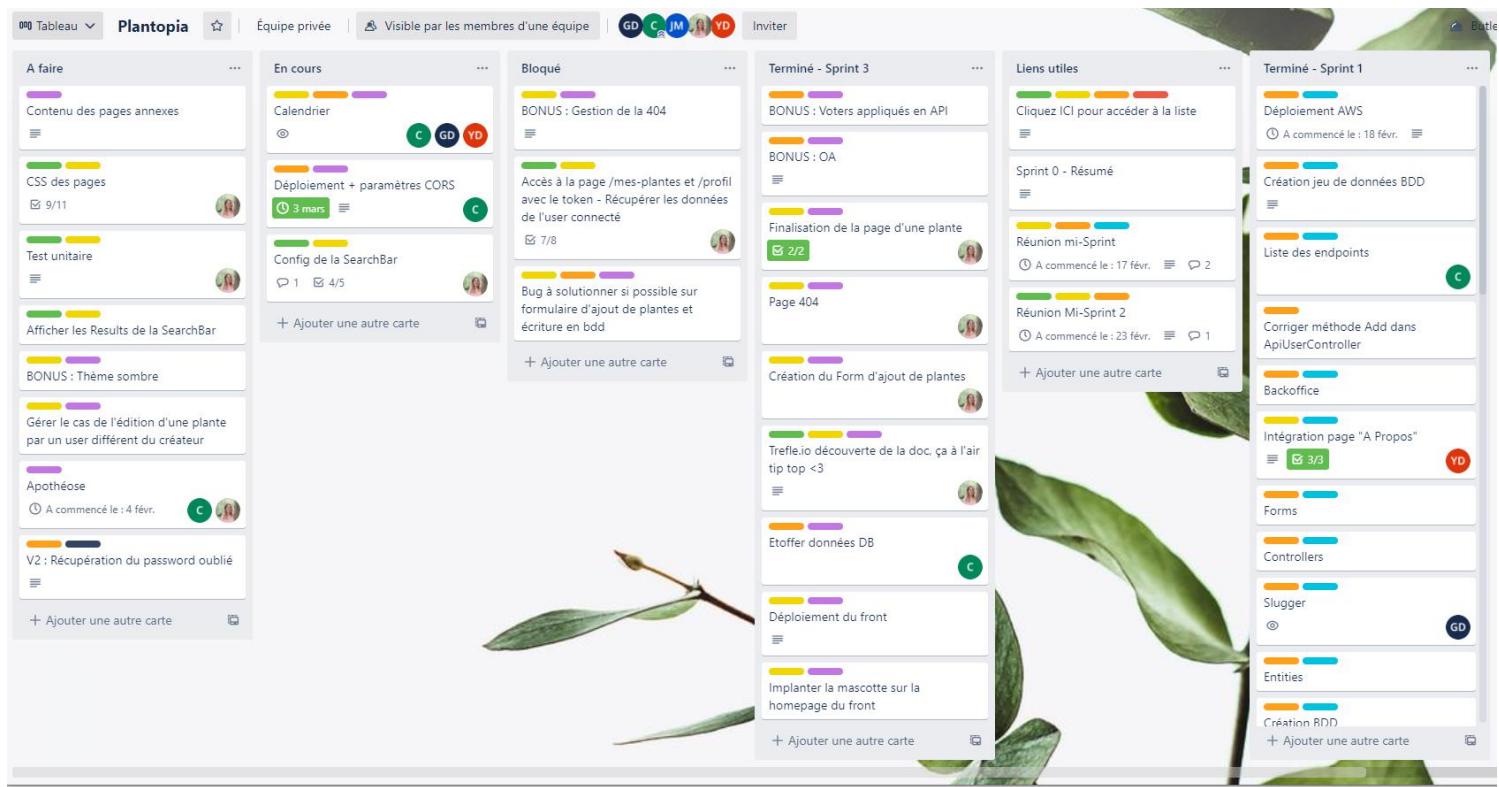
Etant donné que nous travaillions tous les quatre depuis chez nous, il était indispensable que soyons équipés d'outils de collaboration afin de s'assurer de la bonne communication entre les membres de l'équipe et entre la partie front-end et la partie back-end.

Pour le développement, nous avons utilisé Visual Studio Code qui nous donne la possibilité de travailler en *pair programing* (plusieurs utilisateurs sur un même code)

Nous avions donc à notre disposition Slack, permettant la communication entre tous les membres de l'équipe. Nous avions également accès à un Google Drive, répertoriant les divers documents nécessaires au bon déroulement du projet (Cahier des charges, charte graphique, wireframes...)

Nous avions également accès à un Trello répertoriant toutes les user stories, les fonctionnalités, et les besoins de l'application. Nous avons organisé le Trello en 4 blocs principaux :

A faire, En cours, Bloqué et Terminé.



### *Le Trello de Plantopia avec ses différentes colonnes.*

Afin d'assurer les meetings quotidiens, et pour l'échange vocal et vidéo, nous avons utilisé Discord durant l'intégralité du développement.

Le partage de documents, comme le cahier des charges, les logos, la charte graphique, etc. s'est faite grâce à Google Drive.

Pour le versioning, nos projets, séparés en deux repository distincts, un pour la partie front-end et un pour la partie back-end, sont stockés sur GitHub, accessibles à tous les membres de l'équipe depuis n'importe quel poste, et assurant avec son système de branches la bonne cohésion du projet.

## **Les Sprints**

Nous avons organisé le développement de l'application autour de 4 sprints :

Sprint 0 : Structuration de l'architecture de l'application, à travers l'établissement du cahier des charges, la réalisation des wireframes, user stories et MCD.

Sprint 1 : Installation du Framework Symfony et mise en place de la librairie React. Création du back-office, de la base de données.

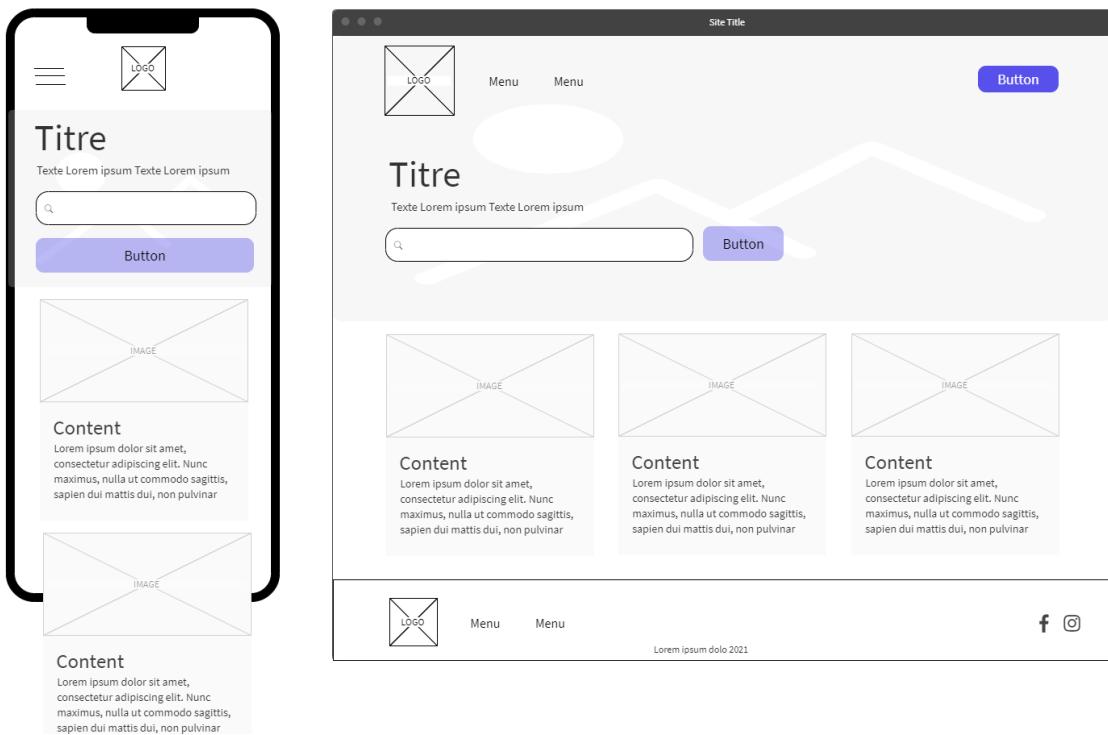
Sprint 2 : Développement des principales fonctionnalités, mise en place du système de tokens et de sécurité principale de l'application.

Sprint 3 : finalisation des dernières fonctionnalités, vérifications et tests unitaires. Déploiement et démonstration de l'application.

# Wireframes et MCD

## Les Wireframes

Notre application étant mobile-first, nous avons conçu les wireframes de la version mobile en priorité. Nous avons également réalisé des wireframes en version desktop, en un deuxième temps.



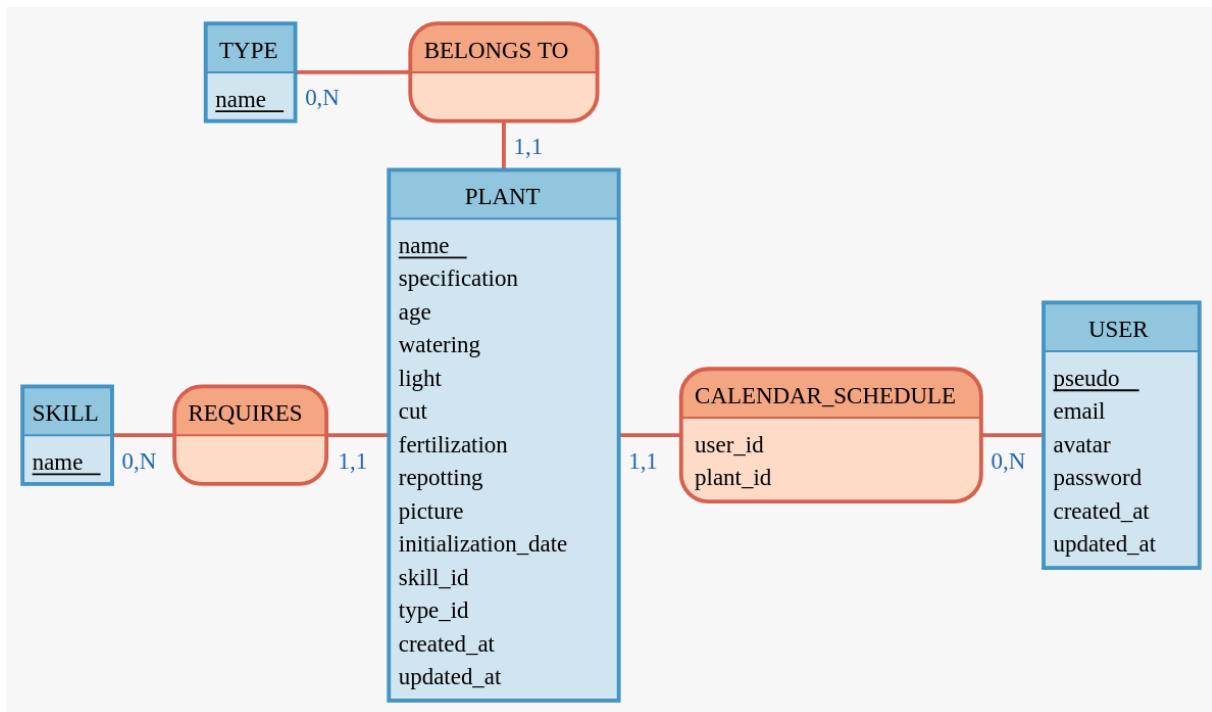
Wireframe de la page d'accueil de l'application.

## Le MCD

Notre application étant centrée sur les plantes, notre **MCD** (Modèle Conceptuel de Données) est doté de peu de tables mais d'une table Plant, en relation avec toutes les autres tables, comprenant de nombreuses colonnes. Chaque plante appartient à un utilisateur. Afin de personnaliser les plantes de chacun, par exemple pour y ajouter une photo, nous avons fait en sorte qu'ajouter une plante depuis la bibliothèque créé une nouvelle plante dans la base de données. Ainsi, chaque plante est unique.

Ainsi il est possible de récupérer les informations de Type ou de Skill pour une plante donnée en passant par la propriété type\_id et skill\_id, et la table intermédiaire Calendar\_Schedule nous permet de récupérer l'utilisateur propriétaire d'une plante à travers ses propriétés user\_id et plant\_id. Cette table intermédiaire sert également et surtout à créer les calendriers personnels de chaque utilisateur.

Chaque propriété de la table Plant est renseignée par l'utilisateur sous la forme d'un formulaire lors de l'ajout de la plante à sa bibliothèque de plantes personnelles.



*MCD v8, centralisant la table PLANT.*

## Charte Graphique

**Code Couleurs :**

Beige :	Marron :	Orange :
HEX #EBE3DC	HEX #824427	HEX #D06D13
RGB 235, 227, 220	RGB 130, 68, 39	RGB 208, 109, 19
HSL 28, 6%, 89%	HSL 19, 70%, 33%	HSL 29, 91%, 45%
Vert clair :	Vert foncé :	
HEX #4C724A	HEX #234523	
RGB 76, 114, 74	RGB 35, 69, 35	
HSL 117, 35%, 37%	HSL 120, 49%, 20%	

## Front-end

Pour la partie front-end, nous nous sommes servis de la librairie JavaScript **React**.

Celle-ci nous a permis une connexion efficace à la base de données grâce aux middlewares et en particulier à **Axios**, une librairie permettant la liaison entre le front et le back.

Grâce aux requêtes **Axios**, il est possible de récupérer le *token* transmis lors de la connexion à la route *login\_check*, mise en place pour vérifier l'authenticité des identifiants d'un utilisateur. Lors de sa connexion, les informations de l'utilisateur sont stockées dans le *state* grâce à la librairie Redux qui vient compléter la partie front. Celle-ci permet d'enregistrer les informations et de les récupérer (comme le token) afin de les donner à chaque nouvelle requête.

## Back-end & Back-office

### Installation

Tout a commencé par la mise en place du Framework Symfony. Celui-ci nous a permis de rapidement installer la structure MVC du back-office, avec des controllers servant spécifiquement à l'affichage et la manipulation des données en back-office.

L'architecture de notre application fonctionne autour de la structure MVC (Model-View-Controller), où les Models (ou dans le cadre du framework Symfony, les entités) représentent les tables de la base de données, les divers controllers servent à manipuler la base de données afin d'en afficher le contenu dans les Views.

Pour la création de la base de données, nous avons utilisé Doctrine, l'ORM par défaut de Symfony. Nous avions donc très rapidement mis en place la base de données avec un accès rapide et pratique grâce au back-office dont le CSS a entièrement été fait à la main (voir détails dans mes réalisations personnelles).

Nous avons donc ensuite procédé à la création des Entités (les Models de Symfony) et des templates HTML grâce à Twig, le moteur de templates de Symfony, en suivant le cahier des charges et le dictionnaire de données (cf : Annexe).

Nous avons ensuite mis en place les formulaires d'ajout ou de modifications de plantes et d'utilisateurs, toujours en back-office. En effet, le framework Symfony intègre un système de formulaires modifiable permettant de récupérer des données depuis le back-office et de les persister sur notre base de données.

La base de données permet à la fois de stocker des plantes pré-enregistrées qui pourront être ajoutées par les utilisateurs, et celles directement enregistrées par les utilisateurs.

## API

L'un des défis de cette application était le transfert des données de la base au front-office via une API. Nous avons pour cela mis en place des controllers spécifiques à l'API. Ceux-ci, comme pour le back-office, utilisent la méthode BREAD – accronyme pour Browse, Read, Edit, Add, Delete. Cette méthode permet l'accès à de vastes bases de données en listant, détaillant, modifiant, ajoutant et supprimant des données.

J'ai effectués des tests de connexion API avec la plateforme Insomnia qui permet de tester les différentes routes que j'ai programmées. Cette plateforme nous a principalement permis de tester les *endpoints* de notre API (cf : Annexe) ainsi que la sécurité avec la bonne transmission des *tokens*.

## Sécurité

Pour ce qui concerne la sécurité, nous avons mis en place un système d'authentification à l'entrée de l'application, que ce soit côté front-end ou côté back-end. Aidés par le bundle Lexik JWT Token, permettant la création et la passation,

dans le header d'une requête, du token d'identification permettant d'authentifier les utilisateurs se connectant à la plateforme, nous avons pu assurer la bonne sécurité de l'application.

Pour renforcer cette sécurité nous avons mis en place des *voters* customisés qui vont aller vérifier la légitimité d'une requête afin de sécuriser l'accès aux données.

## Services

Nous avons, pour les besoins spécifiques de l'application, mis en place des services « faits maison » qui nous ont permis de nous débloquer sur plusieurs points (cf : difficultés rencontrées).

L'un d'entre eux est l'*IntervalStringer* (cf : Annexe), qui, comme son nom l'indique, sert à lire sous forme de chaîne de caractère un objet de la classe `DateInterval`, une classe native de PHP qui sert à stocker un nombre fixe de durées (en années, mois, jours, heures, etc.)

# Difficultés rencontrées

## Relation back-front et Tokens

Un des points qui nous a présenté le plus de difficultés est la relation entre la partie back-end et front-end. La bonne transmission du token s'est avérée compliquée pour nous et nous nous sommes souvent retrouvés avec des messages d'erreur « *Invalid Credentials* » ou « *Token not found* » durant le développement de l'application.

L'origine de ces problèmes venait du fait que le back a avancé plus vite que le front. Ce qui nous (le back) a poussé à mettre en place des éléments de sécurité comme JWT ou CORS trop tôt et a géné le front lorsqu'il a tenté les premières connexions avec Axios, car il ne savait pas si les erreurs étaient normales dû à la sécurité mise en place, ou si elles venaient du code.

Heureusement, nous avons initié l'authentification et les tokens très tôt dans le développement, ce qui nous a permis de nous attarder dessus en sprint 1, et lors du 2<sup>ème</sup> sprint nous avions réglé tous les problèmes persistants.

Nous avons effectué nos tests de connexion grâce à Insomnia, qui nous a permis de vérifier que nous récupérions bien le token lors de la connexion à la route `login_check`, permettant de vérifier l'authenticité des identifiants de l'utilisateur. Après modification de la syntaxe des requêtes Axios côté front, nous avons pu débloquer la situation.

## Interprétation des données

L'interprétation des données, et particulièrement des DateInterval(), mis en place pour l'arrosage des plantes et permettant d'obtenir un intervalle de dates, nous a également posé problème.

Cette donnée est cruciale pour l'affichage du Calendrier personnel d'un utilisateur, lui indiquant quand arroser ses plantes, les fertiliser, etc.

Nous avons donc bien pu récupérer les données de DateInterval, mais l'interprétation de ces données pour l'affichage dans le calendrier s'est avérée plus problématique. Nous pensons que la façon dont nous avons stocké les informations des utilisateurs et de leurs plantes, à travers le table Calendar\_Schedule, nous a posé plus de problèmes que vraiment apporté des solutions. Nous envisageons pour une v2 de revoir la construction de la base de données et des tables, afin de faciliter le passage d'information et l'affichage de données côté front.

# Réalisations personnelles

## Mon rôle au quotidien

En tant que Git Master, mon rôle était de bien m'assurer de la bonne tenue des versions du projet, que chaque membre fasse bien ses *commits*, et que des branches soient créées pour chaque nouvelle fonctionnalité de l'application.

Pour ce faire, nous avons utilisé GitHub. La partie front et la partie back sont deux projets différents, reliés par une API. En tant que Git Master je devais donc m'assurer de la bonne utilisation des *commits* de l'équipe et que le *versioning* se passe bien.

Chaque projet était centré autour de la branche Master. Celle-ci sert de base en début de sprint, et on y centralise tout en fin de sprint, une fois la stabilité de la version assurée. Au cours du sprint, notre travail était enregistré sur la branche du sprint en cours. En fin de validation de chaque nouvelle fonctionnalité, ou après avoir débuggé ou amélioré une fonctionnalité, un enregistrement était fait sur cette branche.

Pour le développement de chaque fonctionnalité nouvelle branche était créée.

Mon rôle était donc de bien m'assurer que les membres de l'équipe respectent cette organisation, et ne soient jamais en train de travailler sur une branche sur laquelle ils ne devraient pas être.

## Requête personnalisée, MVC et Voters

Pour une meilleure gestion du back-office et un accès optimisé et instinctif aux données, j'ai effectué des requêtes personnalisées dans le UserRepository.php, fichier PHP qui permet de nous mettre en relation avec la base de données par le biais de requêtes, afin de pouvoir accéder aux plantes d'un utilisateur directement depuis son profil.

Pour ce faire, je suis passé par la table intermédiaire CalendarSchedule de la base de données, qui répertorie les plantes par leur identifiant en leur attribuant l'identifiant d'un utilisateur.

Le framework Symfony permet une exploitation simple de la base de données en utilisant le **DQL** (Doctrine Query Language) qui est très similaire au **SQL** (Structured Query Language) mis à part qu'il permet de référencer des **objets PHP** (ex : App\Entity\Plant). J'ai donc utilisé ce langage pour effectuer mes requêtes. L'une d'entre elle permet de prendre en argument de fonction l'identifiant de l'utilisateur pour qui on cherche à récupérer les plantes. Elle va ensuite faire une recherche dans la base de données en se référant à l'identifiant fourni.

```

1  public function findPlantByUser($id)
2  {
3      $entityManager = $this->getEntityManager();
4      $query = $entityManager->createQuery(
5          "SELECT p
6          FROM App\Entity\Plant p
7          JOIN App\Entity\CalendarSchedule c
8          WITH c.id = p.calendarSchedule
9          JOIN App\Entity\User u
10         WITH u.id = c.user
11         WHERE u.id = :id"
12     );
13     $query->setParameter(':id', $id);
14
15     return $query->getArrayResult();
16 }

```

*UserRepository.php.*

Une fois la requête définie dans le UserRepository.php, j'ai pu fournir la requête customisée au UserController.php. Celui-ci va pouvoir, en se fournissant de l'identifiant de l'utilisateur enregistré en session, récupérer les informations de l'utilisateur ainsi que de toutes ses plantes. Afin de s'assurer de la bonne sécurité de notre application, nous avons également pour ce controller mis en place des *Voters* qui vont empêcher l'accès aux détails d'un utilisateur si vous n'en avez pas le droit.

```

1  /**
2  * @Route("/{id}", name="user_show", methods={"GET"})
3  */
4  public function show(User $user, UserRepository $userRepo): Response
5  {
6      $userId = $user->getId();
7      $plants = $userRepo->findPlantByUser($userId);
8      $this->denyAccessUnlessGranted(UserVoter::USER_READ, $user);
9      return $this->render('back/user/show.html.twig', [
10          'user' => $user,
11          'plants' => $plants
12      ]);
13 }

```

*UserController.php*

Les voters sont un des meilleurs moyens de Symfony de vérifier les permissions d'un utilisateur. Ils permettent de centraliser toutes les logiques de permissions de l'application, et de les réutiliser dans de nombreux endroits. Ils peuvent être utilisés de plusieurs manières afin d'avoir le contrôle sur les différents accès aux différents utilisateurs de notre application.

Pour notre application, nous avons configuré les *voters* de telle manière qu'un utilisateur est la seule personne qui puisse accéder aux informations de son compte, en dehors des administrateurs. Pour ce faire, j'ai établi une fonction *VoteOnAttribute* qui va en un premier temps vérifier que l'utilisateur est bien un utilisateur connecté et non anonyme. Ensuite, à l'aide d'un switch – instruction qui équivaut à une série de if – J'ai pu établir des règles à suivre selon la lecture, l'édition ou la déletion d'un utilisateur.

Pour chaque case (possibilité) du *switch*, la fonction ira vérifier si l'utilisateur qui cherche à accéder à la ressource est bien administrateur, puis s'il est l'utilisateur concerné par la requête. S'il est l'un ou l'autre, il aura accès à la ressource. Sinon, l'accès lui sera refusé.

```
● ● ●
1 protected function voteOnAttribute($attribute, $subject, TokenInterface $token)
2 {
3     /** @var User $user */
4     $user = $token->getUser();
5     // if the user is anonymous, do not grant access
6     if (!$user instanceof UserInterface) {
7         return false;
8     }
9     switch ($attribute) {
10         case 'USER_READ':
11             if (in_array('ROLE_ADMIN', $user->getRoles()))
12             {
13                 return true;
14             }
15             if ($user == $subject)
16             {
17                 return true;
18             }
19         case 'USER_EDIT':
20             if (in_array('ROLE_ADMIN', $user->getRoles()))
21             {
22                 return true;
23             }
24             if ($user == $subject)
25             {
26                 return true;
27             }
28         case 'USER_DELETE':
29             if (in_array('ROLE_ADMIN', $user->getRoles()))
30             {
31                 return true;
32             }
33             if ($user == $subject)
34             {
35                 return true;
36             }
37         break;
38     }
39     return false;
40 }
```

Extrait de *UserVoter.php*.

Une fois que le Voter a accordé ou refusé l'accès à la ressource, le controller peut ensuite envoyer ces informations à la *View* (vue).

Ayant utilisé ma requête customisée, je peux accéder aux informations de l'utilisateur ainsi qu'à celles de ses plantes grâce aux variables **\$user** et **\$plants**. Il suffit ensuite juste de faire une boucle for dans le fichier Twig User/show.html.twig et nous avons accès aux plantes d'un utilisateur.

```
● ● ●  
1  <tr>  
2      <th>Plantes</th>  
3      <td>{% for plant in plants %}  
4          {% if plant is defined %}  
5              <a class="btnplantindex" href="/plant/{{ plant.id }}">{{ plant.name }}</a>  
6          {% endif %}  
7          {% endfor %}  
8      </td>  
9  </tr>
```

*User/show.html.twig.*

## CSS Customisé & Back-office

Pour un usage plus agréable de la partie back-office de l'application, j'ai mis au point un fichier CSS (*Cascading Style Sheets* en anglais, ou « feuilles de style en cascade ») personnalisé afin que l'affichage des données puisse être le plus adéquat et le plus confortable d'utilisation.

Un exemple d'ensemble de règles mis en place pour le back-office est la customisation du bouton d'ajout d'élément, que ce soit des plantes, des utilisateurs, des catégories de plantes, etc.

```
● ● ●  
1  .element_add {  
2      color: white;  
3      text-decoration: none;  
4      padding: 6px 13px;  
5      box-shadow: 0px 0 0px 3px rgb(73, 141, 73);  
6      font-size: 30px;  
7      border-radius: 30px;  
8      margin: 2em;  
9      background-color: rgb(12, 22, 12);  
10 }  
11 .element_add:hover:after {  
12     content: 'Ajouter un nouvel élément'  
13 }  
14 .element_add:hover {  
15     color: white;  
16     background: rgb(12, 22, 12);  
17 }  
18 }
```

*app.css*



*Au passage de la souris sur le bouton +, le texte se déplie*

Afin de rendre la navigation sur le back-office plus intuitive et rapide, j'ai créé des tableaux interactifs (cf : Annexe) permettant de naviguer entre les différentes entrées de la base, spécialement entre les utilisateurs et les plantes. Ainsi il est possible depuis la fiche d'un utilisateur d'accéder à toutes ses plantes et depuis la fiche d'une plante d'accéder à son propriétaire.

Pour ce faire j'ai utilisé le langage twig, propre à Symfony, facilitant l'ajout de variables au corps HTML de la page. Avec les fonctionnalités de twig telle que le *slice* (permet de raccourcir une chaîne de caractère), *length* (permet de vérifier la longueur d'une chaîne de caractère), ou la boucle *for* (permet de parcourir un tableau), il est possible d'afficher les données sous la forme voulue. En parcourant les différentes variables générées grâce à des requêtes customisées depuis le repository, nous pouvons accéder à toutes les données dont nous avons besoin.

## Formulaire d'ajout de plante

Pour ajouter les plantes des utilisateurs, ainsi que celles depuis le back-office à la base de données j'ai mis en place un formulaire permettant de renseigner toutes les caractéristiques propres à la plante (cf. Annexe).

Celui-ci est composé de plusieurs entrées.

**Nom** : Le simple nom de la plante. Celui-ci est obligatoire pour éviter les plantes sans nom. C'est la seule entrée obligatoire du formulaire.

**Photo** : Une photo de la plante. Nous avons utilisé un service *FileUploader* (cf : Annexe) afin de récupérer et stocker les images envoyées par l'utilisateur. Cette entrée est optionnelle.

**Age** : permet de déterminer l'âge de la plante.

**Date d'initialisation calendrier** : Cette entrée permet l'initialisation du calendrier personnel de l'utilisateur.

**Type de plante** : Le type de la plante. Pour la V1, seuls les types intérieur et extérieur sont disponibles. Nous avons envisagé d'ajouter plus de types pour une future v2, afin d'améliorer la recherche de plantes.

**Niveau de compétence** : Le niveau de compétence requis pour l'entretien de la plante. Les 3 niveaux sont débutant, intermédiaire et expert.

**Rythme d'arrosage** : le rythme d'arrosage de la plante. Avec la classe DateInterval nous avons pu récupérer un intervalle de temps, qu'il est possible de renseigner au calendrier afin d'y afficher toutes les dates d'arrosage.

**Exposition à la lumière** : la quantité de lumière requise au bon développement de la plante. Un simple texte à titre indicatif.

**Période de taille** : la période à laquelle il est le plus adapté de tailler la plante. L'utilisateur peut choisir le(s) mois pendant le(s)quel(s) il devra tailler sa plante.

**Mois de fertilisation** : Même principe que pour la taille de la plante. L'utilisateur peut choisir plusieurs mois pendant lesquels il serait judicieux de fertiliser la plante.

**Mois de rempotage** : Une fois n'est pas coutume, le principe est le même que pour la taille et la fertilisation. L'utilisateur peut choisir plusieurs mois pour répondre aux besoins de sa plante.

**Spécifications de la plante** : Une entrée textuelle, l'utilisateur peut saisir les spécificités de sa plante selon ses envies, le texte est libre et permet la personnalisation des plantes.

La mise en place de ce formulaire a été un des éléments les plus importants du développement de l'application car les plantes d'un utilisateur sont le cœur de l'application. Nous nous devions donc d'y mettre le plus d'éléments possibles afin de pousser la personnalisation le plus loin possible et que cela puisse répondre à tous les besoins de l'utilisateur (et de ses plantes).

Une fois le formulaire envoyé, le controller prend le relais pour récupérer ces informations et les enregistrer en base de données. Celui-ci passe par l'entité Plant pour inscrire les données sur la base. Le controller créé alors une nouvelle entrée dans la table Calendar Schedule afin de lier l'utilisateur à sa plante.

Le même formulaire a été utilisé, avec les informations pré-remplies, pour la modification d'une plante en back-office.

Lors de la liaison entre la partie back-end et front-end, l'utilisateur peut alors envoyer ses informations pour la création d'une plante, qui sont traduite au format JSON et transmises à la base de données.

### Route *Watering*

Afin de compléter le calendrier personnel d'un utilisateur, il nous était nécessaire de récupérer les informations d'arrosage d'une plante sous une certaine syntaxe et pour laquelle nous avons créé une route supplémentaire, permettant de récupérer ces informations sous forme d'un tableau JSON récupérable en API par le front-office. Cette fonction consiste à créer un tableau en récupérant la date d'initialisation renseignée par l'utilisateur et l'intervalle de jour d'arrosage également renseigné par l'utilisateur.

L'interprétation de ces données par le front-office s'est avérée plus compliquée que prévu, nous nous sommes donc concertés pour concevoir une nouvelle version de l'application, en repensant l'organisation du MVC, les relations entre les tables, le format du stockage des données. Cette première version nous a permis d'apprendre beaucoup sur les besoins d'une telle application et de la démarche à suivre pour son bon fonctionnement.

# Présentation du jeu d'essai

Pour le jeu d'essai, je vais procéder à la création d'une plante dans la base de données via le back-office.

Pour ce faire je vais passer par le formulaire d'ajout de plante. Celui-ci comporte différentes entrées que je vais compléter et soumettre le formulaire. Devrait en résulter la création d'une plante dans notre base de données.

Voici donc les données en entrée :

### Créer une nouvelle plante

Nom de la plante : Cactus

Photo de la plante : Choisir un fichier 10757138\_...35\_o.jpg

Age de la plante : 16 | 02 | 1993

Date d'initialisation calendrier : 01 | 05 | 2021

Type de plante : Intérieur

Niveau de compétence : débutant

Rythme d'arrosage : 30

Exposition à la lumière : forte

Période de taille : Janvier Février Mars Avril Mai Juin Juillet Août Septembre Octobre Novembre Décembre

Mois de fertilisation : Janvier Février Mars Avril Mai Juin Juillet Août Septembre Octobre Novembre Décembre

Mois de rempotage : Janvier Février Mars Avril Mai Juin Juillet Août Septembre Octobre Novembre Décembre

Spécifications de la plante : Très joli cactus

**Enregistrer**

*Formulaire de création d'une plante en back-office.*

Et voici les données attendues en sortie :

### Liste des plantes

+

Photo	Nom	Propriétaire	Spécifications	Age	Date d'initialisation	Rythme d'arrosage	Eclairage	Coupe	Fertilisation	Rempotage	Type	Niveau de compétence	Création	Mise à jour	Actions
	Kentia	admin	Comportant un tronc ou stipe unique, ce palmier se [...]	12/06/2014	08-03-2021	Tous les 3 jours	8h/jrs		Mai Juin Juillet Août	Mars Avril	Intérieur	intermédiaire	17/02/2021 09:38:12		Détails Modifier
	Laurier	admin	Couleur des fleurs : blanc			Aucune information renseignée	Oui						26/02/2021 14:07:47		Détails Modifier
	Laurier	admin	Couleur des fleurs : blanc			Aucune information renseignée	Oui						26/02/2021 14:10:25		Détails Modifier
	Cactus	admin	Très joli cactus	16/02/1993	01-05-2021	Tous les 30 jours	forte	Septembre	Avril	Septembre Octobre	Intérieur	débutant	21/04/2021 18:37:11		Détails Modifier

*Tableaux des plantes de la base de données en back-office.*

# Veille technologique sur la sécurité

Les principales sources de documentation utilisées au cours du projet sont :

- [symfony.com](#) : le point de départ de toute recherche pour le back-end. En partant du principe que la documentation officielle doit faire référence, j'ai systématiquement commencé par lire la documentation Symfony avant d'explorer d'autres sources.
- [StackOverFlow](#) : site faisant référence pour la résolution de problèmes, je l'ai consulté en particulier pour la mise en place des tokens JWT, la compréhension de la relation front-back avec API, le déploiement d'un projet React, les conflits entre bundles.
- [MDN Web Docs](#) : utilisé ponctuellement au cours du projet, sur les thématiques des CORS, du JSON et du CSS.
- [php.net](#) : utilisé en particulier pour la mise en place et la gestion de DateInterval, format de données des plantes à transférer au calendrier pour paramétriser les rappels.
- Documentation GitHub pour différents bundles, certains conservés et d'autres mis de côté : LexikJWTAuthenticationBundle, NelmioCorsBundle, OpenAPI, nelmio/alice, LiipImagineBundle.
- [aws.amazon.com](#) : son fonctionnement était déjà éprouvé, j'y ai surtout effectué des recherches dans le cadre du déploiement du front, car des solutions alternatives existaient, il me fallait distinguer celles qui étaient potentiellement utilisables avec un back Symfony de celles qui reposaient sur un back Node et un projet intégralement JavaScript.
- [apache.org](#) : utilisé lors des recherches pour paramétriser le front et le back sur le même serveur, avant le choix de Surge pour le déploiement du front.

D'autres articles ont pu être consultés ponctuellement, pour confirmer ou approfondir certains points, mais les sources ci-dessus ont été majoritairement utilisées et détenaient la plupart des réponses recherchées.

# Extrait d'une ressource anglophone et traduction

L'un des obstacles qui s'est présenté à nous fût la mise en place des tokens pour l'identification des utilisateurs. Nous nous sommes donc renseignés auprès d'une ressource extérieure pour la disposition du *token* via le bundle Lexik JWT.

La majorité des contenus concernant le développement web étant en anglais, il est bien indispensable de pouvoir les comprendre afin de bien les utiliser.

Voici donc un extrait de cette ressource extérieure, accompagné de ma traduction personnelle. Celle-ci nous a aidés dans l'installation du bundle Lexik JWT, et guidé dans sa configuration.

## Usage

### 1. Obtain the token

The first step is to authenticate the user using its credentials.

You can test getting the token with a simple curl command like this (adapt host and port):

```
curl -X POST -H "Content-Type: application/json" http://localhost/api/login_check -d '{"username":"johndoe", "password":"te'>
```

If it works, you will receive something like this:

```
{  
    "token" : "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXUYJ9eyJleHAiOiE0MzQ3Mjc1MzYsInVzZXJuYW1lIjoia29ybGVvbiIsImhdCI6IjE0MzQ2NDE  
}'>
```

Store it (client side), the JWT is reusable until its ttl has expired (3600 seconds by default).

### 2. Use the token

Simply pass the JWT on each request to the protected firewall, either as an authorization header or as a query parameter.

By default only the authorization header mode is enabled : `Authorization: Bearer {token}`

See [configuration reference](#) document to enable query string parameter mode or change the header value prefix.

## Traduction personnelle :

### Utilisation

#### 1. Obtenir le jeton

La première étape est d'authentifier l'utilisateur en utilisant ses identifiants.

Vous pouvez récupérer le jeton à l'air d'une simple commande curl comme ceci (adaptez l'hôte et le port) :

Si cela marche, vous recevrez quelque chose comme ceci :

Enregistrez-le (côté client), le jeton est réutilisable jusqu'à ce que sa *ttl* (*time-to-live* ou temps de vie) ait expiré (3600 secondes par défaut).

## 2. Utiliser le jeton

Passez simplement le jeton à chaque requête au pare-feu de protection, soit en tant qu'*header* d'autorisation soit en tant que paramètre *query*.

Par défaut seule l'header d'autorisation est activé : Authorization : Bearer {token}

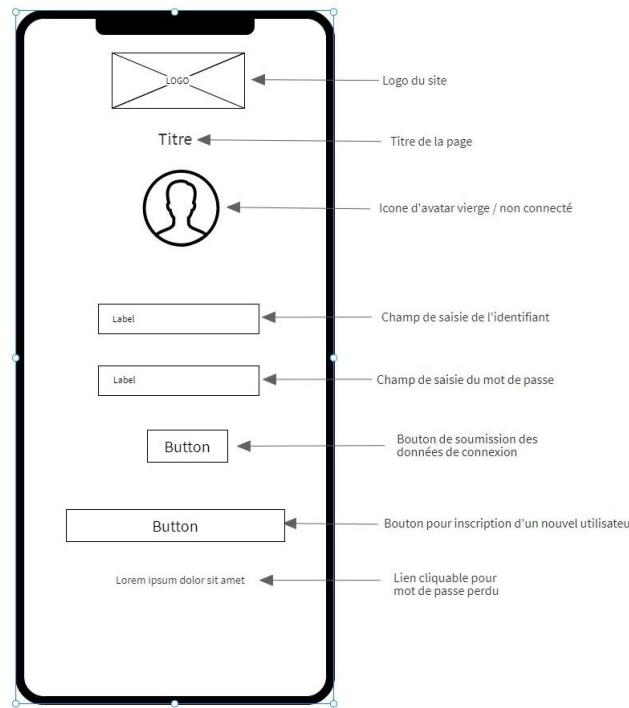
Voyez le document de référence configuration pour activer le mode de paramètre de chaîne de caractère *query* ou pour changer la valeur du prefix en *header*.

# Conclusion

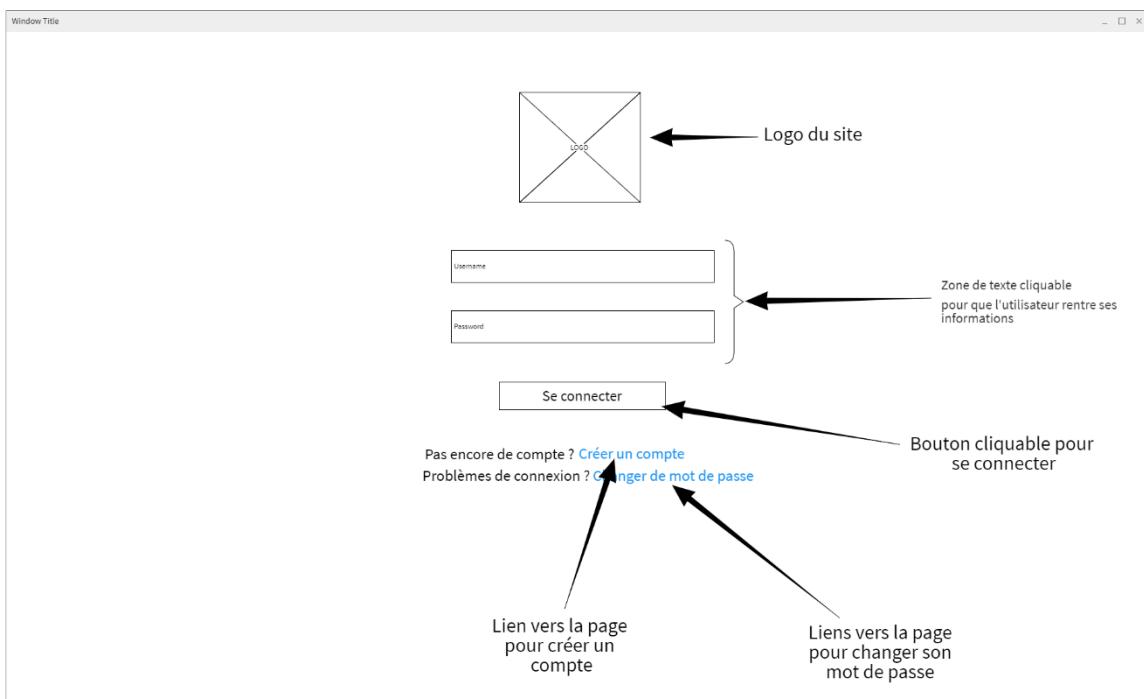
Plantopia représente le dénouement d'une formation qui fût intense et passionnante. Ce projet nous a permis de mettre en pratique toute la théorie que nous avions intégrée depuis ces derniers mois intenses.

Cela a été l'occasion pour moi d'approfondir beaucoup de notions vues pendant la formation, et d'apprendre beaucoup de choses de par la nécessité et les besoins de notre application. J'ai pu développer et maîtriser la création de base de données ainsi que leur manipulation, la création de back-office, la communication entre front-end et back-end sur 2 projets différents, le déploiement d'une application, la sécurité autant en back-end qu'en front-end (validation des données entrées par l'utilisateur et contrôle en back-office). Ce projet fût une véritable aventure et je suis impatient de pouvoir y retourner afin de le peaufiner, le retravailler, le perfectionner.

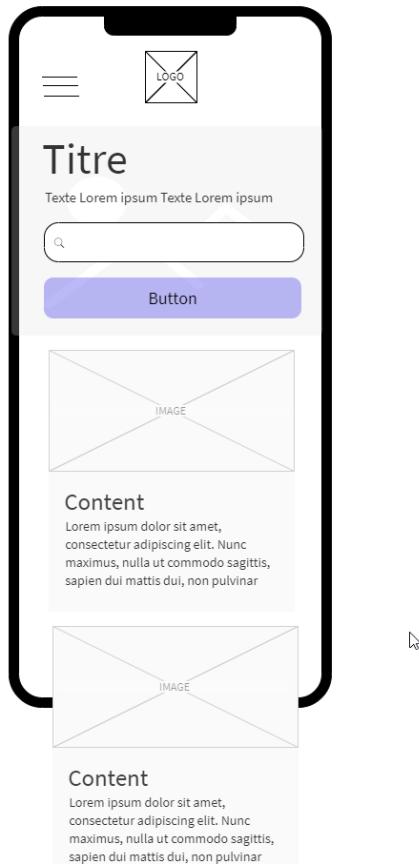
# Annexes



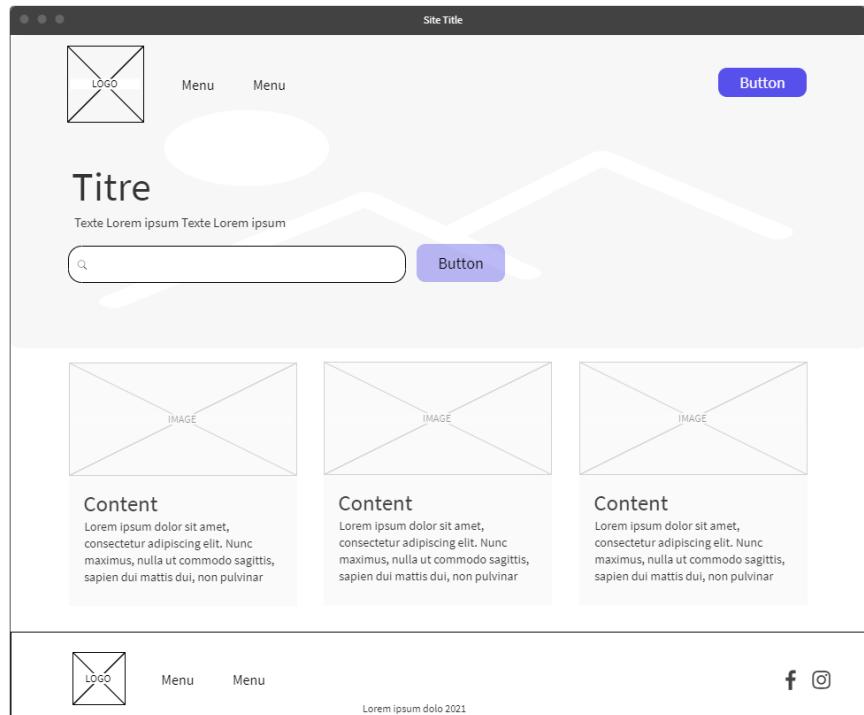
*Page de login, version mobile.*



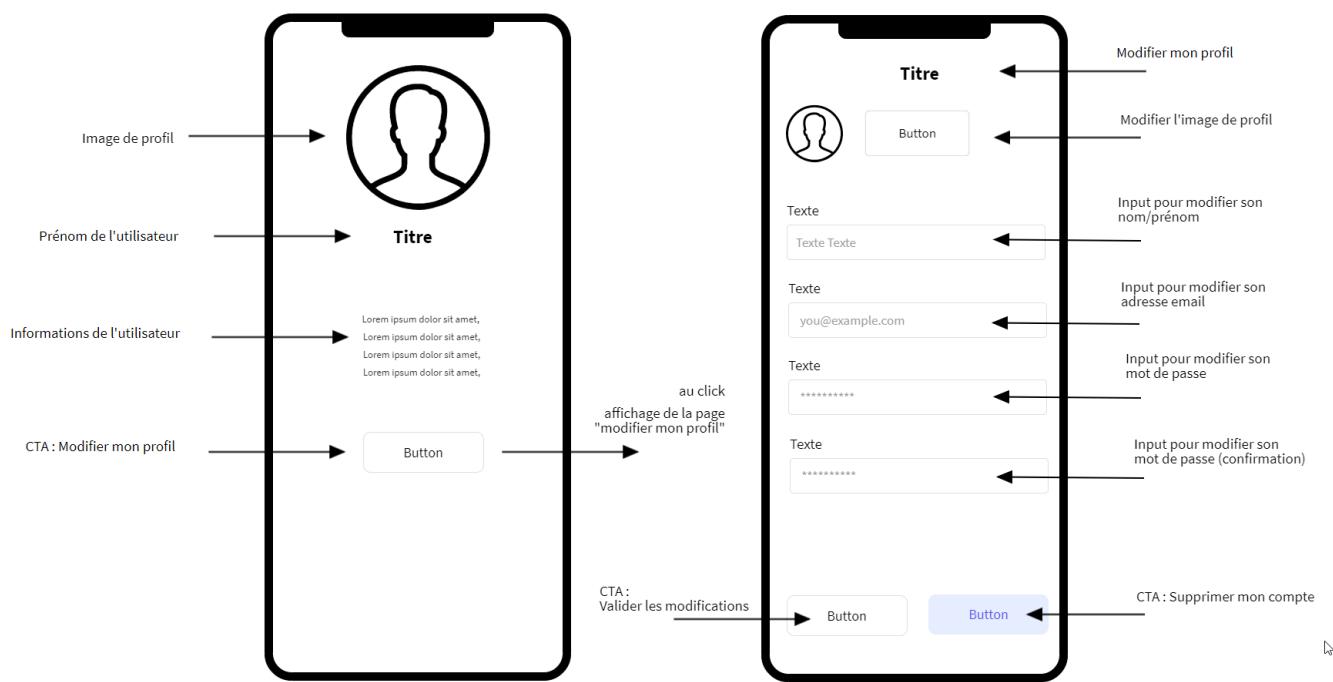
*Page de login, version desktop.*



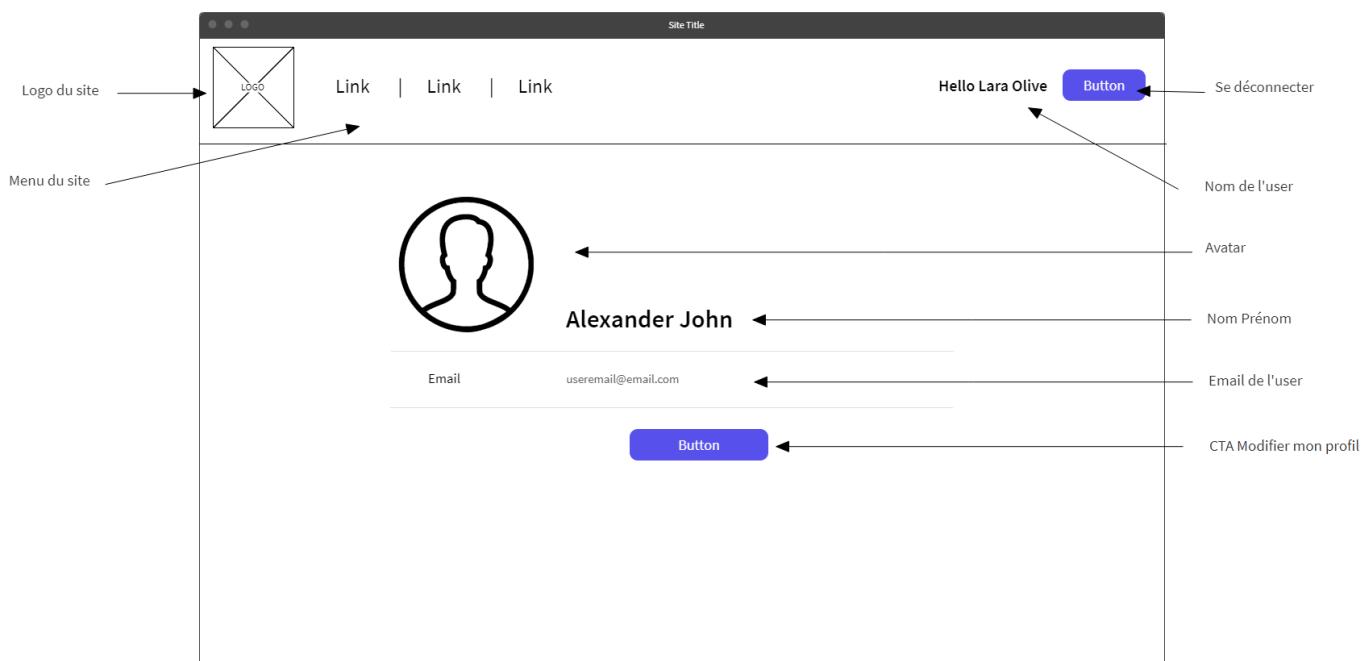
*Page d'accueil, version mobile.*



*Page d'accueil, version desktop.*

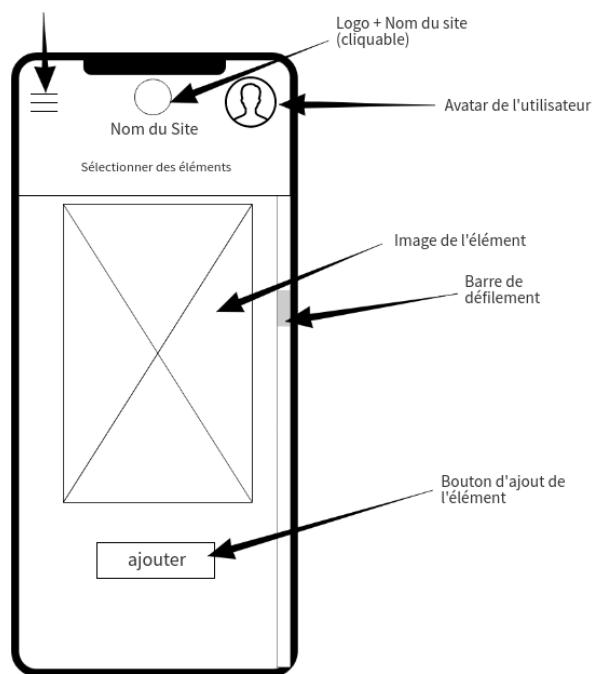


*Page de profil, version mobile.*

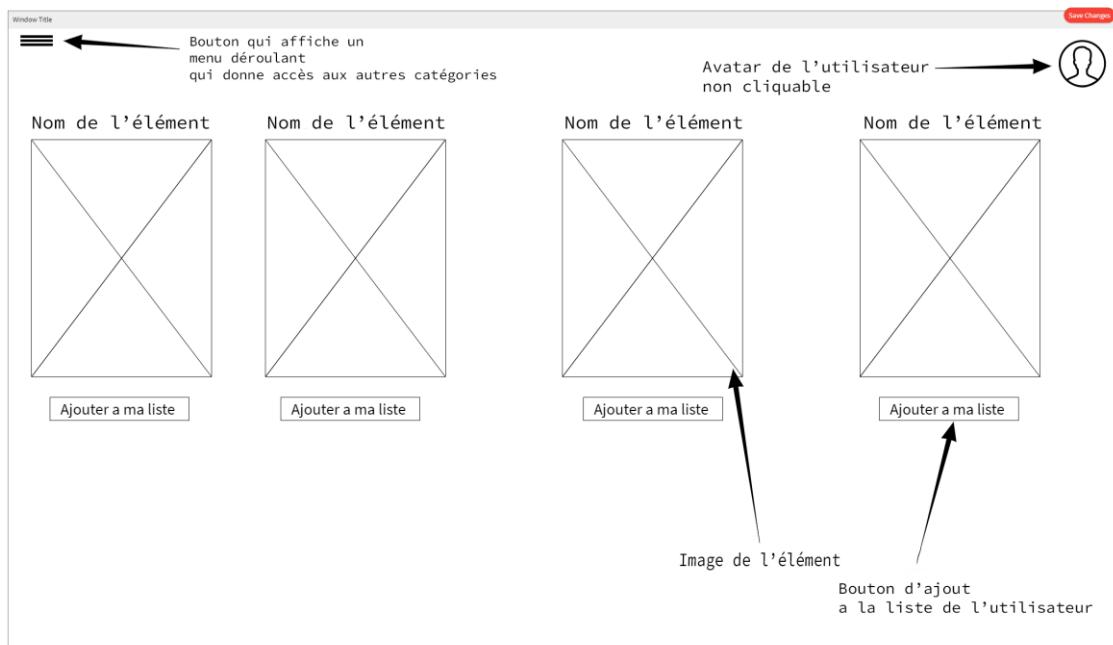


*Page de profil, version desktop.*

Menu déroulant

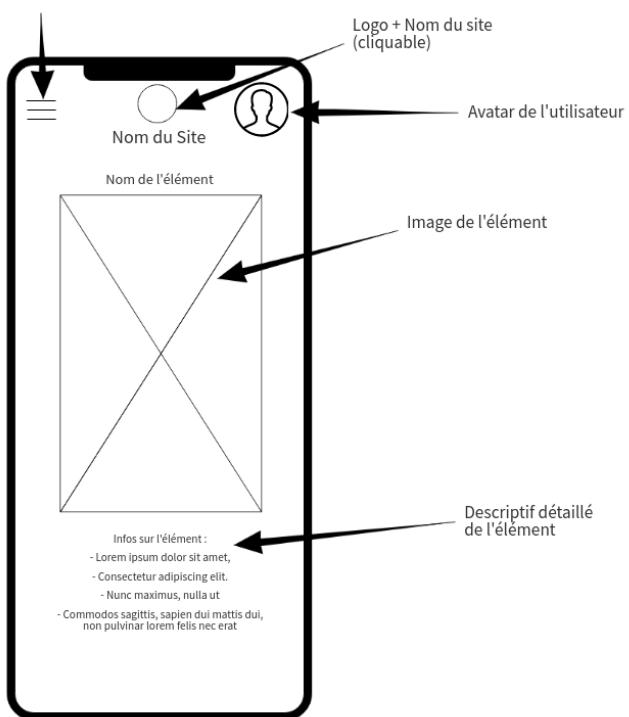


*Ajout de plante, version mobile.*

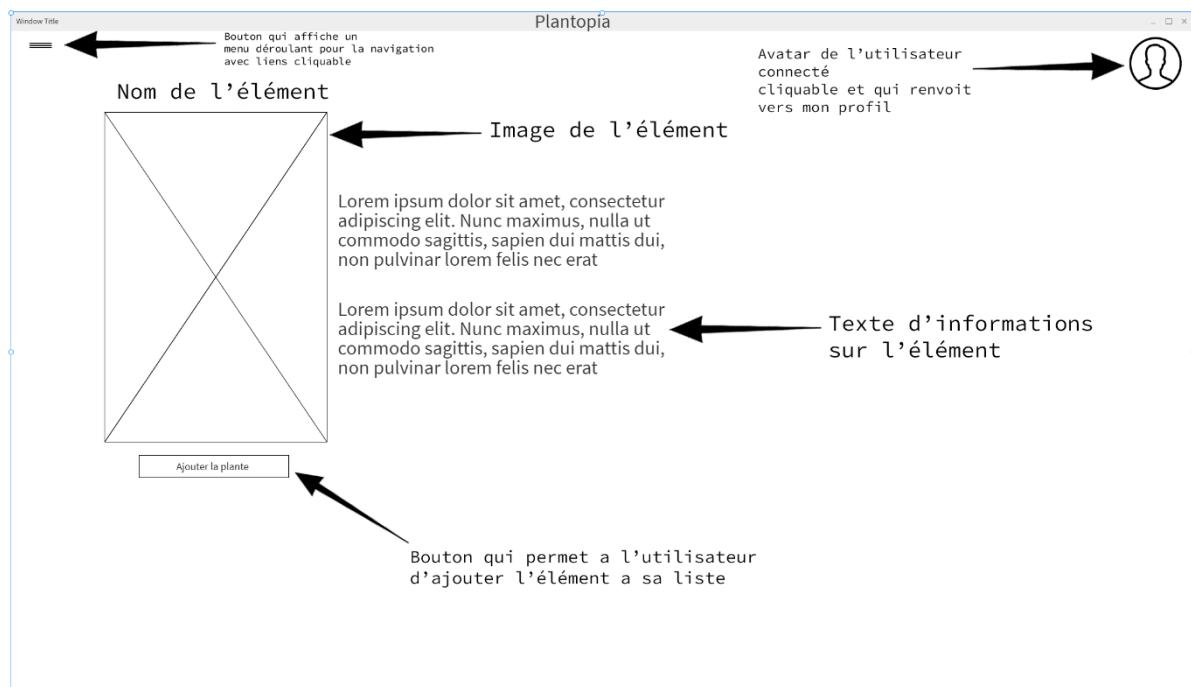


*Ajout de plante, version desktop.*

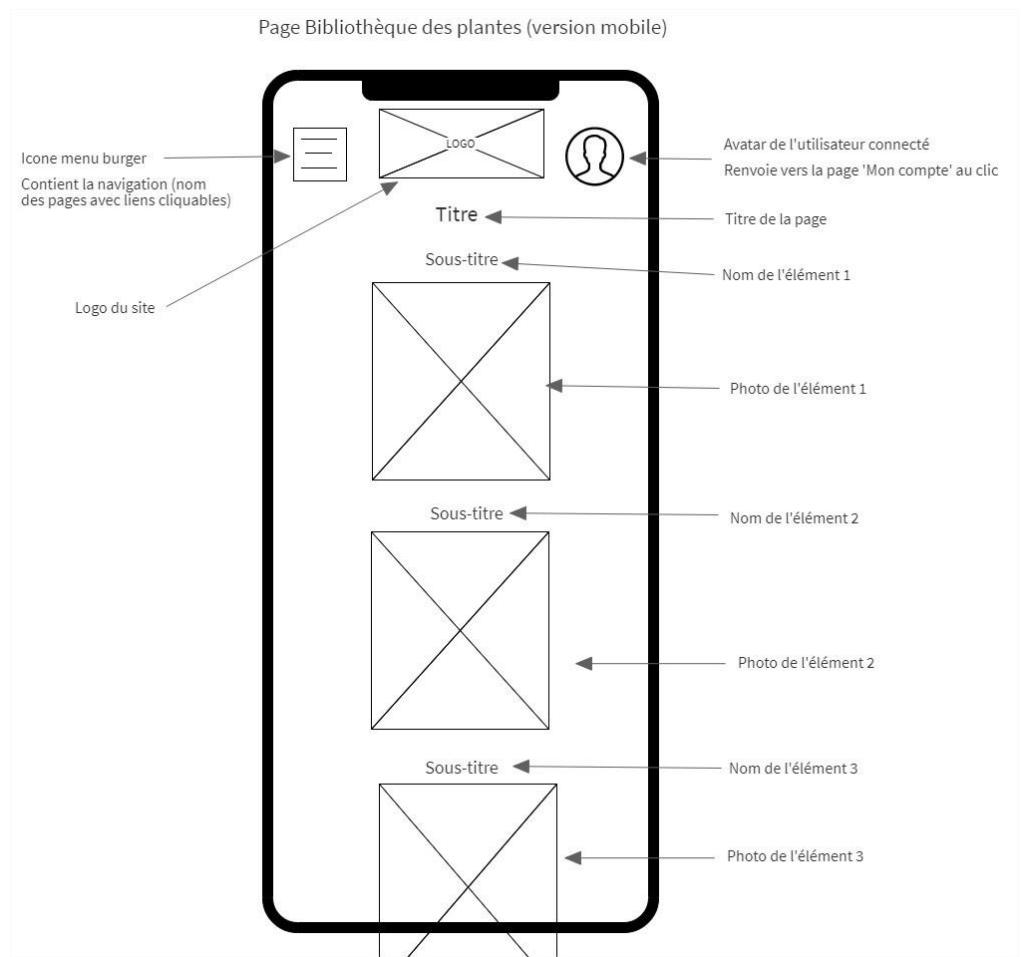
Menu déroulant



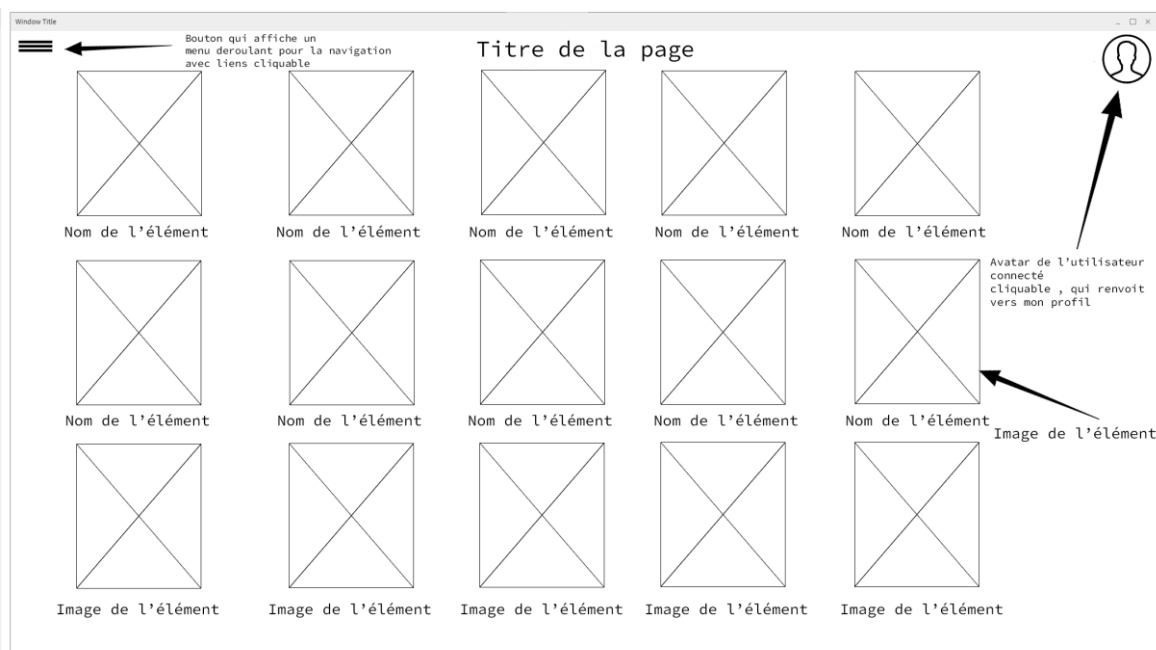
Page d'une plante, version **mobile**.



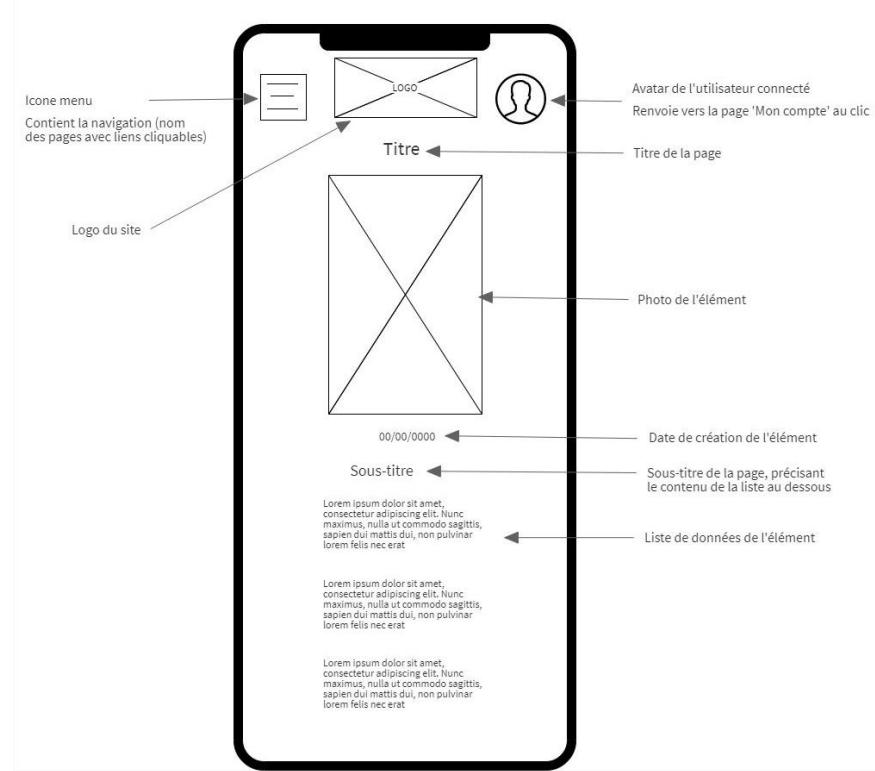
Page d'une plante, version **desktop**.



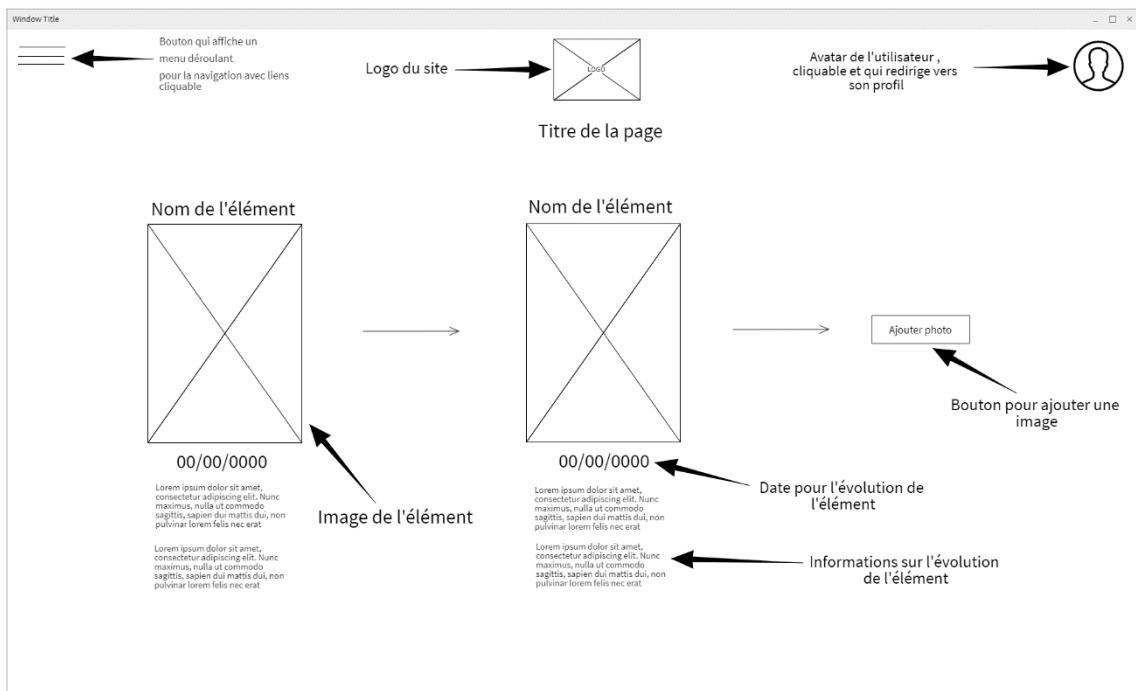
*Bibliothèque de plantes, version **mobile**.*



*Bibliothèque de plantes, version **desktop**.*



*Historique d'une plante, version mobile.*



*Historique d'une plante, version desktop.*

## Dictionnaire de données.

### Plante ( plant )

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	plant's ID
name	VARCHAR(128)	NOT NULL	plant's name
specification	LONGTEXT	NULL	additional informations : wintering, specs...
age	DATE	NULL	plant's age
initialization_date	DATE	NULL	calendar starting point
watering	TINYINT	NULL	date interval of watering
light	TEXT	NULL	plant's light exposure method
cut	ENUM	NULL	cut's month
fertilization	TINYINT	NULL	plant's fertilization interval
repotting	TEXT	NULL	plant's repotting method
picture	TEXT	NULL	plant's picture
type_id	INT	FOREIGN KEY, NULL, UNSIGNED	plant's type : inside or outside
skill_id	INT	FOREIGN KEY, NULL, UNSIGNED	users required skills : beginner, average or expert
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	plant's creation date
updated_at	DATETIME	NULL	plant's update date

*Table Plant et ses spécificités.*

### Utilisateur ( user )

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	user's ID
pseudo	VARCHAR(64)	NOT NULL	user's pseudo
password	VARCHAR(64)	NOT NULL	user's password
email	VARCHAR(64)	NOT NULL	user's email
avatar	VARCHAR(64)	NULL	user's avatar
created_at	DATETIME	NOT NULL, DEFAULT CURRENT_TIMESTAMP	user's creation date
updated_at	DATETIME	NULL	user's update date

*La table User.*

### Type ( type )

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	type's ID
name	VARCHAR(64)	NOT NULL	type's name (0=inside, 1=outside)

*La table Type, destinée à s'expandre.*

## Skill ( skill )

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	user's ID
name	VARCHAR(64)	NOT NULL	user's skill (0=beginner, 1=average, 2=expert)

*La table Skill.*

## CalendarSchedule ( calendar\_schedule )

Champ	Type	Spécificités	Description
user_id	INT	FOREIGN KEY, NOT NULL, UNSIGNED	user's ID
plant_id	INT	FOREIGN KEY, NOT NULL, UNSIGNED	plant's ID

*La table Calendar Schedule, permettant la création d'un calendrier personnel.*

---

## API : Endpoints et controllers

### Endpoints

Routes	Nom de la route	Méthodes (HTTP)	Commentaires
/api/plants	api_plants_browse	GET	
/api/plants/{id}	api_plants_read	GET	
/api/plants/edit/{id}	api_plants_edit	PUT	
/api/plants	api_plants_add	POST	
/api/plants/delete/{id}	api_plants_delete	DELETE	
/api/myaccount/browse	api_myaccount_browse	GET	
/api/myaccount	api_myaccount_read	GET	
/api/myaccount/edit/{id}	api_myaccount_edit	PUT	
/api/myaccount	api_myaccount_add	POST	
/api/myaccount/watering	api_myaccount_add	POST	
/api/myaccount/fertilization	api_myaccount_add	POST	
/api/myaccount/repotting	api_myaccount_add	POST	
/api/myaccount/cut	api_myaccount_add	POST	
/api/myaccount/delete/{id}	api_myaccount_delete	DELETE	
/api/login_check	-	POST	Credentials check and token generation

*Liste des différents endpoints de l'API, avec leur nom de route et leur méthode HTTP.*

## Controllers

Routes	Controller	->méthode()	Commentaires
api_plants_browse	ApiPlantController	->browse()	
api_plants_read	ApiPlantController	->read()	
api_plants_edit	ApiPlantController	->edit()	
api_plants_add	ApiPlantController	->add()	
api_plants_delete	ApiPlantController	->delete()	
api_myaccount_browse	ApiController	->browse()	
api_myaccount_read	ApiController	->read()	
api_myaccount_edit	ApiController	->edit()	
api_myaccount_add	ApiController	->add()	
api_myaccount_watering	ApiController	->watering()	
api_myaccount_fertilization	ApiController	->fertilization()	
api_myaccount_repotting	ApiController	->repotting()	
api_myaccount_cut	ApiController	->cut()	
api_myaccount_delete	ApiController	->delete()	

**Controllers** utilisés pour l'API, avec les **méthodes** utilisant le **BREAD**.

## Services : Interval Stringer

```
● ● ●
1 <?php
2
3 namespace App\Service;
4
5 class IntervalStringer
6 {
7
8 /**
9  * Affiche un DateInterval lisible par les humains
10 *
11 * @param DateInterval $date_interval
12 *      Intervalle à afficher
13 * @return String
14 *      Intervalle lisible par les humains
15 */
16
17     function interval_to_string($date_interval)
18     {
19         $parametres = array(
20             array("y", 'année', 'années'),
21             array("m", 'mois', 'mois'),
22             array("d", 'jour', 'jours'),
23             array("h", 'heure', 'heures'),
24             array("i", 'minute', 'minutes'),
25             array("s", 'seconde', 'secondes'),
26         );
27
28         $resultat = array();
29
30         foreach ($parametres as $parametre)
31         {
32             $format = $parametre[0];
33             $singulier = $parametre[1];
34             $pluriel = $parametre[2];
35
36             if ($date_interval->format("%".$format) > 0)
37             {
38                 $resultat[] = $date_interval->format('%'.$format)." ".($date_interval->format('%'.$format) > 1 ? $pluriel : $singulier);
39             }
40         }
41
42         return implode(', ', $resultat);
43     }
44 }
```

*IntervalStringer.php, facilitant la lecture des **DateInterval()**.*

```
● ● ●
1 /**
2  * @Route("/{id}", name="read", methods="GET")
3 */
4 public function read (Plant $plant, PlantRepository $plantRepo, IntervalStringer $intervalStringer): Response
5 {
6     if (is_null($plant)) {
7         throw new NotFoundHttpException("");
8     }
9     else {
10         $plantW = "";
11         if (!is_null($plant->getWatering())) {
12             $plantW = $intervalStringer->interval_to_string($plant->getWatering());
13         }
14         $plantId = $plant->getId();
15         $plantType = $plantRepo->findTypeById($plantId);
16         $plantSkill = $plantRepo->findSkillById($plantId);
17         return $this->json([$plant, $plantType, $plantSkill, $plantW]);
18     }
19 }
```

*Utilisation du service **IntervalStringer** dans le controller d'API **apiPlantController.php**.*

```
1 public function buildForm(FormBuilderInterface $builder, array $options)
2 {
3     $builder
4         ->add('name', TextType::class, [
5             'label' => 'Nom de la plante',
6             'constraints' => [
7                 new NotBlank()
8             ]
9         ])
10        ->add('picture', FileType::class, [
11            'mapped' => false,
12            'label' => 'Photo de la plante',
13            'required' => false,
14            'constraints' => [
15                new File([
16                    'maxSize' => '2M'
17                ])
18            ]
19        ])
20        // ->add('created_at')
21        ->add('specification', TextareaType::class, [
22            'label' => 'Spécifications de la plante',
23            'required' => false
24        ])
25        ->add('age', BirthdayType::class, [
26            'label' => 'Age de la plante',
27            'widget' => 'choice',
28            'format' => 'ddMMyyyy',
29            'placeholder' => '',
30            'required' => false,
31        ])
32        ->add('initialization_date', DateType::class, [
33            'label' => 'Date d\'initialisation calendrier',
34            'widget' => 'choice',
35            'format' => 'ddMMyyyy',
36            'placeholder' => '',
37            'required' => false,
38        ])
39        ->add('type', null, [
40            'label' => 'Type de plante',
41            'expanded' => false,
42        ])
43        ->add('skill', null, [
44            'label' => 'Niveau de compétence',
45            'expanded' => false,
46        ])
47        ->add('watering', DateIntervalType::class, [
48            'label' => 'Rythme d\'arrosage',
49            'days' => array_combine(range(1, 60), range(1, 60)),
50            'with_days' => true,
51            'with_years' => false,
52            'with_months' => false,
53            'required' => false,
54        ])
55        ->add('light', TextType::class, [
56            'label' => 'Exposition à la lumière',
57            'required' => false,
58        ])

```

```

59         ->add('cut', ChoiceType::class, [
60             'label' => 'Période de taille',
61             'expanded' => true,
62             'multiple' => true,
63             'required' => false,
64             'choices' => [
65                 'Janvier' => "Janvier",
66                 'Février' => "Février",
67                 'Mars' => "Mars",
68                 'Avril' => "Avril",
69                 'Mai' => "Mai",
70                 'Juin' => "Juin",
71                 'Juillet' => "Juillet",
72                 'Août' => "Août",
73                 'Septembre' => "Septembre",
74                 'Octobre' => "Octobre",
75                 'Novembre' => "Novembre",
76                 'Décembre' => "Décembre",
77             ],
78             'required' => false
79         ])
80     ->add('fertilization', ChoiceType::class, [
81         'label' => 'Mois de fertilisation',
82         'expanded' => true,
83         'multiple' => true,
84         'required' => false,
85         'choices' => [
86             'Janvier' => "Janvier",
87             'Février' => "Février",
88             'Mars' => "Mars",
89             'Avril' => "Avril",
90             'Mai' => "Mai",
91             'Juin' => "Juin",
92             'Juillet' => "Juillet",
93             'Août' => "Août",
94             'Septembre' => "Septembre",
95             'Octobre' => "Octobre",
96             'Novembre' => "Novembre",
97             'Décembre' => "Décembre",
98         ],
99         'required' => false
100    ])
101    ->add('repotting', ChoiceType::class, [
102        'label' => 'Mois de rempotage',
103        'expanded' => true,
104        'multiple' => true,
105        'required' => false,
106        'choices' => [
107            'Janvier' => "Janvier",
108            'Février' => "Février",
109            'Mars' => "Mars",
110            'Avril' => "Avril",
111            'Mai' => "Mai",
112            'Juin' => "Juin",
113            'Juillet' => "Juillet",
114            'Août' => "Août",
115            'Septembre' => "Septembre",
116            'Octobre' => "Octobre",
117            'Novembre' => "Novembre",
118            'Décembre' => "Décembre",
119        ],
120        'required' => false
121    ]);
122    ;
123 }

```

**Formulaire d'ajout d'une plante.**

```

1  /**
2   * @Route("", name="add", methods="POST")
3   */
4  public function add (Request $request, EntityManagerInterface $em, Slugger $slugger): Response
5  {
6      $plantAsArray = json_decode($request->getContent(), true);
7
8      $plant = new Plant();
9
10     $form = $this->createForm(PlantType::class, $plant, ['csrf_protection' => false]);
11     $form->submit($plantAsArray);
12     if ($form->isValid())
13     {
14         $plantSkill = $form->get('skill')->getData();
15         $plant->setSkill($plantSkill);
16         $plantType = $form->get('type')->getData();
17         $plant->setType($plantType);
18         $calendarSchedule = new CalendarSchedule();
19         $user = $this->getUser();
20         $calendarSchedule->addPlant($plant);
21         $calendarSchedule->setUser($user);
22         $em->persist($calendarSchedule);
23         $em->persist($plant);
24         $em->flush();
25         // on slugifie après le flush car slugifyPlant a besoin d'un id
26         $slugger->slugifyPlant($plant);
27     }
28     //dd($plant);
29     return $this->json([$plant, $plantType, $plantSkill], 201, [], [
30         AbstractNormalizer::IGNORED_ATTRIBUTES => [
31             'calendarSchedule'
32         ]
33     ]);
34 }

```

*Fonction add de l'ApiPlantController.php*

```

1 /**
2  * @Route("/watering", name="watering", methods="GET")
3 */
4 public function watering(Request $request, UserRepository $userRepo)
5 {
6     $user = $this->getUser();
7     $userId = $user->getId();
8
9     $listeDateArrosage = [];
10
11    $plants = $userRepository->findPlantByUser($userId);
12
13    foreach ($plants as $plant) {
14
15        $dateDeDebut = $plant["initialization_date"];
16        $joursInterval = $plant["watering"];
17        $titre = $plant['name'];
18
19        if (!is_null($joursInterval)) {
20
21            $dateCalendrier = new DatePeriod($dateDeDebut, $joursInterval, 10);
22
23            for ($date = 0; $date<10; $date++)
24            {
25                $tableauTemp = [
26                    "title" => $titre,
27                    "start" => $dateCalendrier->getStartDate(),
28                    "intervalle" => $joursInterval
29                ];
30            }
31
32            $listeDateArrosage[] = $tableauTemp;
33        }
34    }
35
36    return $this->json($listeDateArrosage);
37 }

```

*Route additionnelle **watering**, du ApiUserController.php*

Plantopia - /api/myaccount/watering/{id}

200 OK    533 ms    398 B

Preview	Header 10	Cookie	Timeline
<pre> 1 [ 2   [ 3     { 4       "titre": "Piranha", 5       "date de début": "2020-06-27T00:00:00+02:00", 6       "intervalle": "P0Y0M34DT0H0M0S" 7     }, 8     { 9       "titre": "Laurier", 10      "date de début": "2021-01-17T00:00:00+01:00", 11      "intervalle": "P0Y0M34DT0H0M0S" 12    }, 13    { 14      "titre": "Cactus", 15      "date de début": "2021-03-09T00:00:00+01:00", 16      "intervalle": "P0Y0M30DT0H0M0S" 17    }, 18    { 19      "titre": "Kentia", 20      "date de début": "2021-03-08T00:00:00+01:00", 21      "intervalle": "P0Y0M3DT0H0M0S" 22  ] </pre>			

*Tableau JSON résultant de la route **watering**.*



```
1  {% for plant in plants %}
2    <tr>
3      <td>{{ if (plant.picture is null) }}
4        
5      {% else %}
6        
7      {% endif %}
8    </td>
9    <td><a class="btnplantindex" href="/plant/{{ plant.id }}"/>{{ plant.name }}</a></td>
10   <td><a class="btnplantindex" href="/user/{{ plant.calendarSchedule.user.id }}"/>{{ plant.calendarSchedule.user.pseudo }}</a></td>
11
12  <td>{{ plant.specification | slice(0, 50) }}
13  {% if plant.specification is defined %} {% if plant.specification|length >50 %} ...
14  ...
15  {% endif %}
16  {% endif %}
17 </td>
18  <td>{{ plant.age ? plant.age|date('d/m/Y') : '' }}</td>
19  <td>{{ plant.initializationDate ? plant.initializationDate|date('d-m-Y') : '' }}</td>
20  <td>{{ if(plant.watering.d) is defined }}
21    {% if (plant.watering.d == 1) %}
22      Tous les jours
23    {% elseif plant.watering.d > 1 %}
24      Tous les {{ plant.watering.d }} jours
25    {% endif %}
26    {% else %} Aucune information renseignée
27    {% endif %}
28 </td>
29  <td>{{ plant.light }}</td>
30  #<td>{% for cut in plant.cut %}
31    {{ cut }}
32  {% endfor %}
33 </td> #
34  <td>{{ if plant.cut %}
35  {% if plant.cut|length == 1 %}
36  {{ plant.cut.0 }}
37  {% else %}
38  {{ plant.cut|first }} -> {{ plant.cut|last }}</td>
39  {% endif %}
40  {% endif %}
41  <td>{% for fertilization in plant.fertilization %}
42    {{ fertilization }}
43  {% endfor %}
44 </td>
45  <td>{% for repotting in plant.repotting %}
46    {{ repotting }}
47  {% endfor %}
48 </td>
49  <td>{{ if (plant.type is defined) }}
50  {{ plant.type }}
51  {% endif %}
52 </td>
53  <td>{{ if (plant.skill is defined) }}
54  {{ plant.skill }}
55  {% endif %}
56 </td>
57  <td>{{ plant.createdAt ? plant.createdAt|date('d/m/Y H:i:s') : '' }}</td>
58  <td>{{ plant.updatedAt ? plant.updatedAt|date('d/m/Y H:i:s') : '' }}</td>
59  <td>
60    <a href="{{ path('plant_show', {id: plant.id}) }}" class="btnaction">Détails</a>
61    <a href="{{ path('plant_edit', {id: plant.id}) }}" class="btnaction">Modifier</a>
62  </td>
63 </tr>
64 {% else %}
65 <tr>
66  <td colspan="14">Aucune donnée trouvée</td>
67 </tr>
68 {% endfor %}
```

plant/index.html.twig

Liste des plantes																
																
Photo	Nom	Propriétaire	Spécifications	Age	Date d'initialisation	Rythme d'arrosage	Eclairage	Coupe	Fertilisation	Rempotage	Type	Niveau de compétence	Création	Mise à jour	Actions	
	Kentia	admin	Comportant un tronc ou stipe unique, ce palmier se [...]	12/06/2014	08-03-2021	Tous les 3 jours	8h/jrs		Mai Juin Juillet Août	Mars Avril	Intérieur	intermédiaire	17/02/2021 09:38:12		<a href="#">Détails</a> <a href="#">Modifier</a>	
	Laurier	admin	Couleur des fleurs : blanc			Aucune information renseignée	Oui							26/02/2021 14:07:47		<a href="#">Détails</a> <a href="#">Modifier</a>
	Cactus	admin	Très joli cactus	16/02/1993	01-05-2021	Tous les 30 jours	forte	Septembre	Avril	Septembre Octobre	Intérieur	débutant	21/04/2021 18:37:11	21/04/2021 19:03:09	<a href="#">Détails</a> <a href="#">Modifier</a>	

**Tableau représentant les données de la table plant.**

### Créer une nouvelle plante

Nom de la plante

Photo de la plante  Choisir un fichier Aucun fichier choisi

Age de la plante

Date d'initialisation calendrier

Type de plante

Niveau de compétence

Rythme d'arrosage

Exposition à la lumière

Période de taille  Janvier  Février  Mars  Avril  Mai  Juin  Juillet  Août  Septembre  Octobre  Novembre  Décembre

Mois de fertilisation  Janvier  Février  Mars  Avril  Mai  Juin  Juillet  Août  Septembre  Octobre  Novembre  Décembre

Mois de rempotage  Janvier  Février  Mars  Avril  Mai  Juin  Juillet  Août  Septembre  Octobre  Novembre  Décembre

Spécifications de la plante

[Enregistrer](#)

[Retour à la liste](#)

**Formulaire d'ajout d'une plante en back-office.**



```
1 class FileUploader
2 {
3
4     private $plantFolder;
5     private $userFolder;
6
7     public function __construct($plantFolder, $userFolder)
8     {
9         $this->plantFolder = $plantFolder;
10        $this->userFolder = $userFolder;
11    }
12
13 /**
14 * Et là j'ai la même fonctionnalité dédiée à un cas particulier
15 *
16 * @param UploadedFile|null $image on autorise le null si jamais aucune image n'a été fournie
17 * @return string|null
18 */
19 function moveImage(?UploadedFile $image, string $targetFolder, $prefix = ''): ?string
20 {
21     $newFilename = null;
22
23     if ($image !== null) {
24         // on a décidé d'appeler notre fichier
25         $newFilename = $prefix . uniqid() . '.' . $image->guessExtension();
26
27         // Move the file to the directory where brochures are stored
28         $image->move(
29             $targetFolder,
30             $newFilename
31         );
32     }
33     return $newFilename;
34 }
35
36 function movePlantPicture(?UploadedFile $image, Plant $plant)
37 {
38     $imageName = $this->moveImage($image, $this->plantFolder, 'plant-');
39     if ($imageName !== null) {
40         $plant->setPicture($imageName);
41     }
42 }
43
44 function moveUserAvatar(?UploadedFile $image, User $user)
45 {
46     $imageName = $this->moveImage($image, $this->userFolder, 'user-');
47     if ($imageName !== null) {
48         $user->setAvatar($imageName);
49     }
50 }
51 }
```

le service **FileUploader**, permettant le **stockage** d'images uploadées par l'utilisateur.

# Glossaire

**API** : *Application Programming Interface*, ou Interface de programmation d'application.

**API REST** : API respectant les contraintes REST.

**BREAD** : *Browse, Read, Edit, Add, Delete*. Cette méthode permet l'accès à de vastes bases de données en listant, détaillant, modifiant, ajoutant et supprimant des données.

**GitHub** : service web d'hébergement et de gestion de développement de logiciels.

**Doctrine** : ORM (Object Relational Mapping) utilisé par défaut par Symfony.

**MCD** : Modèle Conceptuel de Données.

**MVP** : Minimum Viable Product.

**ORM** : Object-Relational Mapping.

**Symfony** : Framework PHP.

**React** : Librairie Javascript.

Redux : Librairie complémentaire de React. Permet d'envoyer des informations, de faire passer des props (données) et de faire des actions en asynchrone en liant les composants avec la base de données.

**Token** : Jeton. En informatique, sert à identifier un utilisateur.

**Twig** : Moteur de templates intégré à Symfony.