

2021/2022

EasyGarden

[Dossier de projet]



LEFEVRE Emmanuel
<LE CABINET DE CURIOSITES>

Table des matières

1.	Introduction.....	2
2.	Présentation de l'association	2
3.	Résumé du projet.....	3
4.	Cahier des charges	5
4.1	Expression des besoins.....	5
4.1.1	Définition du projet.....	5
4.1.2	Objectifs du projet	6
4.1.3	Cibles.....	6
4.2	Logo et charte graphique.....	8
4.3	Cahier des charges fonctionnelles.....	9
4.3.1	Besoin du client.....	9
4.3.2	Solution envisagée par le client	9
4.3.3	Règles métiers.....	10
4.4	Cahier des charges techniques	12
4.4.1	Back-end	12
4.4.2	Front-end	15
4.4.3	Cross Origin	17
4.4.4	SGDB	17
4.4.5	Hardware	17
4.4.6	Versioning	18
5.	Réalisation.....	19
5.1	Environnement	19
5.2	Back-end	19
5.3	Front-end	33
5.4	Hardware	61
6.	Remerciements	63
7.	Conclusion	63
8.	Annexes	64

1. Introduction

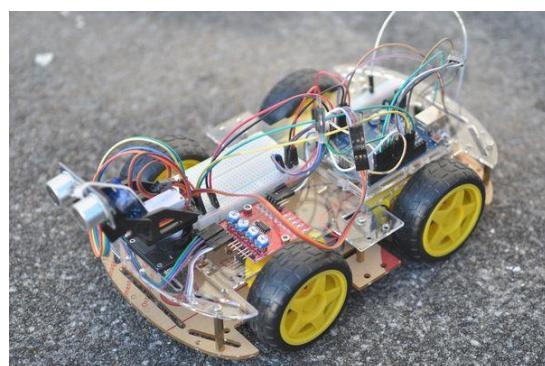
J'ai réalisé mon stage de formation dans le cadre de mon titre professionnel Concepteur Développeur d'Applications au sein de l'association Le cabinet de curiosités située à Angoulême.

J'ai tout d'abord analysé de manière précise les attentes et les besoins afin de rédiger ensuite un cahier des charges détaillé.

Ma tâche principale fut de développer une solution web et mobile permettant de gérer des équipements de parcs et jardins, connaître leur état en temps réel et être en mesure de les activer/désactiver via une IHM (Interface Homme-Machine), créer un compte utilisateur ainsi qu'un espace permettant la modification de ses données personnelles et un second pour la gestion de plusieurs jardins liés à ce même utilisateur.

2. Présentation de l'association

Le cabinet de curiosités fut tout d'abord une association à but non lucratif, créée en octobre 2019, celle-ci offrait à ses adhérents un cadre destiné à l'apprentissage par la réalisation de projets ayant une composante scientifique et technique liée à la robotique et la domotique. Dans ce cadre furent développés et réalisés un draw bot, un vélo générateur, des rovers fabriqués par exemple à base de Lego Technics ou bien encore un four solaire intelligent et connecté. Un atelier propose par exemple de dessiner à l'encre conductrice afin d'aborder les principes de l'électricité, un autre d'apprendre à programmer en pilotant les aiguillages d'un train miniature ou bien encore de manipuler les mathématiques en programmant un robot capable de sortir d'un labyrinthe. Le dernier en date était de construire un mini-véhicule capable de gravir la pente la plus abrupte possible, l'occasion d'éprouver les concepts de centre de gravité, de roues motrices et de démultiplication.



Son président Mr Clément Nicolas est titulaire d'un DUT Génie Electrique et Informatique Industrielle, d'un diplôme d'ingénieur dans la même spécialité citée précédemment ainsi que d'un DESS Intelligence artificielle Reconnaissance des formes et Robotique. Il a ensuite

entres autres travaillé pour Cap Gemini, Thalès Air System et Alten. Il a exercé en tant que professeur de technologie au lycée Saint Paul d'Angoulême, un dernier poste qui l'a convaincu de son appétence pour transmettre le goût des sciences aux plus jeunes.

Il a de plus été co-auteur et porteur du projet de magazine web « Le meilleur des mondes est à écrire ». Ce programme est devenu force de proposition pour Arte, avec qui il a collaboré plus d'un an, tant sur l'éditorialisation que sur la médiation. Le but était de proposer des fictions et anticipations sur les nouvelles technologies, et comment celles-ci allaient évoluer (l'analogie avec la série « Black Mirror » me semble pertinente).

Courant 2022, un changement s'opère, le cabinet de curiosités change de statut pour devenir une SARL et ouvre un laboratoire participatif à Angoulême dans lequel seront proposés des formations au numérique, à la robotique et à l'utilisation de logiciels. Le but premier de l'entreprise est d'encadrer des activités ludo-scientifiques et numériques pour les jeunes et de proposer des stages de robotiques durant les vacances. C'est un club de robotique qui encourage l'exploration scientifique en proposant un cadre pédagogique et en suscitant la curiosité des participants. Les adultes sont aussi concernés, puisque des ateliers permettront de se former à la conception, à la programmation ou au développement de projets techniques. Des imprimantes 3D, des appareils de découpe laser, des banques de composants et de cartes électroniques seront mis à disposition des adhérents afin de les accompagner dans la réalisation de leurs projets.

La pédagogie autour de tout cela ? « L'envie d'accompagner les jeunes tout au long de l'année pour qu'ils acquièrent de l'autonomie dans leurs projets et découvrent les filières scientifiques et les métiers techniques », souligne Nicolas. L'objectif du Cabinet de curiosités n'est pas d'imiter l'école, mais de proposer au public une alternative à la promotion et la découverte des sciences. C'est un outil pour traiter les sciences de manière détournée, appliquée et intuitive. Plusieurs formules d'abonnements sont possibles, ponctuelles ou à l'année.

Le cabinet de curiosités

129, rue de clerc a sillac

16000 Angoulême

Tel : 06 75 87 67 34

Email : clementnico@hotmail.com

Page Facebook : <https://www.facebook.com/CabinetCuriositesAngouleme>

3. Résumé du projet

Dans le cadre de mon stage j'ai souhaité créer une application de gestion d'équipements dans le cadre du domaine des parcs et jardins. Cette solution applicative devra convenir dans le même temps aux particuliers comme aux professionnels. Pour ce faire il lui faudra adopter un design épuré et une certaine facilité d'utilisation non moins performante vis-à-vis d'une utilisation par des professionnels. Elle devra aussi être constituée de deux supports, l'un mobile et le second pour le web. De plus, une interface responsive accessible via le web pour tous types de smartphones serait un gros plus afin d'étendre au persona l'attribut -support et mode d'utilisation- et donc enrichir le profil du groupe d'utilisateurs ciblé.

Le choix du nom s'est porté sur «EasyGarden», la charte graphique devra tout naturellement être à dominante verte. Le projet devra lui aussi être «vert» et inclusif d'un certain nombre de principe d'éco-conception web tel qu'un code source propre et conforme aux bonnes pratiques du web (W3C, ECMAScript, PSR4, frameworks, minifier le CSS...), un hébergeur écologique, limiter les animations ainsi que le nombre de pages navigables, optimiser les images, faciliter l'accessibilité de EasyGarden aux personnes en situation de handicap (déficience visuelle et auditive, troubles cognitifs) ...

Cette solution se devra d'être innovante, utile, esthétique et durable. Les adjectifs définissant l'usabilité du service seront simplicité, facilité, navigabilité, compréhensibilité et accessibilité. Ces premiers critères seront déterminants afin de limiter au maximum le taux de rétention et de maximiser le taux de fidélité.

En outre, elle apportera un gain de temps, d'énergie ainsi que la possibilité d'interagir à distance et de n'importe où sur les équipements de son jardin tels que mettre en eau les différents réseaux d'arrosage intégré et automatique, démarrer son robot tondeuse, allumer les divers éclairages d'ambiance paysagés, commander les jets d'eau, la roue à aubes et le Manneken Pis de son bassin ou alors ouvrir son portail à distance grâce à la version mobile. Le tout depuis une seule et unique application.

EasyGarden mettra à disposition deux interfaces propres à chaque rôle ici même déterminé, c'est-à-dire un rôle User et un rôle Admin.

EasyGarden doit aussi permettre de gérer des utilisateurs, ceux-ci doivent pouvoir s'inscrire, modifier et supprimer leurs données de profil, accéder à une liste des jardins s'ils en possèdent plusieurs. Une interface administration privative et exclusive est de mise pour ce qui est tout d'abord de l'assistance et du support client mais aussi pour permettre de modifier le statut d'un équipement défectueux par exemple.

Certains contrôles devront de plus être intégrés, qu'ils soient d'ordre «sécuritaire» (l'impossibilité de commander la fermeture du portail si le capteur de présence est activé ou mettre en fonctionnement un éclairage si l'ampoule est défectueuse) et d'ordre «raisonnée» (rendre impossible l'allumage des éclairages si la luminosité est importante ou démarrer le robot tondeuse si la batterie est quasiment déchargée).

La majorité de ces contrôles s'effectueront côté serveur mais aussi via l'interface mise à disposition des utilisateurs et ce, en rendant les boutons inactifs. Cette même interface

devra être totalement ergonomique et ne disposer que du strict minimum du point de vue des interactions afin de ne pas perturber l'expérience d'utilisation de EasyGarden.

Le changement de statut des équipements et l'affichage en temps réel (Two-Way Binding) des données sur l'IHM s'effectuera via une API Rest (Application Mobile Interface) qui sera chargée d'apporter une procédure sécurisée dans le mode d'échange des données et un ORM (Doctrine) qui se chargera de stocker ces informations en base de données relationnelles (MySQL). La partie GPIO (General Purpose Input/Output) fonctionnera quant à elle grâce à un Raspberry Pi (Raspbian distribution dérivée de Debian) qui communiquera par BLE (bluetooth) avec des Arduino. L'utilisation de frameworks a été choisie: Angular13 en front et Symfony6 en back, ainsi que de ApiPlatform.

La sécurité de l'application et des données est un point décisif, il est primordial de mettre tout en œuvre pour sécuriser EasyGarden d'un point de vue technologique. Le parallèle avec la RGPD n'est pas fortuit et se tenir informé sur le site de la CNIL est une bonne pratique.

Pour finir, la maintenance est un élément substantiel à prendre en compte, qui plus est éco-responsable! Adopter de bonnes pratiques sur le sujet amène sur le long terme un gain de temps et de travail. Pour ce faire il convient de prendre soin de bien commenter et factoriser son code, utiliser des modèles génériques de templatings ou bien encore utiliser des variables dynamiques.

4. Cahier des charges

4.1 Expression des besoins

Le projet consiste à créer une application desktop et mobile permettant à l'utilisateur de gérer des équipements de parcs et jardins ainsi que d'administrer son compte utilisateur.

4.1.1 Définition du projet

Le projet de EasyGarden est en premier lieu de répondre à un besoin pour mon propre jardin mais l'absence d'offre de service sur ce secteur pourtant porteur est un objectif à garder en tête! En effet selon les chiffres de l'Insee datant de 2008 «Les logements des français» dont une partie était consacrée au jardin, il apparaît que 59 % des français disposeraient d'un jardin soit 39M de personnes rien que pour notre si verdoyant pays.

EasyGarden sera disponible pour tous types d'appareils allant de Android, IOS, tablette, smartphone à desktop, le but étant de rendre accessible au plus grand nombre l'application quel que soit le support utilisé.

EasyGarden sera utilisée à l'extérieur comme à l'intérieur d'une habitation et nécessitera soit une connexion internet/wifi permettant une utilisation depuis l'autre côté du globe soit une connexion bluetooth pour une utilisation locale déconnectée du web. Le sens d'utilisation de l'appareil en version smartphone/tablette s'effectuera aussi bien de manière horizontale que verticale.

Dans un premier temps EasyGarden se destinera à être commercialisée uniquement sur le territoire de la France et ne bénéficiera que d'une seule langue en l'occurrence le français. Celle-ci restera néanmoins accessible pour les pays francophones.

4.1.2 Objectifs du projet

La méthode choisie pour la définition des objectifs est la méthode SMART qui est une méthode de management par objectif introduite par Peter Drucker. Celle-ci permet de se fixer des buts et objectifs clairs et pertinents pour son activité. Cette méthode est applicable d'un point de vue général comme plus spécifique tel qu'une fonctionnalité.

1/ Spécifique (définir clairement le résultat attendu):

Il s'agit de développer une application permettant de gérer des équipements de parcs et jardins et d'en afficher le suivi des données en temps réel.

2/ Mesurable (quantifier l'objectif afin de déterminer s'il est atteint):

Mettre en place des outils de cartographie mentale tel que le mindmapping ainsi qu'un cycle de méthode agile afin de surveiller et contrôler les échéances des livrables. Un daily-scrum est un bon moyen de se situer journalièrement et la méthode Kanban hebdomadairement.

3/ Atteignable (définir des moyens réalistes):

Définir des objectifs acceptables tout en restant ambitieux car pour obtenir de l'implication la cible à atteindre doit nécessiter un effort conséquent.

4/ Réaliste (environnement, ressources humaines et techniques):

Etant quasiment seul à travailler sur ce projet je dois effectuer ma montée en compétences de manière autodidacte sur les technologies ciblées. Mieux organiser mon temps de travail afin d'intégrer plus rapidement les nouvelles connaissances requises, celles-ci acquises dans les travaux pratiques et cours en ligne que je suis, nécessaires à la bonne conduite du projet en cours.

5/ Temporel (date limite de l'objectif):

L'association me donne le temps nécessaire à l'aboutissement de ce projet. Il est donc essentiel de bien doser mon investissement personnel et de conserver une certaine productivité afin de rester efficace. Cela me permettra de garder cette notion de sentiment d'urgence lorsque l'on est soumis à une deadline.

4.1.3 Cibles

L'étude de l'Insee de 2008 met en avant que 89% des français se disent adeptes du jardinage. Il apparaît que 59 % d'entre eux disposeraient d'un jardin (soit 39M de personnes

rien que pour notre pays), 89% d'un espace de jardinage. Elle révèle de même que la moitié des jardins ont une superficie d'au moins 600 m² et que le panier moyen annuel est de 113 euros (tandis qu'au pays des tulipes, les Pays-Bas, le panier moyen est quant à lui de 241 euros). C'est-à-dire que ceux-ci dépensent une somme plus de deux fois supérieure à leurs homologues de l'Hexagone : une somme qui s'explique notamment par une attraction historique très forte des néerlandais pour l'aménagement extérieur. Malgré tout ces chiffres sont revus à la hausse depuis la crise sanitaire et ce secteur tend à continuer de se développer dans les futures années.

Selon une enquête menée en 2019 par l'Ifop et l'Unef 7 français sur 10 prennent plaisir à jardiner.

Acteurs primaires : Persona USER lambda.



— Quote —

“ Pouvoir gérer simplement mon jardin à distance est un gain de temps non négligeable, mon emploi du temps et le temps de trajet jusqu'à mon lieu de travail ne me laissant que très peu de disponibilités.

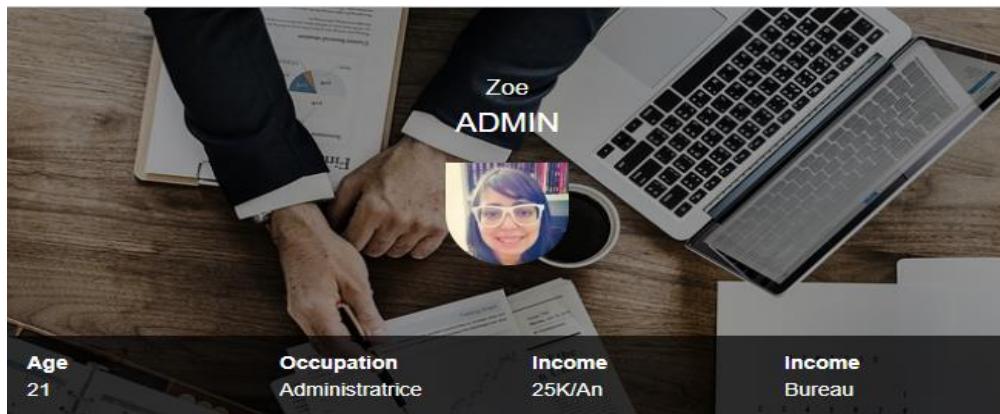
Acteurs primaires : Persona USER professionnel.



— Quote —

“ Une simple application regroupant tout les jardins dont j'ai la charge de l'entretien.

Acteur secondaire : Persona ADMIN



— Quote —

“ Administrer une application et offrir un service client grâce à une interface personnalisée ainsi que de gérer des utilisateurs ainsi que leur rôle.

4.2 Logo et charte graphique

Comme énoncé dans le résumé du projet, le nom de l'application «EasyGarden» a été choisi, la charte graphique devra donc être à dominante verte. Le choix du logo s'est porté sur une feuille de fougère afin de rester dans le ton du projet :



Celui-ci sera placé dans le menu, ainsi qu'en miniature dans le libellé de l'onglet, afin de mettre en avant la touche EasyGarden.

Le choix des polices de caractères s'est porté sur la police Roboto (plusieurs de ses graisses) et la police Charlotte Southern:

- Roboto 400 sans-serif pour la police principale.
- Roboto 100 sans-serif pour le contenu des tableaux des équipements.
- Roboto 400 serif pour les <th> des tableaux des équipements.
- Roboto 300 sans-serif pour les boutons.
- Charlotte Southern pour les titres.

En ce qui concerne le choix des couleurs, la couleur dominante sera la couleur verte #95cb11.

La police d'écriture ainsi que les ombrages seront de couleur grise ou blanche.

Les backgrounds hériteront de trois couleurs neutres afin de conserver sobre le design de EasyGarden.

Les couleurs des cellules des tableaux seront grises alternées.

4.3 Cahier des charges fonctionnelles

4.3.1 Besoin du client

Le besoin est de développer un outil facilitant la gestion d'équipements de parcs et jardins. Cet outil est destiné à des particuliers qui auront la possibilité de créer et de gérer un compte utilisateur, et doit permettre de plus à l'entreprise d'administrer tout ces utilisateurs.

La solution à développer doit permettre à chaque utilisateur la gestion d'un ou plusieurs jardins qui possèdent des équipements. Ces équipements seront l'arrosage, l'éclairage, le portail, le bassin et la tondeuse.

D'autre part toutes ces informations devront être enregistrer et accessibles.

Pour finir l'entreprise devra avoir la possibilité de gérer tout les utilisateurs dans un but d'assistance clientèle et de maintenance technique.

4.3.2 Solution envisagée par le client

La solution envisagée par le client est donc une application desktop et mobile qui regroupera à travers différentes pages des moyens de gestion des équipements et de profil utilisateur. Ces pages contiendront les incontournables:

Public

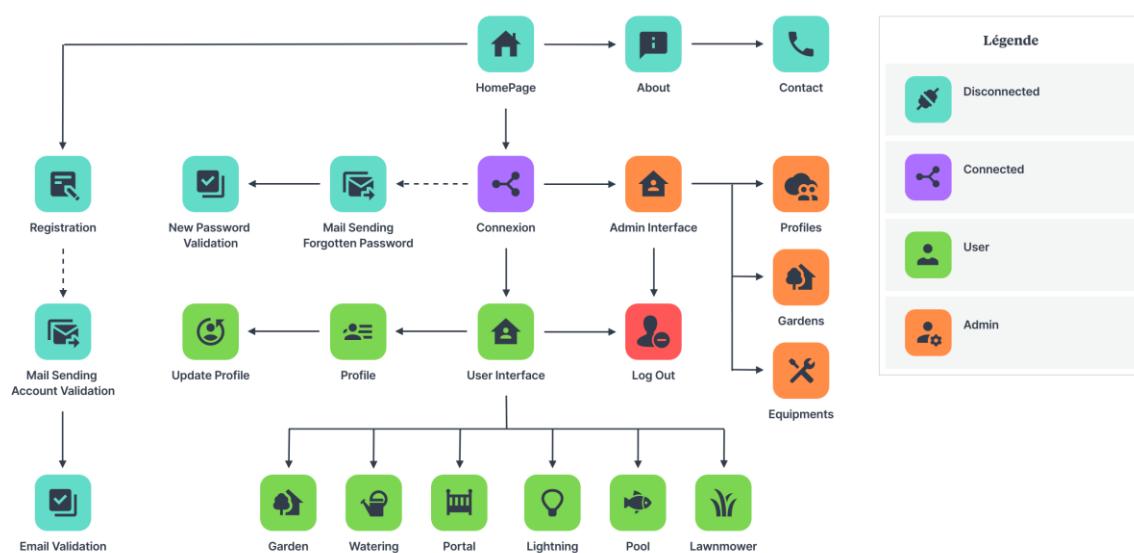
- Accueil
- RGPD
- Contact
- A propos
- Création d' un compte
- Connexion

Privé

- Jardin

- Arrosage
- Eclairage
- Tondeuse
- Bassin
- Portail
- Profil

L'ensemble de cette description est représenté grâce au site map suivant.



Matériel de pilotage et de contrôle

Sur site des boîtiers de commande et de contrôle devront permettre le pilotage des différents équipements via l'application. Les utilisateurs pourront ainsi contrôler les boîtiers. Ceux-ci seront répartis dans le jardin ils devront être étanches et alimentés par secteur si cela est possible (sinon par batterie). Pour faciliter leur installation les boîtiers communiqueront par des moyens non filaires en l'occurrence du BLE.

4.3.3 Règles métiers

Utilisateur:

- Un utilisateur peut posséder aucun ou plusieurs jardins.
- Un utilisateur a un rôle (User ou Admin).
- Un utilisateur a un nom.
- Un utilisateur a un prénom.
- Un utilisateur a un email.

- Un utilisateur a un pseudo.
- Un utilisateur a un numéro de téléphone.
- Un utilisateur a une date de création de compte.
- Un utilisateur a une date de modification de compte.
- Un utilisateur a un compte vérifié ou non.

Jardin:

- Un jardin est associé à un seul utilisateur.
- Un jardin est associé à aucun ou plusieurs arrosages.
- Un jardin est associé à aucun ou plusieurs éclairages.
- Un jardin est associé à aucun ou plusieurs bassins.
- Un jardin est associé à aucune ou une tondeuse.
- Un jardin est associé à aucun ou plusieurs portails.
- Un jardin a un nom.

Arrosage:

- Un arrosage est associé à un unique jardin.
- Un arrosage possède un nom.
- Un arrosage comprend un capteur de pression.
- Un arrosage comprend un capteur de débit.
- Un arrosage peut être activé ou désactivé.

Bassin:

- Un bassin est associé à un unique jardin.
- Un bassin possède un nom.
- Un équipement de bassin peut-être activé ou désactivé.

Éclairage:

- Un éclairage est associé à un unique jardin.
- Un éclairage possède un nom.
- Un éclairage peut-être activé ou désactivé.

Portail:

- Un portail est associé à un unique jardin.

- Un portail possède un nom.
- Un portail peut-être ouvert ou fermé.
- Un portail comprend un capteur de présence.

Tondeuse:

- Une tondeuse est associé à un unique jardin.
- Une tondeuse possède un nom.
- Une tondeuse peut-être allumée ou éteinte.
- Une tondeuse comprend un capteur de niveau de batterie.

4.4 Cahier des charges techniques

4.4.1 Back-end

Pour la partie back-end, mon choix s'est naturellement dirigé vers l'utilisation d'un framework PHP. Ce langage est majoritairement utilisé sur le web. En effet l'utilisation d'un framework permet tout d'abord de construire un projet plus rapidement, de plus celui-ci est livré avec des composants et des modules génériques qui peuvent être réutilisés. Un framework représente ainsi un ensemble de fonctions au sein d'un système. Il est généralement soumis à une structure comme par exemple le pattern Model Vue Controller (MVC). Il favorise également la maintenance à long terme, la stabilité et la haute évolutivité, tout en préservant la bonne conformité du code.

De ces faits le choix s'est porté sur Symfony6, framework PHP créé par Fabien Potencier, le dirigeant de Sensiolabs, fonctionnant sur le modèle MVC.



Au-delà des caractéristiques citées plus haut, Sf6 est complètement configurable et est d'ailleurs souvent considéré comme le meilleur framework pour la création d'applications web hautement sécurisées. Il permet aussi de développer en POO (Programmation Orientée Objet). Une documentation, des forums et une communauté active ont par ailleurs aussi motivés mon choix. Il est également livré avec les meilleures pratiques intégrées qui peuvent être facilement appliquées sans avoir à en comprendre le fond. Il est aussi bien adapté à des utilisateurs débutants comme avancés. Sf6 offre également plusieurs façons et mécanismes de mise en cache pour améliorer les performances systèmes.

Sf6 se présente sous la forme d'un bundle complet, chaque bundle fournissant des fonctionnalités qu'il est possible d'utiliser ou pas selon ses besoins. Cela le rend très flexible et extensible. Ceux-ci, s'ils conviennent à un besoin très spécifique, peuvent de même être réutilisés dans un autre projet. Il dispose d'un composant de routing que nous utiliserons sous le format d'annotations. Egalement il intègre un puissant ORM (Object Relational Mapper) du nom de Doctrine, celui-ci permettant de mapper un objet PHP à des éléments d'un système de persistance.



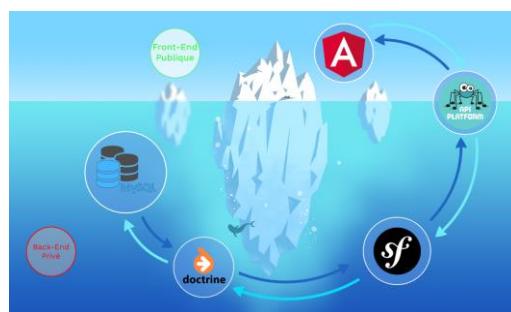
Doctrine possède son propre langage le DQL (Doctrine Query Language) qui est un langage de requête orienté objet propre à lui-même. Il peut être utilisé à la place du langage SQL pour créer les requêtes d'accès et de manipulation de la base de données.

Mon choix s'étant porté sur des frameworks front et back, écrits dans des langages de programmation différents, l'utilisation d'une API était nécessaire. La question de la sécurité m'a amené à choisir une API REST (créeée en 2000 par Roy Fielding, qui a participé au développement de HTML, HTTP, Apache...) plutôt que SOAP. Il est à prévoir dans le futur de sécuriser les routes de L'API. Il est primordial que le client ne connaisse que le point d'entrée de l'API ainsi que le type de données qui est attendu.



Un autre de ses principaux avantages est qu'elle soit totalement « stateless » : les interactions persistent malgré une coupure serveur, le client garde en mémoire des informations sans avoir constamment besoin de les demander au serveur. REST se veut donc orientée client-serveur.

Les API REST sont basées sur HTTP (Hypertext Transfer Protocol). Cela propose une meilleure intégration. Ce ne sont que des méthodes qui peuvent définir les requêtes que le client peut effectuer GET, POST, PUT, DELETE. L'analogie avec la surface visible et cachée d'un iceberg est fréquente.



Il est important de garder à l'esprit que la réponse envoyée n'est pas une ressource, c'est la représentation d'une ressource. La représentation choisie sera du format JSON car celle-ci représente un objet (stockage des données multi-types) et est très légère grâce à sa syntaxe simple. Qui plus est JSON est issu de l'univers JS.



Une rapide recherche internet m'a permis de découvrir ApiPlatform pour ce qui est du développement de l'API avec Symfony6.

Google

X

Tous
Vidéos
Images
Actualités
Shopping
Plus
Outils

Environ 7290 000 résultats (0,43 secondes)

Qu'est-ce que Symfony? **Symfony est un cadriel MVC (Modèle Vue Contrôleur) développé en PHP par la compagnie SensioLabs.** Le français Fabien Potencier est à l'origine de ce projet grandiose. Aujourd'hui, plusieurs projets open source reposent sur des Bundles (modules) de Symfony. 15 avr. 2016

API REST
Avec
Symfony

<https://atomrace.com> › créer une API REST avec Symfony

[Créer une API REST avec Symfony - Atomrace](#)

[À propos des extraits optimisés](#) • [Commentaires](#)

<https://symfony.com> › ... › French ▾

Exposer une API avec API Platform (Symfony Docs)

Une API pourrait être utilisée par une application mobile pour afficher toutes les conférences, leurs commentaires, et peut-être permettre la soumission de ...

ApiPlatform est un framework web utilisé pour générer des API REST et GraphQL, se basant sur le pattern de conception MVC. La partie serveur du framework est écrite en PHP et basée sur le framework Symfony, tandis que la partie client est écrite en JavaScript et TypeScript. ApiPlatform permet de créer rapidement et efficacement de puissantes API REST au moyen de quelques annotations sur nos entités. ApiPlatform tire profit de nombreux composants Symfony, tel que le Serializer, Validator, HTTPFoundation, Doctrine... De plus la création d'une documentation au format OpenAPI (ex Swagger) est automatique. Il facilite également le support de nombreux formats standard: JSON, JSON+LD (Hydra) ou XML.

Sf6 est livré avec une puissante barre d'outils de débogage: le Profiler. Avec PHPUnit, il offre la première couche de test fonctionnel qui simule les requêtes HTTP et examine les résultats

sans avoir à écrire de scripts à l'aide d'outils de test. Il possède son propre fichier de gestion de dépendances composer.json.

Le framework ajoute la possibilité d'utiliser une CLI (Command Line Interface) très utilisé dans la communauté Symfony et même PHP dans son ensemble: Composer.



Composer est un gestionnaire de dépendances qui ne sont autres que toutes les bibliothèques dont le projet dépend pour fonctionner. Aussi bien pour leurs mises à jour que pour leurs paramètres d'autoload. Ces bibliothèques sont centralisées sur packagist.org

Grâce à ces atouts, le développeur peut se concentrer sur la logique applicative, ce qui est, à mon sens, l'un des aspects les plus essentiels de notre métier.

4.4.2 Front-end

Pour la partie front-end mon choix s'est aussi porté en premier lieu sur l'utilisation d'un framework orienté front-end, et ce pour en grande partie les mêmes avantages énoncés plus haut qu'ils ont à offrir. Il est nécessaire que celui-ci soit RTA (Real Time Application) afin de réagir en temps réel à l'affichage des données sur l'IHM, orienté SPA «Single-page application» et donc peu gourmand en ressources côté client. Néanmoins, désirant de plus des interactions utilisateur/IHM et prévoyant une interface minimaliste ma curiosité m'a fait découvrir le framework Angular13 qui semblait d'ores et déjà convenir à ces besoins et en bien d'autres points.



Une documentation très étayée est disponible en ligne et une grande communauté gravite autour de ce framework. Une autre de ses forces est de proposer de nouvelles versions et améliorations très régulièrement, une version majeure tous les six mois environ. Dans le même temps la rétrocompatibilité (depuis la version 4) est au cœur de la politique d'évolution de ce framework.

Ang13 est un framework de développement web, open-source, développé et géré par Google, cela induit un certain gage de qualité et sûrement une relative pérennité. Côté client, il a une structure de type MVVM (Model-View-ViewModel) qui permettra d'effectuer un contrôle très sérieux des données soumises au back-end (mais aussi au niveau de l'affichage des données sur l'IHM) grâce au TypeScript inventé par Anders Helsberg, lui-même le principal inventeur du langage C#. Le TypeScript est un sur-ensemble du langage JavaScript (tout code Js peut être utilisé avec TypeScript) qui est transcompilé en Js afin

d'être compréhensible par les navigateurs. Il permet un typage strict des variables ou d'une valeur passée vers ou en retour d'une fonction, et permet de la sorte de s'assurer du type prévu, la finalité étant d'améliorer et de sécuriser la production de code Js. Il introduit le principe de classe, à la base de la POO.

Pareillement, celui-ci met nativement à disposition de nombreuses fonctionnalités très utiles telles que la liaison des données (data-binding), les filtres (pipes), le client API, les directives, le routage, la validation des formulaires. Il inclut un module de routing et dispose d'un gestionnaire de paquets officiel nommé NPM.



Grâce à ce gestionnaire de paquets nous pouvons installer le CLI d'Ang13 de manière globale sur notre machine. NPM n'a pas besoin d'être installé. En effet il fait partie de l'environnement NodeJS qui est nécessaire au fonctionnement de Ang13.



Afin d'interpréter le code Js que l'on va écrire, NodeJS se repose sur le moteur V8 (pris au navigateur Chrome) et utilise celui-ci en dehors du navigateur. Il permet d'écrire des services côté serveur appelés API (Application Programming Interface). Il est single thread (mono-tâche) et non bloquant (il a la capacité de lancer une tâche sans forcément attendre qu'elle se finisse pour passer à la suivante). Il est donc capable de gérer énormément de requêtes en parallèle sans les faire attendre les unes les autres et c'est pourquoi il est particulièrement adapté aux SPA et RTA.

HTML (Hyper Text Markup Language) est un format de données conçu pour représenter les pages web, la version 5 en est la plus récente. Scss (Syntactically Awesome StyleSheets) est un langage de script préprocesseur qui décrit la présentation des documents HTML.



4.4.3 Cross Origin

Le partage de ressources entre origines multiples est un mécanisme qui permet à des ressources d'être récupérées par un domaine extérieur.

Lorsque le client réalise une requête HTTP multi-origin (cross-origin) il demande une ressource provenant d'un domaine, d'un protocole ou d'un port différent de ceux utilisés pour la page courante.

C'est typiquement le cas dans notre projet où Ang13 fonctionne sur le port 4200 et Sf6 sur le port 8000, le recours à un bundle de Sf6 s'avère donc nécessaire. Il se nomme NelmioCorsBundle.



4.4.4 SGDB

Les bases de données relationnelles permettent de mettre en avant les relations entre les données. C'est un répertoire d'éléments de données dotés d'une relation prédefinie entre eux. Ces éléments sont organisés en des tableaux définis, composés de colonnes et d'entrées. Ces tableaux (tables) répertorient un certain type de données. Elles permettent également de marquer avec un identifiant unique (clé primaire) un ensemble de valeurs (entrées) et d'associer ces tables entre elles en utilisant des clés étrangères. Ces données sont accessibles de multiples manières sans qu'il ne soit nécessaire de réorganiser les tables elles-mêmes.

Tout cela est donc particulièrement adapté aux besoins de cette application. MySQL étant le système de SGDB relationnel le plus utilisé à travers le monde, celui-ci fut donc retenu pour le projet.



4.4.5 Hardware

La partie hardware se composera quant à elle d'un Raspberry Pi et d'Arduinos Nano.

- Raspberry

Le Raspberry Pi est un nano-ordinateur monocarte à processeur ARM conçu par des professeurs de l'université de Cambridge dans le cadre de la fondation Rapsberry Pi. Le but à la base du projet était de créer un ordinateur dont le prix ne dépasserait pas les 35 dollars. Son principal avantage est de permettre l'exécution d'un système d'exploitation notamment le célèbre GNU/Linux mais aussi de Windows bien que celui-ci soit relativement instable. Il

dispose de pins GPIO qui permettent la connexion de cartes d'extension ou d'autres composants électroniques.

De nombreux projets sont réalisables et simples à mettre en oeuvre grâce à celui-ci, citons par exemple une console de rétro-gaming, un media center, un mini ordinateur portable, un serveur web, un système de vidéosurveillance ou une station météo.

Ici son utilisation sera d'être une passerelle de communication utilisant un protocole serial entre la box et les Arduinos.

- Arduino

Arduino est la marque d'une plateforme de prototypage open-source qui permet aux utilisateurs de créer des objets électroniques interactifs à partir de cartes électroniques matériellement libres sur lesquelles se trouve un microcontrôleur. Les schémas de ces cartes électroniques sont publiés en licence libre. Cependant, certaines composantes, comme le microcontrôleur par exemple, ne le sont pas. Le microcontrôleur peut être programmé pour analyser et produire des signaux électriques, de manière à effectuer des tâches très diverses comme la domotique. C'est une plateforme basée sur une interface entrée/sortie simple.

L'Arduino est donc une carte électronique programmable dotée d'un processeur et d'une mémoire sur laquelle on peut brancher des capteurs de température, d'humidité ou de lumière, une caméra, des boutons, des potentiomètres, des contacts de réglages. Il dispose aussi de connecteurs pour brancher des LED, des moteurs, des relais, des afficheurs ou encore un écran.

Une carte Arduino est un cerveau qui permet de rendre intelligent des systèmes électroniques et d'animer des dispositifs mécaniques.

Les Arduinos communiqueront avec le raspberry en BLE.

4.4.6 Versioning

- La version de PHP utilisée sera la version 8.0.2.
- La version de Symfony utilisée sera la version 6.
- La version d'Angular utilisée sera la version 13.
- La version de MariaDB utilisée sera la version 10.4.21.
- La version de PHPMyAdmin utilisée sera la version 5.1.1.

5. Réalisation

5.1 Environnement

Afin de développer j'utilise un éditeur de code extensible nommé VsCode, développé par Microsoft pour les trois systèmes d'exploitations principaux (Windows, Linux et MacOS).



Pour l'utilisation de ma SGDB en local j'ai installé XampServer.



XampServer est composé de Apache (serveur HTTP très populaire), MySQL, PHP et PHPMyAdmin qui est une interface pour gérer plus facilement MySQL. Ce dernier permet de mieux voir et éditer les informations liées à la base de données au lieu d'utiliser des lignes de commandes.



Celui-ci permet de se connecter à GitHub, le service en ligne où est hébergé (cloud) le code source du projet: <https://github.com/EmmanuelLefevre/EasyGardenCDA>



5.2 Back-end

J'ouvre un terminal dans VsCode et me place dans le répertoire du projet:
C:\xampp\htdocs\EasyGardenCDA puis tape la commande suivante:

`symfony new api`

Puis j'installe le CLI de Symfony et le met à jour grâce aux commandes:

`composer install`

`composer update`

Et j'installe différents bundles nécessaires à mon développement:

composer require symfony/orm-pack -> (Doctrine).

composer require symfony/maker-bundle --dev -> (Maker)

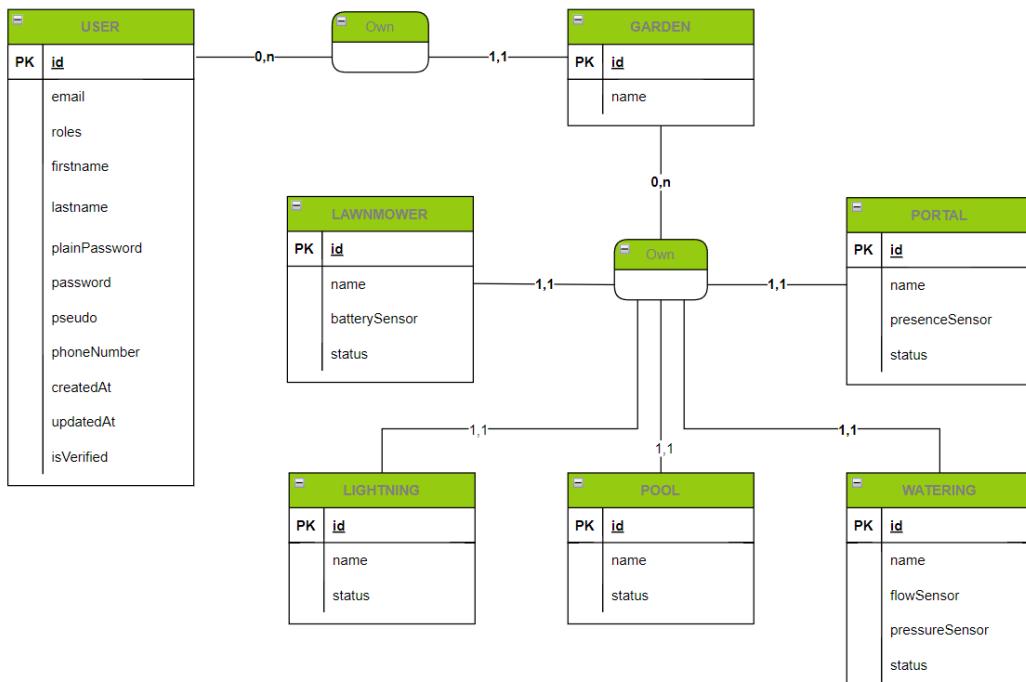
composer require symfony/profiler-pack --dev -> (Profiler)

Le projet Symfony étant créé je peux donc réaliser le modèle conceptuel de données (MCD) en m'appuyant sur le cahier des charges fonctionnelles. Pour ce faire j'utilise le logiciel en ligne Draw.io. Celui-ci me permet la création des entités.



Le modèle conceptuel des données a pour but de décrire de façon formelle les données qui seront utilisées par le système d'information (modéliser la sémantique des informations). Il s'agit donc d'une représentation des données, facilement compréhensible, permettant de décrire le système d'information à l'aide d'entités.

Conceptual Data Model

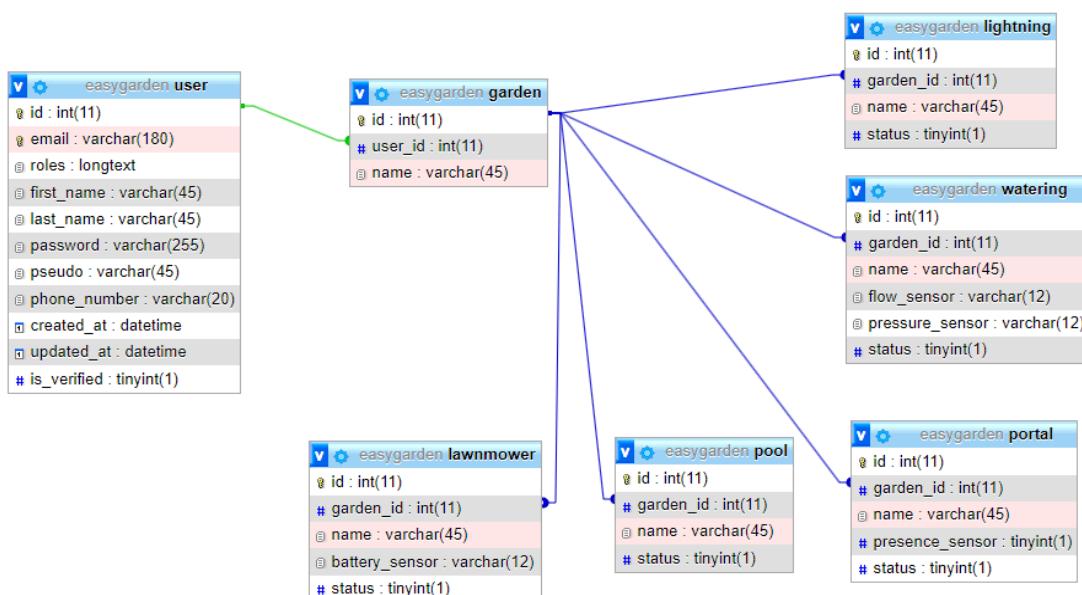


Je crée aussi le modèle logique de données relationnelles (MLDR) grâce à MySQLWorkbench.

Le modèle logique de données permet de modéliser la structure selon laquelle les données seront stockées dans la future base de données et ce, sans faire référence à un langage de programmation. Il s'agit donc de préciser le type de données utilisées lors des traitements. Sa création est déterminée par le MCD :

- Une entité du MCD devient une relation, c'est-à-dire une table.
- Son identifiant devient la clé primaire de la relation.
- Les autres propriétés deviennent les attributs de la relation.

Une association de type 0,n pour l'entité User se traduit par la création d'une clé étrangère dans la relation correspondante à l'entité côté n c'est-à-dire dans l'entité Garden. De même l'entité Garden possédant une relation 0,n avec les entités des équipements, une clé étrangère référençant l'entité Garden sera créée dans chaque entité correspondante des équipements.



Je vais maintenant installer ApiPlatform avec la commande:

`composer req/api`

Ceci étant réalisé je peux donc créer mes entités dans Symfony ainsi que leurs relations avec l'aide du Maker Bundle installé en premier lieu.

`php bin/console make:user`

`php bin/console make:entity`

Lors de la création de nos entités le Maker va nous demander de stipuler si l'on souhaite exposer ou non cette classe comme une ressource de l'API.

```
$ Mark this class as an API Platform resource (expose a CRUD API for it) (yes/no)
[no]:
> yes
```

Une fois terminé nous constatons que l'annotation `@[ApiResource()]` est bien présente dans notre entité. Cela a aussi pour conséquence d'avoir ajoutée cette dernière à notre documentation.

The screenshot shows the "Garden" section of the API documentation. It lists the following operations:

- GET /api/gardens**: Retrieves the collection of Garden resources.
- POST /api/gardens**: Creates a Garden resource.
- GET /api/gardens/{id}**: Retrieves a Garden resource.
- PUT /api/gardens/{id}**: Replaces the Garden resource.
- DELETE /api/gardens/{id}**: Removes the Garden resource.
- PATCH /api/gardens/{id}**: Updates the Garden resource.

J'ai ensuite déclaré des groupes de normalisation et de dénormalisation, ainsi que des opérations sur les collections. Il convient de déclarer ces derniers avant la classe elle-même et ce dans l'annotation `@[ApiResource()]`

```
Pool.php
...
api > src > Entity > Pool.php > ...
1  <?php
2
3  namespace App\Entity;
4
5  use Doctrine\ORM\Mapping as ORM;
6  use App\Repository\PoolRepository;
7  use ApiPlatform\Core\Annotation\ApiFilter;
8  use ApiPlatform\Core\Annotation\ApiResource;
9  use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\BooleanFilter;
10 use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\OrderFilter;
11 use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;
12 use Symfony\Component\Serializer\Annotation\Groups;
13 use Symfony\Component\Validator\Constraints as Assert;
14
15 #[ORM\Entity(repositoryClass: PoolRepository::class)]
16 #[ApiResource(
17     normalizationContext: ['groups' => ['read:Pool']],
18     denormalizationContext: ['groups' => ['write:Pool']],
19     collectionOperations: [ 'get' => ['normalization_context' => ['groups' => ['read:Pool']]],
20     |           |           |           |           'post' => ['denormalization_context' => ['groups']] ],
21     order: ['status' => 'DESC']])
22 #[ApiFilter(BooleanFilter::class, properties: ['status'])]
23 #[ApiFilter(OrderFilter::class, properties: ['name'])]
24 #[ApiFilter(SearchFilter::class, properties: ['name' => 'partial'])]
```

Et d'exposer les propriétés avec les «groups» que nous souhaitons, pour ce faire l'annotation `##[Groups()]` est mise à disposition.

```
api > src > Entity > Pool.php > Pool > getName
26 class Pool
27 [
28     #[ORM\Id]
29     #[ORM\GeneratedValue]
30     #[ORM\Column(type: 'integer')]
31     #[Groups(['read:User',
32             'read:Garden',
33             'read:Pool'])]
34     private $id;
35
36     #[ORM\Column(type: 'string', length: 45)]
37     #[Assert\NotBlank]
38     #[Assert\Regex(
39         pattern: '/^@[a-zA-ZàáâãçéèéííîñôôôÀÁÃÇÉÈÉÍÍÎÑÔÔÔ-9][a-zàáâãçéèéííîñôôô]+',
40         match: false,
41         message: 'The pool name cannot contain some special character',
42     )]
43     #[Assert\Regex(
44         pattern: '/\w{3,20}\$/',
45         match: false,
46         message: 'The pool name should be between 3 and 20 characters',
47     )]
48     #[Groups(['read:User',
49             'read:Garden',
50             'read:Pool',
51             'write:Pool'])]
52     private $name;
53
54     #[ORM\Column(type: 'boolean', nullable:true)]
55     #[Groups(['read:User',
56             'read:Garden',
57             'read:Pool',
58             'write:Pool'])]
59     private $status;
60
61     #[ORM\JoinColumn(nullable: false)]
62     #[ORM\ManyToOne(targetEntity: Garden::class, inversedBy: 'pool')]
63     #[Groups(['read:Pool'])]
64     private $garden;
```

La classe met de plus à disposition le getter et setter de chaque propriété.

```
Pool.php X
api > src > Entity > Pool.php > Pool
66     public function getId(): ?int
67     {
68         return $this->id;
69     }
70
71     public function getName(): ?string
72     {
73         return $this->name;
74     }
75
76     public function setName(string $name): self
77     {
78         $this->name = $name;
79
80         return $this;
81     }
82
83     public function getStatus(): ?bool
84     {
85         return $this->status;
86     }
87
88     public function setStatus(bool $status): self
89     {
90         $this->status = $status;
91
92         return $this;
93     }
94
95     public function getGarden(): ?Garden
96     {
97         return $this->garden;
98     }
99
100    public function setGarden(?Garden $garden): self
101    {
102        $this->garden = $garden;
103
104        return $this;
105    }
106 }
```

Pour ce qui est de l'authentification et de la sécurité je vais me servir du LexikJWTBundle. JWT (JSON Web Tokens) est un standard ouvert défini dans la RFC 75191. Il permet l'échange

sécurisé de jetons entre plusieurs parties. Cette sécurité de l'échange se traduit par la vérification de l'intégrité des données à l'aide d'une signature numérique. C'est actuellement l'un des moyens les plus pratiques et utilisé pour sécuriser l'accès à une API. En effet, comme chaque jeton à une durée de vie. Même s'ils sont compromis, ces jetons seront considérés comme "expirés" après un certain temps. Bien sûr, la génération d'une nouvelle clé publique et privée invalide tous les jetons précédemment créés. Nous allons générer des jetons JWT pour différents rôles d'utilisateurs ; puis, nous allons les utiliser pour accéder à des ressources exposées par API Platform (des entités Doctrine). Ces ressources vont exposer des informations différentes selon les droits qu'embarquent les jetons.

Afin d'installer le bundle il faut lancer la commande suivante:

```
composer require lexik/jwt-authentication-bundle
```

Ensuite il faut générer la clé privée et la clé publique (après avoir installé openssl en local sur son ordinateur):

```
openssl genrsa -out config/jwt/private.pem 4096
```

```
openssl pkey -in config/jwt/private.pem -out config/jwt/public.pem -pubout
```

SSL a été à l'origine développé par Netscape. OpenSSL est une version libre du protocole SSL (Secure Sockets Layer) et TLS (Transport Layer Security). Il permet de crypter toutes les données échangées entre le client et le serveur de façon à ce que seul le serveur puisse décrypter ce qui vient du client et inversement. Un éventuel pirate ne peut pas, dans un temps raisonnable, décrypter les informations.

Il est nécessaire de mettre en place un peu de configuration dans différents fichiers du dossier config Symfony.

```
lexik_jwt_authentication:
    secret_key: '%env(resolve:JWT_SECRET_KEY)%'
    public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
    pass_phrase: '%env(JWT_PASSPHRASE)%'
    token_ttl: "%env(JWT_TTL)%"
    user_identity_field: email
```

Le fichier `lexik_jwt_authentication.yaml` où l'on définit la propriété de la classe User qui servira d'identifiant.

```
controllers:
    resource: ../src/Controller/
    type: annotation

kernel:
    resource: ../src/Kernel.php
    type: annotation

login_check:
    path: /api/login_check
    methods: ['GET', 'POST']
```

Le fichier routes.yaml où l'on définit quelles verbes seront autorisés pour requêter l'API ainsi que l'URI (Uniform Ressource Identifier) pour vérifier les informations de connexion.

```

security.yaml

api > config > packages > security.yaml

12 providers:
13     # used to reload user from session & other features (e.g. switch_user)
14     app_user_provider:
15         entity:
16             class: App\Entity\User
17             property: email
18
19 firewalls:
20     registration:
21         pattern: ^/api/users
22         stateless: true
23         methods: [POST]
24     login:
25         pattern: ^/api/login
26         stateless: true
27         provider: app_user_provider
28         json_login:
29             check_path: /api/login_check
30             username_path: email
31             success_handler: lexik_jwt_authentication.handler.authentication_success
32             failure_handler: lexik_jwt_authentication.handler.authentication_failure
33     api:
34         pattern: ^/api
35         stateless: true
36         jwt: ~
37
38 # Easy way to control access for large sections of your site
39 # Note: Only the *first* access control that matches will be used
40 access_control:
41     - { path: ^/api/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
42     - { path: ^/api/users, roles: PUBLIC_ACCESS,
43         methods: [POST] }
44     - { path: ^/api/users, roles: IS_AUTHENTICATED_FULLY,
45         methods: [GET, PUT, DELETE] }
46     - { path: ^/api/gardens, roles: IS_AUTHENTICATED_FULLY }
47     - { path: ^/api/lawnmowers, roles: IS_AUTHENTICATED_FULLY }
48     - { path: ^/api/lightnings, roles: IS_AUTHENTICATED_FULLY }
49     - { path: ^/api/pools, roles: IS_AUTHENTICATED_FULLY }
50     - { path: ^/api/portals, roles: IS_AUTHENTICATED_FULLY }
51     - { path: ^/api/waterings, roles: IS_AUTHENTICATED_FULLY }
52
53 role_hierarchy:
54     ROLE_ADMIN: ROLE_USER

```

Et enfin le fichier security.yaml où l'on définit les options du firewall, le provider de l'utilisateur (dont je parlerai plus loin), les access_control qui permettent de définir le niveau d'authentification requis pour obtenir une réponse de l'API ainsi que les verbes autorisés sur chaque URI et le rôle_hierarchy qui définit que le rôle Admin possèdera en plus de ses droits ceux d'un utilisateur normal.

Je vais donc maintenant aller modifier le fichier .env de Symfony afin de définir le DATABASE_URL et pouvoir ensuite effectuer ma migration.

```

# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://root:@127.0.0.1:3306/EasyGarden?serverVersion=mariadb-10.5.8"
# DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"
##< doctrine/doctrine-bundle ###

```

Je sais les commandes suivantes afin de mapper ma BDD :

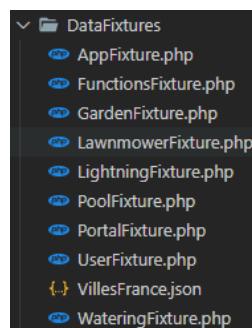
php bin/console doctrine:database:create

```
php bin/console make:migration --no-interaction
```

```
php bin/console doctrine:migrations:migrate --no-interaction
```

La base de données ainsi que ses tables ont bien été générées.

Afin de pouvoir me connecter avec un utilisateur et hydrater mes tableaux d'IHM avec de la data je vais donc créer un jeu de fixtures en utilisant un fichier séparé pour chaque entité, un fichier de fonctions qui servira entre autres à définir des noms d'équipements de manière aléatoire dans un tableau de données. Et de plus le fichier de toutes les villes de France afin que chaque utilisateur se voit attribué aléatoirement une commune de résidence réelle pour son nom de jardin.



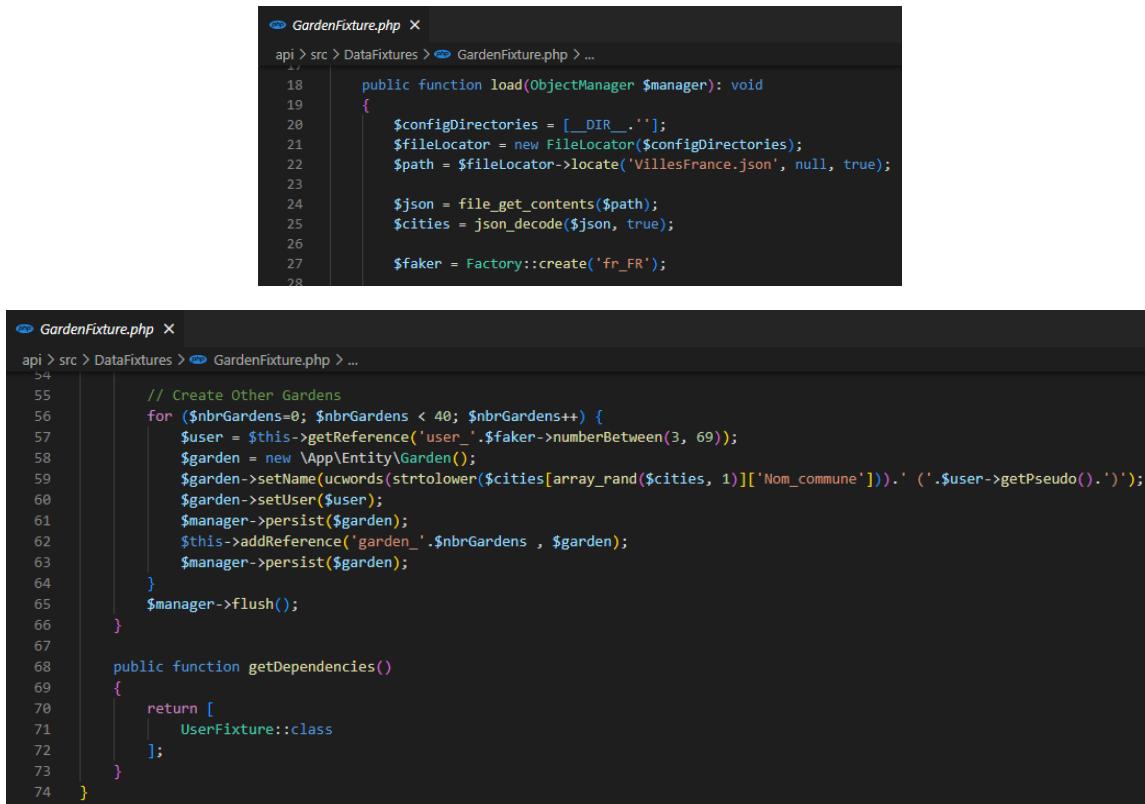
La fixture User où je «set» un utilisateur Admin et un utilisateur me représentant. J'y ajoute des références pour les clés étrangères afin de ne pas créer de conflits lors du load de la fixture.

```
  UserFixture.php ×
api > src > DataFixtures > UserFixture.php > ...
12  class UserFixture extends Fixture
13  {
14      private UserPasswordHasherInterface $hasher;
15
16      public function __construct(UserPasswordHasherInterface $hasher)
17      {
18          $this->hasher = $hasher;
19      }
20
21      public const USER1_REFERENCE = 'user1_';
22      public const USER2_REFERENCE = 'user2_';
23
24      public function load(ObjectManager $manager): void
25      {
26          $configDirectories = [__DIR__.''];
27          $fileLocator = new FileLocator($configDirectories);
28          $fileLocator->locate('FunctionsFixture.php', null, false);
29
30          $faker = Factory::create('fr_FR');
31
32          // User ADMIN
33          $admin = new User();
34          $admin->setFirstName('prenomADMIN');
35          $admin->setLastName('nomADMIN');
36          $admin->setPseudo('pseudoADMIN');
37          $admin->setPassword($this->hasher->hashPassword($admin, 'Admin!33kzo'));
38          $admin->setRoles(['ROLE_ADMIN']);
39          $admin->setEmail('admin@protonmail.fr');
40          $admin->setPhoneNumber('05 12 25 48 71');
41          $admin->setCreatedAt(new \DateTimeImmutable());
42          $admin->setIsVerified(true);
43          $manager->persist($admin);
44
45          // User Manu
46          $user1 = new User();
47          $user1->setFirstName('Emmanuel');
48          $user1->setLastName('Lefevre');
49          $user1->setPseudo('Manu');
50          $user1->setPassword($this->hasher->hashPassword($user1, 'Darka!33kzo'));
51          $user1->setRoles(['ROLE_USER']);
52          $user1->setEmail('emmanuel@protonmail.com');
53          $user1->setPhoneNumber('06 45 91 23 07');
54          $user1->setCreatedAt(new \DateTimeImmutable());
55          $user1->setIsVerified(true);
56          $manager->persist($user1);
57          $this->addReference(self::USER1_REFERENCE , $user1);
```

Je crée également d'autres utilisateurs pour l'interface d'administration. J'utilise la bibliothèque Faker pour la création de certaines datas.

```
  UserFixture.php ×
api > src > DataFixtures > UserFixture.php > ...
73      // Create Other Users
74      for ($nbrUsers=0; $nbrUsers < 70; $nbrUsers++) {
75          $user = new User();
76          $user->setFirstName($fn=$faker->firstname());
77          $user->setLastName($ln=$faker->lastname());
78          $user->setPseudo($fn.mt_rand(0, 100));
79          $user->setPassword($this->hasher->hashPassword($user, $faker->password()));
80          $user->setRoles(['ROLE_USER']);
81          $user->setEmail($fn.".".$ln.'@'.emailData());
82          $user->setPhoneNumber($faker->mobileNumber());
83          $user->setCreatedAt(new \DateTimeImmutable());
84          $user->setIsVerified(mt_rand(0, 1));
85          $manager->persist($user);
86          $this->addReference('user_'.$nbrUsers , $user);
87      }
88      $manager->flush();
```

La fixture Garden où j'attribue de manière aléatoire une ville comme nom de jardin ainsi que la fonction `getDependencies()` qui sert à indiquer dans quel ordre les fichiers de fixtures doivent être chargé. En effet les fichiers de fixtures se charge par ordre alphabétique et une fixture possédant une clé étrangère ne peut être créée si la valeur de l'entité à laquelle elle fait référence n'a pas déjà été créée elle-même, cela provoquerait une erreur.

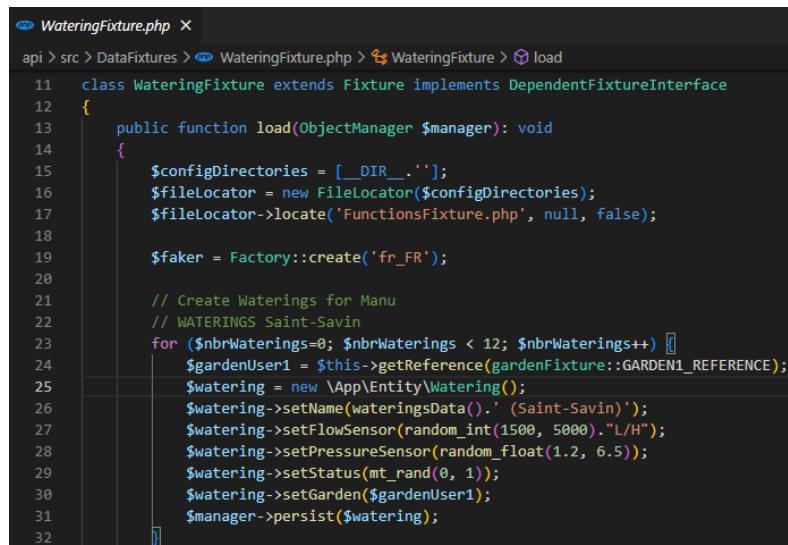


```

    ↗ GardenFixture.php ×
api > src > DataFixtures > ↗ GardenFixture.php > ...
18     public function load(ObjectManager $manager): void
19     {
20         $configDirectories = [__DIR__.''];
21         $fileLocator = new FileLocator($configDirectories);
22         $path = $fileLocator->locate('VillesFrance.json', null, true);
23
24         $json = file_get_contents($path);
25         $cities = json_decode($json, true);
26
27         $faker = Factory::create('fr_FR');
28
54
55     // Create Other Gardens
56     for ($nbrGardens=0; $nbrGardens < 40; $nbrGardens++) {
57         $user = $this->getReference('user_'.$faker->numberBetween(3, 69));
58         $garden = new \App\Entity\Garden();
59         $garden->setName(ucwords(strtolower($cities[array_rand($cities, 1)]['Nom_commune'])) . ('.$user->getPseudo().'));
60         $garden->setUser($user);
61         $manager->persist($garden);
62         $this->addReference('garden_'.$nbrGardens, $garden);
63         $manager->persist($garden);
64     }
65     $manager->flush();
66 }
67
68     public function getDependencies()
69     {
70         return [
71             UserFixture::class
72         ];
73     }
74 }
```

Il convient donc de load la fixture User en premier, puis la fixture Garden et enfin les fixtures pour les équipements.

Pour finir avec cela voici la fixture de l'équipement arrosage ainsi que quelques fonctions dont elle bénéficie.



```

    ↗ WateringFixture.php ×
api > src > DataFixtures > ↗ WateringFixture.php > ↗ WateringFixture > ↗ load
11     class WateringFixture extends Fixture implements DependentFixtureInterface
12     {
13         public function load(ObjectManager $manager): void
14         {
15             $configDirectories = [__DIR__.''];
16             $fileLocator = new FileLocator($configDirectories);
17             $fileLocator->locate('FunctionsFixture.php', null, false);
18
19             $faker = Factory::create('fr_FR');
20
21             // Create Waterings for Manu
22             // WATERINGS Saint-Savin
23             for ($nbrWaterings=0; $nbrWaterings < 12; $nbrWaterings++) [
24                 $gardenUser1 = $this->getReference(gardenFixture::GARDEN1_REFERENCE);
25                 $watering = new \App\Entity\Watering();
26                 $watering->setName(wateringsData().' (Saint-Savin)');
27                 $watering->setFlowSensor(random_int(1500, 5000)."L/H");
28                 $watering->setPressureSensor(random_float(1.2, 6.5));
29                 $watering->setStatus(mt_rand(0, 1));
30                 $watering->setGarden($gardenUser1);
31                 $manager->persist($watering);
32             ]
33         }
34     }
```

```

api > src > DataFixtures > FunctionsFixture.php > poolsData
1  <?php
2
3  // Check string surrounded by parenthesis
4  function stringWithoutParenthesis($str) {
5      $pseudo = substr($str, ($p = strpos($str, '(')+1), strpos($str, ')')-$p);
6  }
7
8  // Check string surrounded by parenthesis and concatenate parenthesis around
9  function stringWithParenthesis($str) {
10     $pseudo = substr($str, ($p = strpos($str, '(')+1), strpos($str, ')')-$p);
11     return (' ('.$pseudo.')');
12 }
13
14 // Generate an aleatory float with two decimals in the interval $min/$max
15 function random_float ($min,$max) {
16     $value = ($min+lcg_value())*(abs($max-$min));
17     return (sprintf("%01.2f",$value)."bars");
18 }
19
20 // @Email.com array
21 function emailData()
22 {
23     $data = ['gmail.com','outlook.fr','yahoo.fr','protonmail.com','orange.fr','live.fr','laposte.net',
24     'icloud.com','sfr.fr','free.fr','mailo.com'];
25     $value = array_rand(array_flip($data), 1);
26     return $value;
27 }
28
29 // Waterings equipements array
30 function wateringsData()
31 {
32     $data = ['Chemin Accès','Allée','Bassin','Enrochemement','Massif Brasero','Bassin','Arrière Maison','Haie Bambous',
33     'Piscine','Massif','Terrasse','Massif Minéral','Pergola','Potager','Tomates','Salade','Courges','Secteur Devant',
34     'Secteur Arrière','Haie','Potiches','Goutte à Goutte','Jardinière'
35 ];
36     $value = array_rand(array_flip($data), 1);
37     return $value;
38 }

```

Le fichier composer.json concède de définir manuellement les versions exactes des bundles ou la version PHP de notre projet Symfony.

```

api > {} composer.json > {} config > {} allow-plugins > symfony/runtime
1  {
2      "type": "project",
3      "license": "proprietary",
4      "minimum-stability": "stable",
5      "prefer-stable": true,
6      "require": {
7          "php": ">=8.0.2",
8          "ext-ctype": "*",
9          "ext-iconv": "*",
10         "api-platform/core": "^2.6",
11         "doctrine/annotations": "^1.13",
12         "doctrine/doctrine-bundle": "2.5",
13         "doctrine/doctrine-migrations-bundle": "3.2",
14         "doctrine/orm": "2.11",
15         "fakerphp/faker": "1.19",
16         "lexik/jwt-authentication-bundle": "2.14",
17         "nelmio/cors-bundle": "2.2",
18         "phpdocumentor/reflection-docblock": "5.3",
19         "phpstan/phpdoc-parser": "1.2",
20         "symfony/asset": "6.0.*",
21         "symfony/console": "6.0.*",
22         "symfony/dotenv": "6.0.*",
23         "symfony/expression-language": "6.0.*",
24         "symfony/flex": "2",
25         "symfony/framework-bundle": "6.0.*",
26         "symfony/property-access": "6.0.*",
27         "symfony/property-info": "6.0.*",
28         "symfony/proxy-manager-bridge": "6.0.*",
29         "symfony/rate-limiter": "6.0.*",
30         "symfony/runtime": "6.0.*",
31         "symfony/security-bundle": "6.0.*",
32         "symfony/serializer": "6.0.*",
33         "symfony/twig-bundle": "6.0.*",
34         "symfony/validator": "6.0.*",
35         "symfony/yaml": "6.0.*"
36     },
37     "require-dev": {
38         "doctrine/doctrine-fixtures-bundle": "3.4",
39         "symfony/maker-bundle": "1.37",
40         "symfony/stopwatch": "6.0.*",
41         "symfony/web-profiler-bundle": "6.0.*"
42     }
},

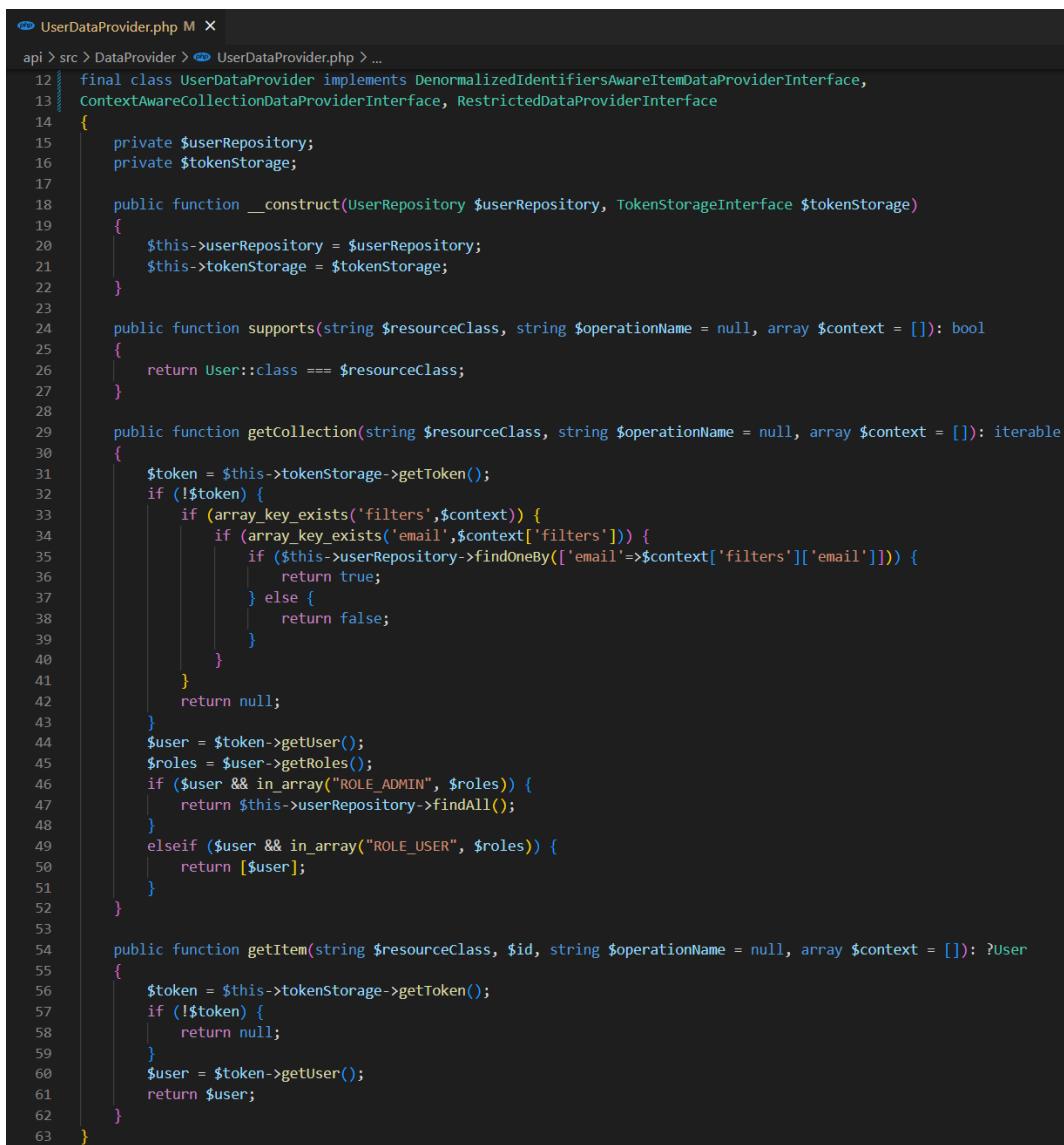
```

Pour que le composant de sécurité comprenne que notre utilisateur vient de la base de données, il faut le lui préciser en créant ce que l'on appelle un provider (un fournisseur de données utilisateurs). Le provider permet au firewall d'interroger une collection d'utilisateurs, c'est une sorte de base de tous les utilisateurs avec les mots de passe, j'utiliserai ici le type entity. Cette partie se configure dans le fichier security.yaml.

```

providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email

```



```

UserDataProvider.php M ×
api > src > DataProvider > UserDataProvider.php > ...
12  final class UserDataProvider implements DenormalizedIdentifiersAwareItemDataProviderInterface,
13  ContextAwareCollectionDataProviderInterface, RestrictedDataProviderInterface
14  {
15      private $userRepository;
16      private $tokenStorage;
17
18      public function __construct(UserRepository $userRepository, TokenStorageInterface $tokenStorage)
19      {
20          $this->userRepository = $userRepository;
21          $this->tokenStorage = $tokenStorage;
22      }
23
24      public function supports(string $resourceClass, string $operationName = null, array $context = []): bool
25      {
26          return User::class === $resourceClass;
27      }
28
29      public function getCollection(string $resourceClass, string $operationName = null, array $context = []): iterable
30      {
31          $token = $this->tokenStorage->getToken();
32          if (!$token) {
33              if (array_key_exists('filters', $context)) {
34                  if (array_key_exists('email', $context['filters'])) {
35                      if ($this->userRepository->findOneBy(['email' => $context['filters']['email']])) {
36                          return true;
37                      } else {
38                          return false;
39                      }
40                  }
41              }
42          }
43          return null;
44      }
45      $user = $token->getUser();
46      $roles = $user->getRoles();
47      if ($user && in_array("ROLE_ADMIN", $roles)) {
48          return $this->userRepository->findAll();
49      }
50      elseif ($user && in_array("ROLE_USER", $roles)) {
51          return [$user];
52      }
53
54      public function getItem(string $resourceClass, $id, string $operationName = null, array $context = []): ?User
55      {
56          $token = $this->tokenStorage->getToken();
57          if (!$token) {
58              return null;
59          }
60          $user = $token->getUser();
61          return $user;
62      }
63  }

```

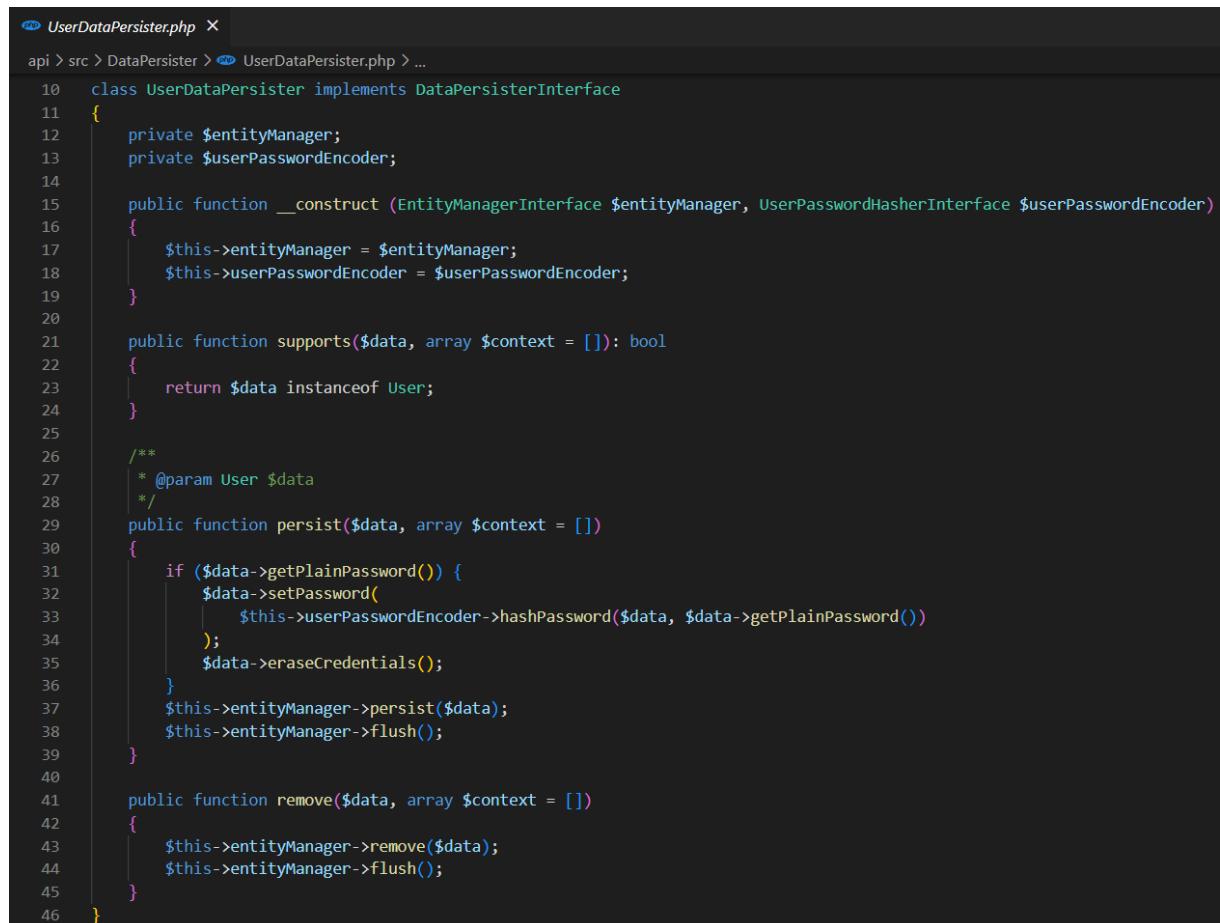
Grâce au groupe que j'ai crée avec ApiPlatform cela me permet de récupérer un fichier JSON contenant toutes les datas de l'utilisateur lorsque j'appelle l'URI /api/users en méthode GET.

Chaque fois que nous utilisons une méthode POST ou PUT, après qu'ApiPlatform ait déserialisé les données dans un objet et l'ait validé, il essaie de persister ces données. Habituellement, cela signifie que nous enregistrons un objet entité dans la base de données via Doctrine. J'ai donc créé un UserDataPersister car j'avais besoin d'encoder le mot de passe

en clair (propriété plainPassword) et de le définir dans la propriété password de l'entité User.

Pour créer ce persisteur de données, j'ai donc implémenté le DataPersisterInterface de Symfony et ajouté une supports() méthode indiquant que nous prenons en charge les objets User.

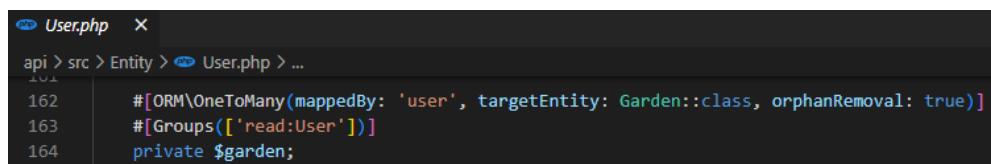
Ayant fait cela, je deviens 100% responsable de la persistance des objets User. Ce qui veut dire que le persisteur de données normal de Doctrine ne sera plus appelé pour les objets User, notre persisteur a la priorité. J'ai donc du appeler ensuite les méthodes persist() et flush(), ainsi que la méthode remove() qui me permet de supprimer un utilisateur ainsi que toutes les données lui étant associées (j'utilise donc l'option orphanRemoval : true de l'attribut #[ORM]).



```

UserDataPersister.php ×
api > src > DataPersister > UserDataPersister.php > ...
10 class UserDataPersister implements DataPersisterInterface
11 {
12     private $entityManager;
13     private $userPasswordEncoder;
14
15     public function __construct (EntityManagerInterface $entityManager, UserPasswordHasherInterface $userPasswordEncoder)
16     {
17         $this->entityManager = $entityManager;
18         $this->userPasswordEncoder = $userPasswordEncoder;
19     }
20
21     public function supports($data, array $context = []): bool
22     {
23         return $data instanceof User;
24     }
25
26     /**
27      * @param User $data
28      */
29     public function persist($data, array $context = [])
30     {
31         if ($data->getPlainPassword()) {
32             $data->setPassword(
33                 $this->userPasswordEncoder->hashPassword($data, $data->getPlainPassword())
34             );
35             $data->eraseCredentials();
36         }
37         $this->entityManager->persist($data);
38         $this->entityManager->flush();
39     }
40
41     public function remove($data, array $context = [])
42     {
43         $this->entityManager->remove($data);
44         $this->entityManager->flush();
45     }
46 }

```



```

User.php ×
api > src > Entity > User.php > ...
161
162     #[ORM\OneToMany(mappedBy: 'user', targetEntity: Garden::class, orphanRemoval: true)]
163     #[Groups(['read:User'])]
164     private $garden;

```

Afin de tester le fonctionnement du back-end j'utilise un logiciel nommé Postman. Celui-ci permet de simuler un front-end en se connectant sur le Endpoint de l'API.



Certaines requêtes HTTP ont de l'intérêt que parce qu'elles envoient de la donnée au serveur via le request body de la requête. C'est le cas des requêtes POST, PUT et PATCH. Les données du body à envoyer au serveur via Postman peuvent être de différents formats mais le plus utilisé est le format JSON (celui utilisé dans notre cas). Les requêtes HTTP peuvent également comporter des request headers. Il s'agit de données transmises en en-tête de chaque requête ayant différentes utilités: dans certains cas, il faut spécifier certains headers, comme un Token d'autorisation ou d'authentification.

Méthode GET d'authentification afin de générer le token:

```

1 {"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiIzI1NiJ9.eyJpYXQiOjE2NTkzMjZmUsImV4cCI6MTY1OTA1ODMzNwicm9sZXMI01siUk9MRV9VU0V5I10sImVtYWlsIjoizWlTY5iZwAcHJvG9ubWFpbC5jb20ifQ.1_8h3RqJ6nZjrhsspsmqjGZdis9hbd576yZEosk11v8052lgm11etux81FV/Vteo8ts1lqeixxpdiCvJ1-Tj3AbPVn8jazs6FvvVcI0Q-JEA031QFK51g5sEA-Rwd63Yp6kzNCJdbcC4ZPSymB4GcBE3Zxh@HayShdHtZjkpcEWkElwdIt9h_Z-Ij5vPUscn3CaNRLyCKka6TxLGeBzFuMpB6iw-1HnaMwJv7zb1c3ovAs9gGLF4B_FdHAGQq26-vPQTl80IoRwtPt4aTfYfjod9RE7qUt1Y-QTp1uUr3Gfvceg2NFV0MyjxE_prk-dxwBRU-19dy3I-TMUFgJ-2KcmPdUzIdzo4IuxMst1EeUzDTo1_Ry70GXjFXUKK54pg3MOK41mCoj_S3cdU_znbw3f02WARebHml_06Gy7UWJXf8PgfLhXhwlg47zKSVNxf95R7x266KCFU_9iuw_9drp6TjvZINv8E8FFIXFvQbPFUrtR7PmG29ukecxLjsS_5_8Pz4t4w6e4p151lGJD6W609jRfnYyoggGfy3CDU9E4FZtIZF8brRs0b1ghmU1z5d-vFwHK7ULMB1WdqKVQjZ50JhvJehszxC1eCaW71k-de0rCYsBmCFIP0wn15rnAs9uF6rZNIMj3P30W4"]

```

Méthode GET afin de récupérer les données utilisateurs grâce au token:

```

1 {
2   "id": 2,
3   "email": "emmanuel@protonmail.com",
4   "firstName": "Emmanuel",
5   "lastName": "Lefevre",
6   "pseudo": "Manu",
7   "phoneNumber": "+33 65 91 23 07",
8   "createdAt": "2022-07-25T20:33:13+00:00",
9   "isVerified": false,
10   "garden": [
11     {
12       "id": 1,
13       "name": "Saint-Savin",
14       "user": "/api/users/2",
15       "lawnmower": [
16         {
17           "id": 1,
18           "name": "Tondeuse (Saint-Savin)",
19           "batterySensor": "2k",
20           "status": true
21         }
22       ],
23       "lightning": [
24         {
25           "id": 1,
26         }
27     ]
28   ]
29 }

```

Méthode PUT afin d'éteindre la tondeuse:

The screenshot shows a POSTMAN interface. The URL is `localhost:8000/api/lawnmowers/1`. The Body tab is selected, showing a JSON object with a single key `"status": false`. The response tab shows a status of `200 OK`.

```

1
2   "status": false
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
  
```

5.3 Front-end

Pour commencer l'utilisation d'un framework front-end tel qu'Angular rendrait illogique l'utilisation du moteur de template intégré à Sf4 (Twig).

J'ouvre un terminal dans VsCode et me place dans le répertoire du projet:
 C:\xampp\htdocs\EasyGardenCDA puis tape la commande suivante:

`ng new spa`

Et j'installe différentes librairies nécessaires à mon développement:

- Font Awesome:

`npm install @fortawesome/fontawesome-svg-core`

`npm install @fortawesome/angular-fontawesome@0.10x`

`npm install @fortawesome/free-solid-svg-icons`

`npm install @fortawesome/free-brands-svg-icons`

- Angular material:

`ng add @angular/material`

- Animate.css

`npm i animate.css`

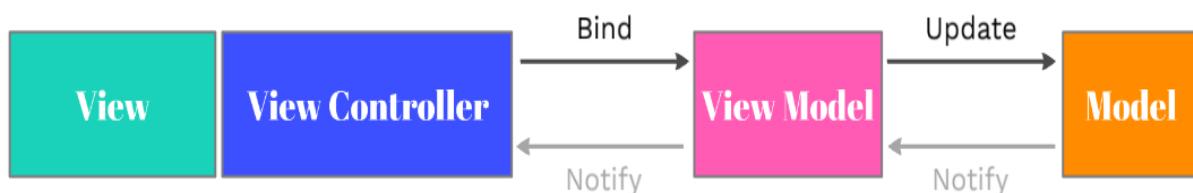
- Ngx-pagination:

`npm i ngx-pagination`

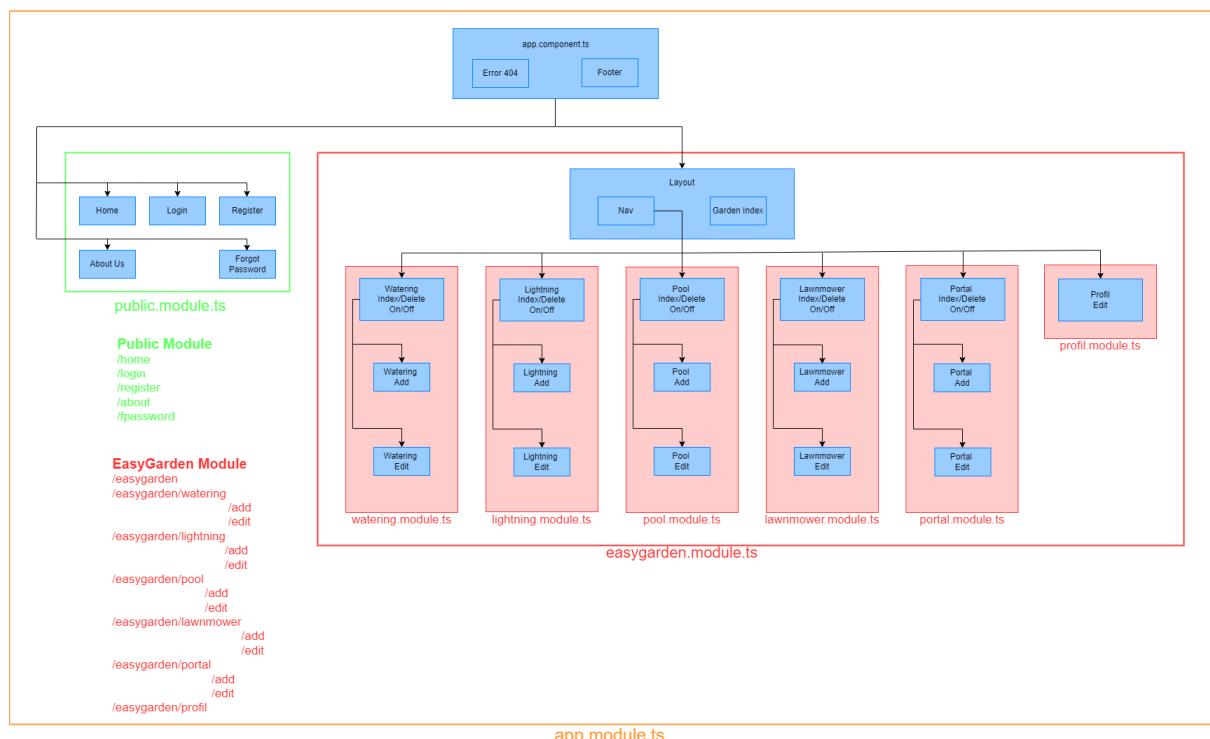
- JWT Decode:

npm i jwt-decode

Angular possède une structure de type MVVM.(Model View ViewModel). Il s'agit d'une variante du modèle MVC. Elle présente quelques différences essentielles, dont la plus importante est la séparation de l'interface utilisateur et de la logique métier. L'autre grande différence est l'utilisation d'un "ViewModel". Le ViewModel est une classe à laquelle l'interface utilisateur se lie. Elle est chargée de coordonner les données envoyées à la vue et les commandes envoyées au contrôleur. Cela permet de découpler la vue des données, et le contrôleur des données et de la vue. Le ViewModel est responsable de la gestion des données et des commandes permettant de les manipuler.



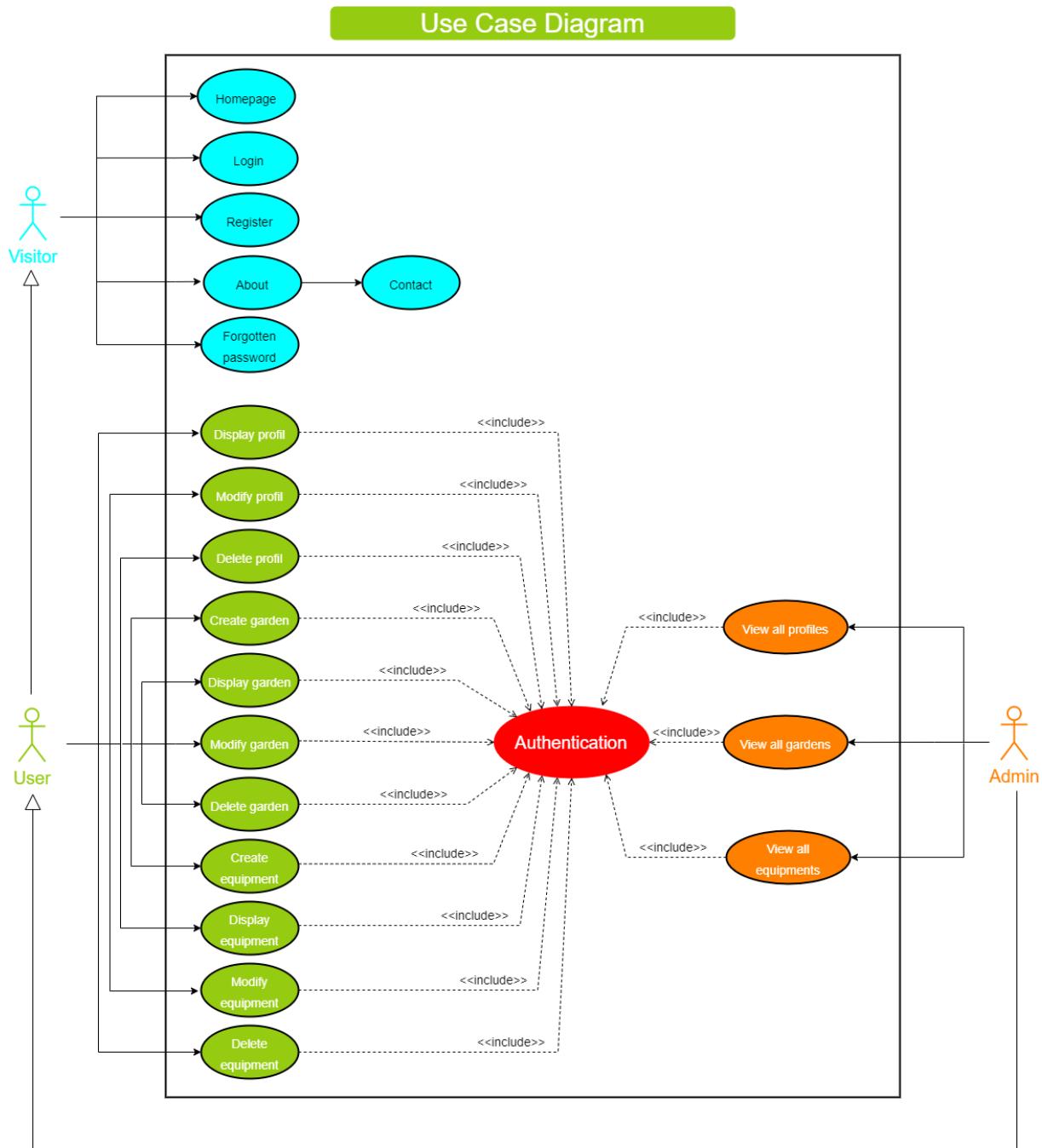
Je vais ici présenter l'architecture de mon projet Angular sous la forme d'un schéma.



Les composants de la partie publique seront accessibles à un utilisateur non connecté tandis que le module EasyGarden, protégé par un guard, ne sera accessible qu'à un utilisateur connecté. Ce même module contiendra quant à lui un module pour chaque entité de notre back-end. Hormis le module public, tout les modules seront chargés en lazy-loading

En fonction du cahier des charges j'ai crée un diagramme de cas d'utilisation. Les diagrammes de cas d'utilisation (DCU) sont des diagrammes UML (Unified Modeling

Language) utilisés pour une représentation du comportement fonctionnel d'un système logiciel. Ils permettent de décrire l'interaction entre l'acteur et le système.



A partir de ce use case diagram j'ai crée plusieurs diagrammes de package. Les diagrammes de package (ou diagramme de paquetages) sont des diagrammes structurels utilisés pour représenter l'organisation et la disposition de divers éléments modélisés sous forme de packages. Un package est un regroupement d'éléments UML apparentés, tels que des diagrammes, des documents, des classes ou même d'autres packages. Tous les éléments du diagramme sont imbriqués dans des packages, qui sont eux-mêmes représentés sous forme de dossiers de fichiers et organisés de manière hiérarchique. Les diagrammes de packages

sont le plus souvent utilisés pour donner un aperçu visuel de l'architecture en couches d'un classifieur UML, tel qu'un système logiciel.

Diagramme de package EasyGarden:

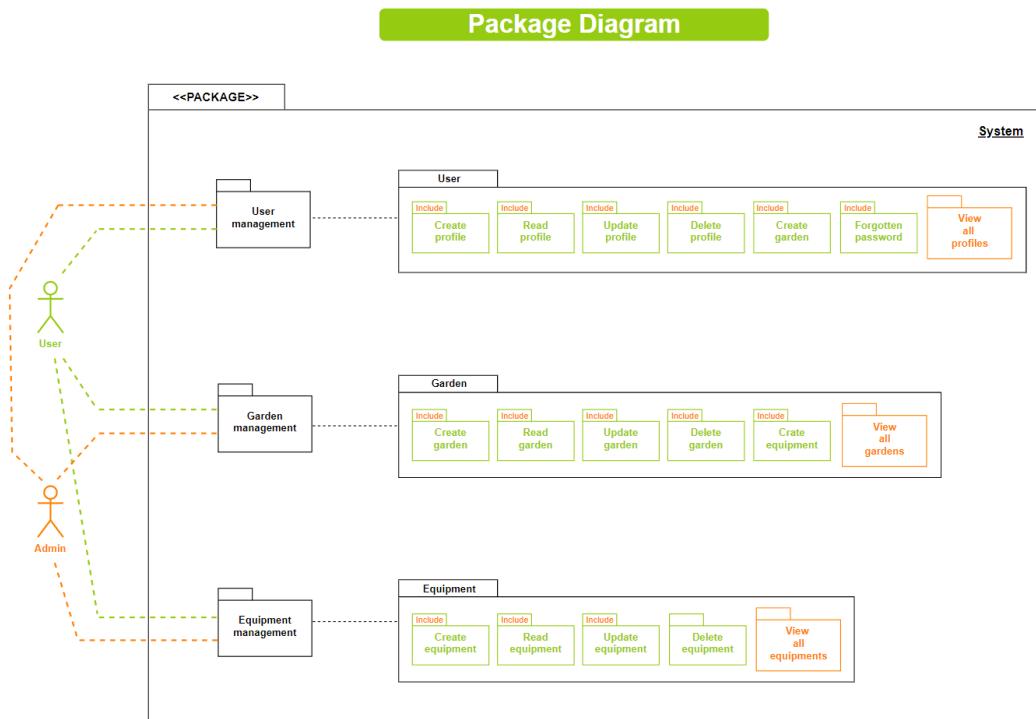


Diagramme de package Garden:

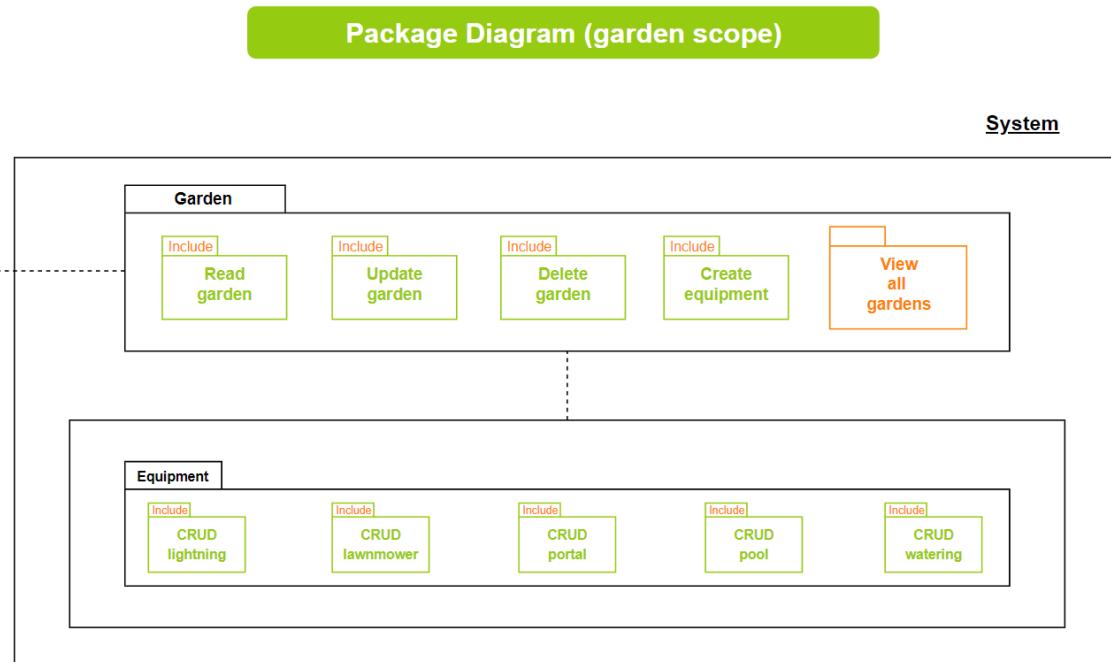
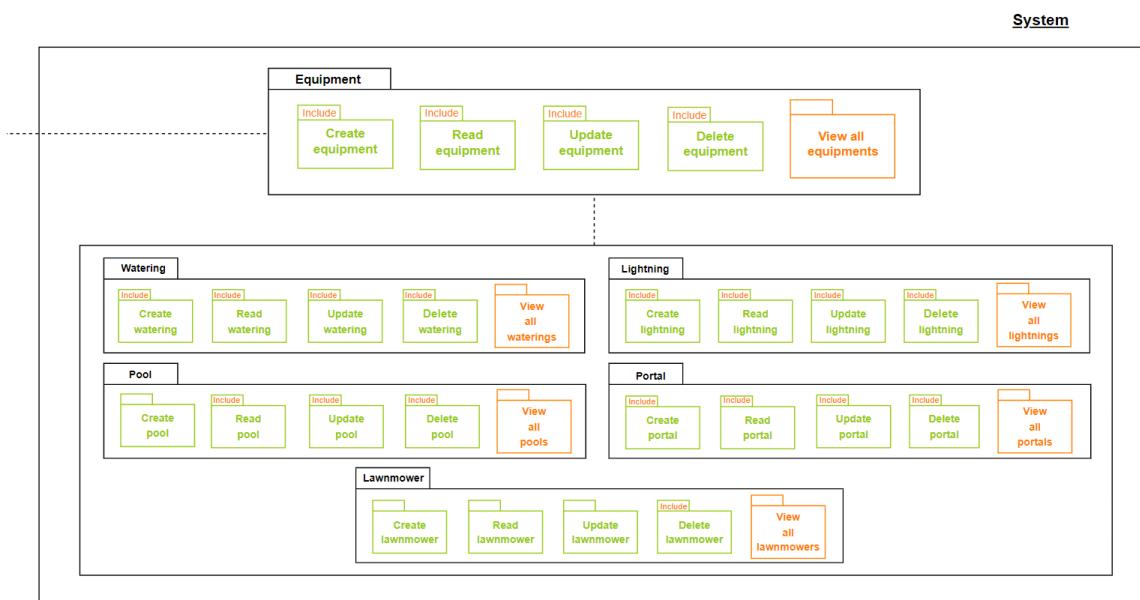


Diagramme de package des équipements:

Package Diagram (equipments scope)



J'ai aussi créer une maquette de zoning pour les IHM de l'application en utilisant le logiciel en ligne Figma.



Figma est une plateforme collaborative pour éditer des graphiques vectoriels et faire du prototypage. Elle permet de concevoir des design systems pour faciliter la création de sites web et d'applications mobiles. C'est une solution à destination des UI et UX designers et des développeurs. L'interface propose de nombreuses fonctionnalités :

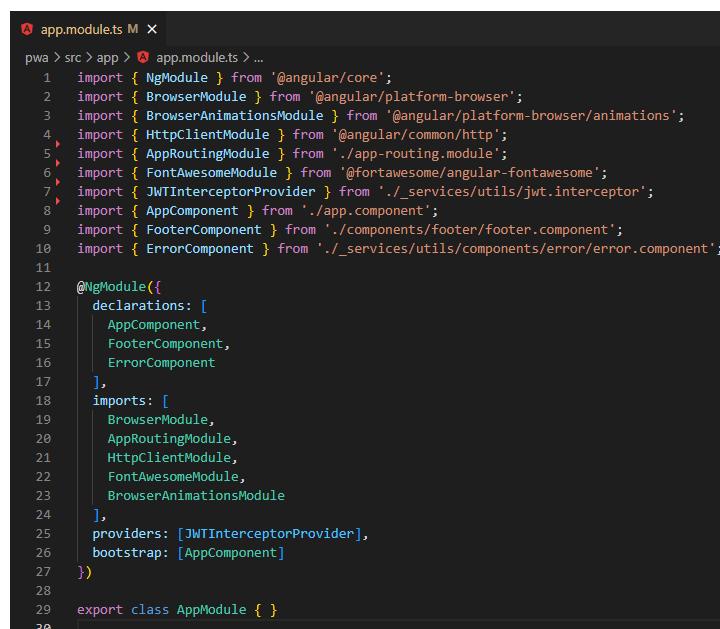
- de design: avec des outils de conception pour le web, des fonctions de mise en page automatique, des plugins pour réduire les tâches répétitives.
- de prototypage: pour tester les concepts très tôt en cours de design.
- de design system : pour concevoir des design cohérents avec des bibliothèques mises à jour en permanence.
- collaborative: pour travailler à plusieurs et en même temps sur un projet, revenir sur une version antérieure si nécessaire ou encore afficher le travail d'un seul collaborateur par exemple.

L'ensemble de ses fonctionnalités est axé sur l'utilisation dans la conception de l'interface utilisateur et de l'expérience utilisateur, en mettant l'accent sur la collaboration en temps réel. Il permet entre autres de créer des composants d'interface réutilisables, des bibliothèques de styles ou bien encore de créer un prototype interactif permettant de tester le fonctionnement des différentes interfaces d'un projet. L'outil offre également des fonctionnalités de partage de maquettes pour permettre de les visualiser en ligne via un un

simple lien, ce qui peut s'avérer très pratique pour un retour client ou lors des présentations lors des réunions. Une dernière fonctionnalité intéressante est la possibilité d'exporter du code: cela permet aux designers de partager ce même code avec les développeurs front-end. Il est également possible d'exporter le code iOS et Android pour les applications mobiles. Le lien pour visualiser la maquette (visuel disponible dans les annexes):

<https://www.figma.com/file/fJDXIoMKpxC9vLCzM3cYwl/EasyGarden-Zoning?node-id=0%3A1>

Angular possède un module de base généré lors de la création du projet. Je déclare les quelques components devant être accessibles dans la partie de l'application ouverte au public non authentifié. J'importe de plus quelques modules importants notamment le JWTInterceptorProvider.

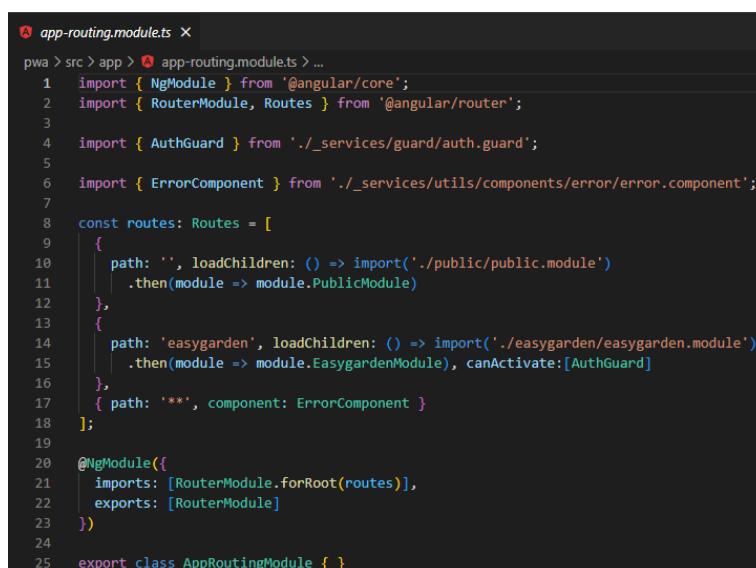


```

 1  import { NgModule } from '@angular/core';
 2  import { BrowserModule } from '@angular/platform-browser';
 3  import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
 4  import { HttpClientModule } from '@angular/common/http';
 5  import { AppRoutingModule } from './app-routing.module';
 6  import { FontAwesomeModule } from '@fortawesome/angular-fontawesome';
 7  import { JWTInterceptorProvider } from './_services/utils/jwt.interceptor';
 8  import { AppComponent } from './app.component';
 9  import { FooterComponent } from './components/footer/footer.component';
10  import { ErrorComponent } from './_services/utils/components/error/error.component';
11
12  @NgModule({
13    declarations: [
14      AppComponent,
15      FooterComponent,
16      ErrorComponent
17    ],
18    imports: [
19      BrowserModule,
20      AppRoutingModule,
21      HttpClientModule,
22      FontAwesomeModule,
23      BrowserAnimationsModule
24    ],
25    providers: [JWTInterceptorProvider],
26    bootstrap: [AppComponent]
27  })
28
29  export class AppModule { }
30

```

Celui-ci possède aussi de base un module de routing.



```

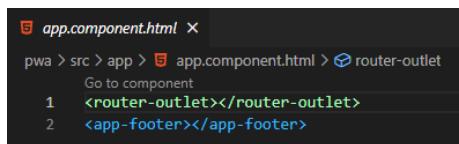
 1  import { NgModule } from '@angular/core';
 2  import { RouterModule, Routes } from '@angular/router';
 3
 4  import { AuthGuard } from './_services/guard/auth.guard';
 5
 6  import { ErrorComponent } from './_services/utils/components/error/error.component';
 7
 8  const routes: Routes = [
 9    {
10      path: '',
11      loadChildren: () => import('./public/public.module')
12        .then(module => module.PublicModule)
13    },
14    {
15      path: 'easygarden', loadChildren: () => import('./easygarden/easygarden.module')
16        .then(module => module.EasygardenModule), canActivate:[AuthGuard]
17    },
18    { path: '**', component: ErrorComponent }
19  ];
20
21  @NgModule({
22    imports: [RouterModule.forRoot(routes)],
23    exports: [RouterModule]
24  })
25  export class AppRoutingModule { }

```

L'on peut constater que dans ce module de routing j'importe le composant d'erreur 404 (afin que celui-ci soit disponible dans toute l'application) et que je déclare son routing dans la constante «routes». Nous noterons la présence de l'import du service de Guard et de la propriété **canActivate** sur la route /easygarden. De plus la propriété **loadChildren** permet quant à elle le lazy-loading du module privé.

Le lazy loading (ou lazy load, «chargement fainéant» en français) consiste à spécifier quels composants d'un programme doivent être chargés lors du démarrage de celui-ci. Si un composant logiciel non préalablement chargé se révèle nécessaire au cours de l'utilisation, ce composant sera chargé à ce moment-là. Cela aura pour effet d'accélérer le fonctionnement global du système, tout en induisant un temps d'attente lors de la sollicitation d'un composant non préalablement chargé.

Je déclare également dans le app.component.html la balise <router-outlet> (pour le routing de l'application) et <app-footer> afin d'injecter mon composant footer dans chaque vue de l'application.



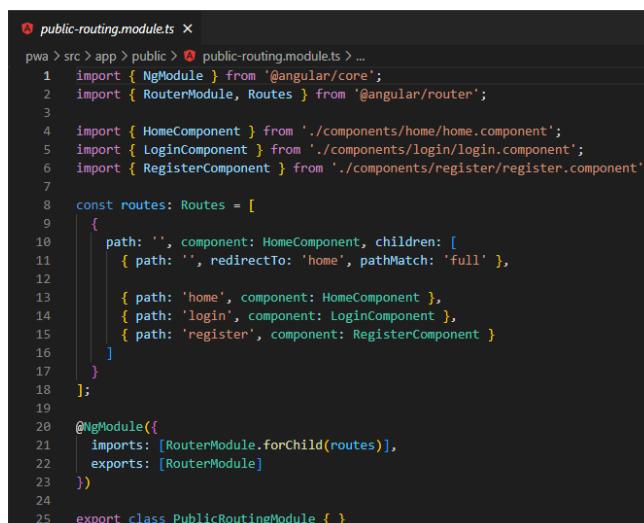
```

1   <router-outlet></router-outlet>
2   <app-footer></app-footer>

```

Je vais maintenant présenter le module public: afin de créer un module ainsi que son routing il faut se placer dans le répertoire app de l'application et lancer la commande:

ng g m public –routing



```

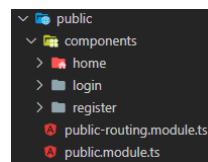
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3
4 import { HomeComponent } from './components/home/home.component';
5 import { LoginComponent } from './components/login/login.component';
6 import { RegisterComponent } from './components/register/register.component';
7
8 const routes: Routes = [
9   {
10     path: '',
11     component: HomeComponent,
12     children: [
13       { path: '', redirectTo: 'home', pathMatch: 'full' },
14       { path: 'home', component: HomeComponent },
15       { path: 'login', component: LoginComponent },
16       { path: 'register', component: RegisterComponent }
17     ]
18   ];
19
20 @NgModule({
21   imports: [RouterModule.forChild(routes)],
22   exports: [RouterModule]
23 })
24 export class PublicRoutingModule { }

```

Pour la création des composants la CLI d'Angular met à disposition la commande suivante:

ng g c nomComposant

L'architecture du module public avec ses différents composants:



Le HomeComponent sera chargé en premier lieu et j'injecterai les composants Login et Register de manière conditionnelle grâce à la directive *NgIf.

```

<div *ngIf="loginComponent || registerComponent; else homepageBlock>
  <ng-container *ngIf="loginComponent">
    <app-login (onClose)="loginComponentClosed($event)"></app-login>
  </ng-container>
  <ng-container *ngIf="registerComponent">
    <app-register (onClose)="registerComponentClosed($event)"></app-register>
  </ng-container>
</div>

<ng-template #homepageBlock>
  <div id="homePage">
    <div class="connexion">
      <div class="icon">
        <button type="button"
          class="button-circle"
          (click)="loadLoginComponent()"
          [routerLink]=["'login'"]>
          <fa-icon [icon]="'faUserCircle'"></fa-icon>
        </button>
        <p>Log In</p>
      </div>
    </div>
    <div class="logo">
      {{title}}
    </div>
    <div class="resume">
      Easy Garden l'application qui vous facilite la gestion de vos équipements de parcs et jardins.  

      Une application all-in-one pour gérer tout les équipements tels que vos systèmes d'arrosage  

      intégrés, vos jeux de lumières, l'ouverture de votre portail, votre tondeuse intelligente et bien plus encore!<br>
      <button type="button"
        (click)="loadRegisterComponent()"
        [routerLink]=["'register'"]>Inscription
      </button>
    </div>
  </div>
</ng-template>
  
```

De plus j'importe le ReactiveFormsModule dans mon public.module.ts afin de gérer les formulaires de connexion et de création de compte utilisateur.

```

import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { PublicRoutingModule } from './public-routing.module';
import { FontAwesomeModule } from '@fortawesome/angular-fontawesome';
import { ReactiveFormsModuleModule } from '@angular/forms';
import { HomeComponent } from './components/home/home.component';
import { LoginComponent } from './components/login/login.component';
import { RegisterComponent } from './components/register/register.component';

@NgModule({
  declarations: [
    HomeComponent,
    LoginComponent,
    RegisterComponent
  ],
  imports: [
    CommonModule,
    PublicRoutingModule,
    FontAwesomeModule,
    ReactiveFormsModuleModule
  ]
})
export class PublicModule { }
  
```

Je ne présenterai ici que le composant Login celui-ci présentant une grande partie de la logique de sécurité. Le fichier login.component.html:

```
<div class="div-wrapper animate__animated animate__fadeInDown" id="form-login">
  <div>
    
  </div>
  <h2 class="form-title title-form-shadow">{{title}}</h2>
  <div class="div-wrapper">
    <div class="form-content form-border form-shadow">
      <div class="div-button-close">
        <button type="button"
          aria-label="Fermer"
          (click)="closeLoginForm()"
          class="button-circle button-shadow">
          <fa-icon [icon]="'faCircleXmark'></fa-icon>
        </button>
      </div>
      <div class="form-title-content">
        <p class="p-form-title-content">LOGIN</p>
      </div>
      <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
        <input formControlName="email"
          name="email"
          id="email"
          type="email"
          placeholder="Email"
          class="input-inset-shadow"
          [ngClass]="{{ 'invalid-feedback': submitted && f['email'].errors }}>
        <div *ngIf="submitted && f['email'].errors">
          <div class="text-alert" *ngIf="f['email'].errors['required']">Veuillez saisir un email!</div>
          <div class="text-alert" *ngIf="f['email'].errors['email']">Format d'email invalide!</div>
          <div class="text-alert" *ngIf="f['email'].errors['validEmail']">
            L'email doit contenir un "." + nom de domaine!
          </div>
        </div>
        <div class="div-password-icon">
          <input formControlName="password"
            name="password"
            id="password"
            [type]={{ visible ? 'text' : 'password' }}
            placeholder="Password"
            class="input-inset-shadow"
            [ngClass]="{{ 'invalid-feedback': submitted && f['password'].errors }}>
          <fa-icon [icon]={{ visible ? faEyeSlash : faEye }} aria-hidden="true" (click)="toggle()"/></fa-icon>
          <div *ngIf="submitted && f['password'].errors">
            <div class="text-alert" *ngIf="f['password'].errors['required']">Veuillez saisir un mot de passe!</div>
          </div>
        </div>
        <div class="div-form-button">
          <button type="submit"
            class="button-shadow"
            [routerLink]=["easygarden"]>Connexion
          </button>
          <button type="button"
            (click)="onReset()"
            class="button-shadow">Annuler
          </button>
        </div>
      </form>
    </div>
  </div>
</div>
```

Le fichier login.component.ts:

```

import { Component, OnInit, EventEmitter, Output } from '@angular/core';
import { AbstractControl, FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';
import { faCircleXmark, faEye, faEyeSlash } from '@fortawesome/free-solid-svg-icons';

import { AuthService } from '../../../../../_services/auth/auth.service';
import { FormValidationService } from '../../../../../_services/service/form-validation.service';
import { TokenService } from '../../../../../_services/auth/token.service';
import { CredentialsModel } from '../../../../../_models/credentialsModel';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})

export class LoginComponent implements OnInit {

  title = 'Easy Garden';
  faCircleXmark = faCircleXmark;
  faEye = faEye;
  faEyeSlash = faEyeSlash;

  // Toggle faEyeSlash
  visible: boolean = false;
  public toggle(): void {
    this.visible = !this.visible;
  }

  // Close component Login
  @Output()
  onClose: EventEmitter<boolean> = new EventEmitter();

  closeLoginForm() {
    this.onClose.emit(true);
  }

  // LoginForm Group
  loginForm = new FormGroup({
    email: new FormControl(''),
    password: new FormControl('')
  });
  submitted = false;
  success = '';

  constructor(private formBuilder: FormBuilder,
              private customValidator: FormValidationService,
              private authService: AuthService,
              private tokenService: TokenService) {
    this.loginForm = this.formBuilder.group({
      email: [
        '',
        [
          Validators.required,
          Validators.email,
          this.customValidator.validEmail()
        ]
      ],
      password: ['', [Validators.required]]
    })
  }

  ngOnInit(): void {
  }

  get f(): { [key: string]: AbstractControl } {
    return this.loginForm.controls;
  }

  onSubmit() {
    this.submitted = true;
    // console.log(this.loginForm);
    if (this.loginForm.invalid) {
      return;
    }
    const typedLoginForm: CredentialsModel = this.loginForm.value;
    this.success = JSON.stringify(typedLoginForm);
    // console.log(typedLoginForm)
    this.authService.logIn(typedLoginForm).subscribe(
      data => {
        // console.log(data.token)
        this.tokenService.saveToken(data.token)
      },
      err => console.log(err)
    )
  }

  onReset(): void {
    this.submitted = false;
    this.loginForm.reset();
  }
}

```

Dans le login.component.html nous noterons tout d'abord la présence d'une méthode **closeLoginForm()** sur une balise <button>. La méthode et l'Emitter dans le component.ts:

```
// Close component Login
@Output()
onClose: EventEmitter<boolean> = new EventEmitter();

closeLoginForm() {
  this.onClose.emit(true);
}
```

Celle-ci sert à émettre l'événement de fermeture du composant Login dans le composant Home (grâce à un booléen instancié initialement à false) et qui ensuite redirige l'utilisateur vers l'URL /home en utilisant le routeur et la méthode **navigate()**.

```
// Login Component
public LoginComponent = false;
loadLoginComponent(){
  this.loginComponent = !this.loginComponent;
}
loginComponentClosed(_onClosed: boolean) {
  this.loginComponent = false;
  this.router.navigate(['home']);
}
```

J'utilise ensuite la balise <form> et la classe [FormGroup] à laquelle j'attribue un nom afin de lier le formulaire créé dans le component.ts à celui du HTML.

```
// LoginForm Group
loginForm = new FormGroup({
  email: new FormControl(''),
  password: new FormControl('')
});
```

Dans ce FormGroup j'instancie deux objets représentant chaque input, ils me permettent de mettre en place un contrôle de saisie utilisateur. Nous pouvons donc voir dans le formulaire du component.html l'utilisation de la directive de classe [ngClass] qui permet d'injecter une classe css (invalid-feedback) si le formulaire soumis n'est pas valide et fait appel à l'AbstractControl d'Angular .et son getter **f()**.

```
get f(): { [key: string]: AbstractControl } {
  return this.loginForm.controls;
}
```

Si le formulaire soumis n'est pas valide des balises <div> de contrôle de validations seront injectées dans le DOM par Angular grâce à la directive *NglIf.

```
<div *ngIf="submitted && f['email'].errors">
  <div class="text-alert" *ngIf="f['email'].errors['required']">Veuillez saisir un email!</div>
  <div class="text-alert" *ngIf="f['email'].errors['email']">Format d'email invalide!</div>
  <div class="text-alert" *ngIf="f['email'].errors['validEmail']">
    L'email doit contenir un "." + nom de domaine!
  </div>
</div>
```

Ces mêmes contrôles de validations sont déclarées dans le constructeur du component.ts.

Je déclare certaines propriétés dans ce même constructeur:

- La propriété formBuilder qui fait référence au service FormBuilder (cela crée une instance du FormControl).
- La propriété customValidator qui fait référence au FormValidationService (service personnalisé de contrôle de validation).
- La propriété authService qui fait référence au AuthService (service personnalisé d'authentification).
- La propriété tokenService qui fait référence au TokenService (service personnalisé pour le token).

```
constructor(private formBuilder: FormBuilder,
           private customValidator : FormValidationService,
           private authService: AuthService,
           private tokenService: TokenService) {
  this.loginForm = this.formBuilder.group({
    email: [
      '',
      [
        Validators.required,
        Validators.email,
        this.customValidator.validEmail()
      ]
    ],
    password: ['', [Validators.required]]
  })
}
```

Le form-validation.service.ts (le fichier complet est présent dans les annexes):

```
validEmail(): ValidatorFn {
  return (control: AbstractControl): { [key: string]: boolean } | null => {
    if (control.value === '') return null;

    let re = new RegExp(`^([a-zA-Z0-9._%+-]+@[a-zA-Z.-]+\.[a-zA-Z]{2,4}$)`);
    if (re.test(control.value)) {
      return null;
    } else {
      return { validEmail: true };
    }
  };
}
```

La balise <button> qui appelle la méthode **onSubmit()** déclaré dans la balise <form> grâce à l'événement (**ngSubmit**). Celle-ci dirigera l'utilisateur vers l'URL /easygarden si le formulaire et les credentials de login sont valides:

```
<div class="div-form-button">
  <button type="submit"
         class="button-shadow"
         [routerLink]="'[easygarden']">Connexion
  </button>
```

```
onSubmit() {
  this.submitted = true;
  // console.log(this.loginForm);
  if (this.loginForm.invalid) {
    return;
  }
  const typedLoginForm: CredentialsModel = this.loginForm.value;
  this.success = JSON.stringify(typedLoginForm);
  // console.log(typedLoginForm)
  this.authService.logIn(typedLoginForm).subscribe(
    data => {
      // console.log(data.token)
      this.tokenService.saveToken(data.token)
    },
    err => console.log(err)
  )
}
```

L'interface `CredentialsModel` qui permet de vérifier que les données reçues respectent une structure et un type explicitement définis.

```
credentialsModel.ts ×
pwa > src > app > _models > credentialsModel.ts > CredentialsModel
1  export interface CredentialsModel {
2    email: string;
3    password: string;
4 }
```

La propriété `value` de l'`AbstractControl` permet de récupérer les valeurs du formulaire pour ensuite les insérer dans la constante `typedLoginForm`. Celles-ci seront convertis en JSON grâce à la méthode `stringify()`. J'appelle ensuite la propriété `authService` et sa méthode `logIn()` à laquelle je passe en paramètre la constante `typedLoginForm` et souscris les datas à la propriété `tokenService` et sa méthode `saveToken()`

```
export class AuthService {
  constructor(private httpClient: HttpClient) { }

  logIn(credentials: CredentialsModel): Observable<TokenModel>{
    return this.httpClient.post<TokenModel>(environment.apis.login.url, credentials)
  }

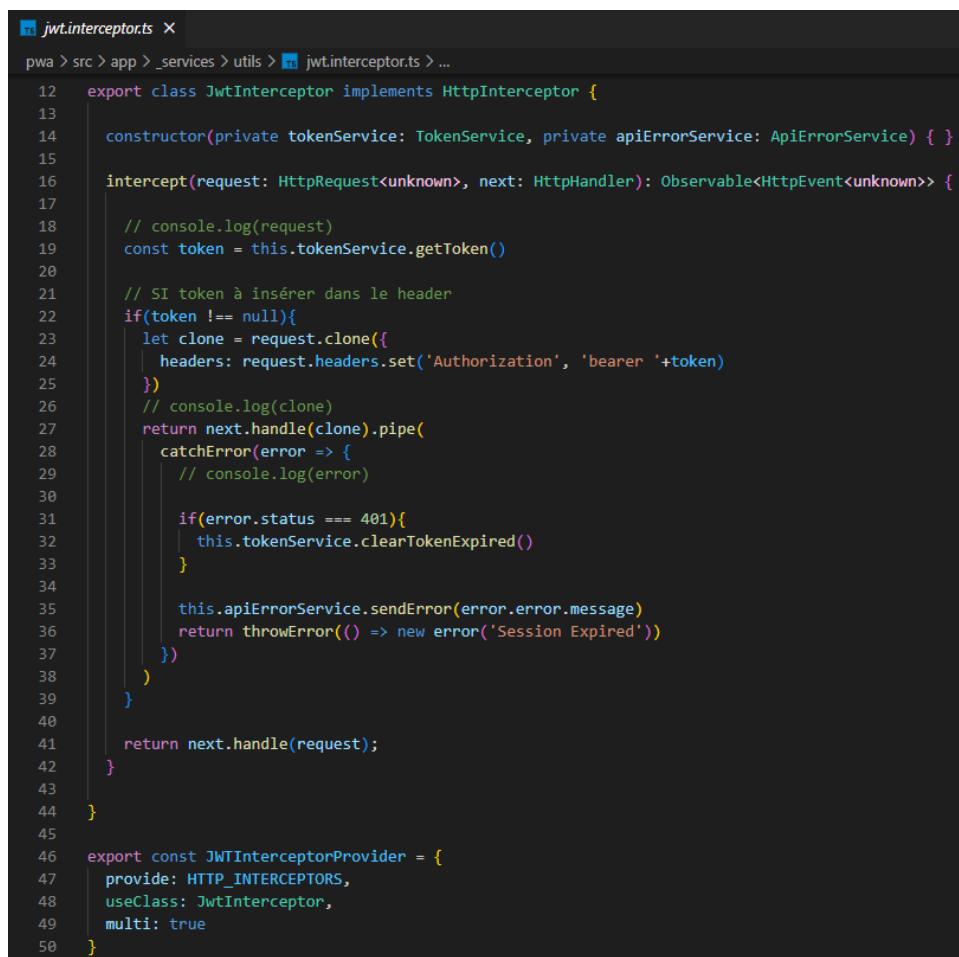
  export class TokenService {
    constructor(private router: Router) { }

    saveToken(token: string): void{
      localStorage.setItem('token', token)
      this.router.navigate(['easygarden'])
    }
  }
}
```

J'ai défini les URI de l'API à appeler dans le fichier `environments.ts`.

```
environments.ts ×
pwa > src > environments > environments.ts > ...
5  export const environment = {
6    production: false,
7    apis: {
8      login: {
9        url: 'http://localhost:8000/api/login_check'
10    },
11      user: {
12        url: 'http://localhost:8000/api/users'
13    },
14      garden: {
15        url: 'http://localhost:8000/api/gardens'
16    },
17      lawnmower: {
18        url: 'http://localhost:8000/api/lawnmowers'
19    },
20      lightning: {
21        url: 'http://localhost:8000/api/lightnings'
22    },
23      pool: {
24        url: 'http://localhost:8000/api/pools'
25    },
26      portal: {
27        url: 'http://localhost:8000/api/portals'
28    },
29      watering: {
30        url: 'http://localhost:8000/api/waterings'
31    }
32  };
33 }
```

Chaque requête sera soumise à l'interceptor grâce à la méthode **intercept()** et du fait d'avoir déclaré celle-ci comme provider dans l'app.module.ts (un provider est un objet que l'on déclare à Angular pour qu'il puisse l'injecter dans l'application). Celui-ci créera un clone de cette requête car une requête ne peut être modifiée. Un interceptor permet d'intercepter les requêtes HTTP entrantes et sortantes afin de les modifier ou d'implémenter une mécanique logicielle particulière, ici un header d'autorisation. Dans notre cas, nous avons pu constater que tous mes appels HTTP ne sont jamais décorés avec le token, nous aurions pu le faire pour chaque méthode mais l'interceptor est là pour nous simplifier la vie. J'intercepte donc ce qui sort, je clone la requête et j'ajoute mon token. La méthode **intercept()** passe ensuite la nouvelle requête à la méthode **next.handle()** pour lui permettre de continuer son chemin.



```

1  jwt.interceptor.ts ×
2  pwa > src > app > _services > utils > jwt.interceptor.ts > ...
3
4  export class JwtInterceptor implements HttpInterceptor {
5
6    constructor(private tokenService: TokenService, private apiErrorHandler: ApiErrorHandler) { }
7
8    intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
9
10       // console.log(request)
11       const token = this.tokenService.getToken()
12
13       // SI token à insérer dans le header
14       if(token !== null){
15           let clone = request.clone({
16               headers: request.headers.set('Authorization', 'bearer ' +token)
17           })
18           // console.log(clone)
19           return next.handle(clone).pipe(
20               catchError(error => {
21                   // console.log(error)
22
23                   if(error.status === 401){
24                       this.tokenService.clearTokenExpired()
25                   }
26
27                   this.apiErrorHandler.handleError(error.error.message)
28                   return throwError(() => new Error('Session Expired'))
29               })
30           )
31
32       }
33
34       return next.handle(request);
35   }
36
37 }
38
39 }
40
41 }
42
43 }
44
45 export const JWTInterceptorProvider = {
46     provide: HTTP_INTERCEPTORS,
47     useClass: JwtInterceptor,
48     multi: true
49 }
50

```

Le Guard entre alors en action, il permet de contrôler l'accès à une route, celui-ci est implémenté au niveau du routing. Le Guard ne doit en aucun cas être considéré comme un mécanisme de sécurité. Tout d'abord la classe doit implémenter l'interface CanActivate.

Lorsqu'un utilisateur essaie d'accéder à une route protégée, le guard est appelé et vérifie l'état d'authentification de l'utilisateur. S'il est authentifié (méthode **isLogged()** de la propriété **tokenService**) la méthode **CanActivate()** retourne True et charge le module de la route /easygarden. Sinon on redirige l'utilisateur vers la page de login pour lui permettre de se connecter.

```

auth.guard.ts ×
pwa > src > app > _services > guard > auth.guard.ts > ...
11  export class AuthGuard implements CanActivate {
12
13    constructor(private router: Router,
14                private tokenService: TokenService) { }
15
16    canActivate(
17      _route: ActivatedRouteSnapshot,
18      _state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
19
20      if(this.tokenService.isLoggedIn()){
21        return true
22      }
23
24      return this.router.navigate(['/'])
25
26    }
27
28  }
}

{
  path: 'easygarden', loadChildren: () => import('./easygarden/easygarden.module')
    .then(module => module.EasygardenModule), canActivate:[AuthGuard]
},
{ path: '**', component: ErrorComponent }

```

J'en profite ici pour présenter le composant d'erreur 404. Une erreur 404 est un code d'erreur HTTP transmis par un serveur web quand une ressource demandée est inexistante, indisponible ou que le serveur n'arrive pas à la trouver. Généralement, c'est une page web qui n'existe pas.



La page que vous recherchez n'a pas été trouvée.

Vous allez être redirigé dans 6s.

[Retour](#)

Les deux astérix du chemin permettent à Angular de sélectionner cette route chaque fois que l'URL demandée ne correspond à aucun autre chemin du routeur.

Je déclare dans le constructeur la méthode **setTimeOut()** afin que le composant ne s'affiche que 7000ms soit 7secondes et redirige l'utilisateur à l'URL d'où il provenait en me servant de la classe Location et sa méthode **back()**. A l'initialisation du composant j'utilise la méthode **setInterval()** que j'auto-invoquerai et qui affichera un timer indiquant à l'utilisateur le temps

restant avant d'être redirigé. Pour cela j'ai instancié une variable **time** de type Number à la valeur 7 et je décrémente celle-ci dans la méthode **setInterval()**.

J'ai de plus créer un bouton permettant à l'utilisateur de revenir à l'URL d'où il provenait sans avoir à attendre. Celui-ci fait aussi appel à la méthode **back()** mais aussi à la méthode **clearTimeout()** afin d'annuler la méthode **setTimeout()** du constructeur.

```

error.component.ts ×
pwa > src > app > _services > components > error > error.component.ts > ...
10  export class ErrorComponent implements OnInit {
11
12    time: number = 7;
13    timeOut: any;
14
15    constructor(private location: Location) {
16      this.timeOut = setTimeout(() => {
17        | this.location.back();
18      }, 7000);
19    }
20
21    ngOnInit(): void {
22      setInterval(() => { if (this.time > 0) this.time--; }, 1000);
23    }
24
25    goBack() {
26      this.location.back();
27      clearTimeout(this.timeOut);
28    }
29
30  }

```

Une fois l'utilisateur connecté, celui-ci est donc dirigé vers l'URL /easygarden et le module privé easygarden.module.ts est chargé. La balise <app-navbar> injecte la navbar de l'application et la balise <router-outlet> injecte le routing du module

```

easygarden-routing.module.ts ×
pwa > src > app > easygarden > easygarden-routing.module.ts > ...
8  const routes: Routes = [
9    {
10      path: '', component: GardenComponent, children: [
11
12        {
13          path: 'profil', loadChildren: () => import('./modules/profil/profil.module')
14          | .then(module => module.ProfilModule)
15        },
16        {
17          path: 'watering', loadChildren: () => import('./modules/watering/watering.module')
18          | .then(module => module.WateringModule)
19        },
20        {
21          path: 'lightning', loadChildren: () => import('./modules/lightning/lightning.module')
22          | .then(module => module.LightningModule)
23        },
24        {
25          path: 'portal', loadChildren: () => import('./modules/portal/portal.module')
26          | .then(module => module.PortalModule)
27        },
28        {
29          path: 'pool', loadChildren: () => import('./modules/pool/pool.module')
30          | .then(module => module.PoolModule)
31        },
32        {
33          path: 'lawnmower', loadChildren: () => import('./modules/lawnmower/lawnmower.module')
34          | .then(module => module.LawnmowerModule)
35        },
36
37        { path: 'garden/:id', component: EditGardenComponent },
38        { path: 'add/:id', component: AddGardenComponent },
39      ]
40    ];
41

```

```

  @File navbar.component.html
  pwa > src > app > easygarden > components > navbar >   navbar.component.html > header
    Go to component
1   <header>
2     <nav>
3       <div>
4         <div>
5           <ul>
6             <div class="nav-logo" [routerLink]="'./'>
7               
8             </div>
9             <li class="border-bottom" [routerLinkActive]="'active-link'>
10               <a [routerLink]="'watering'">Arrosage</a>
11             </li>
12             <li class="border-bottom" [routerLinkActive]="'active-link'>
13               <a [routerLink]="'lightning'">Éclairage</a>
14             </li>
15             <li class="border-bottom" [routerLinkActive]="'active-link'>
16               <a [routerLink]="'pool'">Bassin</a>
17             </li>
18             <li class="border-bottom" [routerLinkActive]="'active-link'>
19               <a [routerLink]="'lawnmower'">Tondeuse</a>
20             </li>
21             <li class="border-bottom" [routerLinkActive]="'active-link'>
22               <a [routerLink]="'portal'">Portail</a>
23             </li>
24           </ul>
25         </div>
26         <div>
27           <ul>
28             <li class="border-bottom" [routerLinkActive]="'active-link'>
29               <a [routerLink]="'profil'">Profil</a>
30             </li>
31             <button type="button"
32                   (click)="logOut()>
33               class="button-shadow">Déconnexion
34             </button>
35           </ul>
36         </div>
37       </div>
38     </nav>
39   </header>

```

La balise `<button>` ayant la méthode `logOut()` sur un événement de type `(click)` appelle quant à elle la méthode `clearToken()` du tokenService. Celle-ci efface le token du localStorage et redirige l'utilisateur à l'accueil de l'application

```

clearToken(): void{
  localStorage.removeItem('token')
  this.router.navigate(['/'])
}

```

Le composant garden est chargé en utilisant une directive `*ngIf` et à la condition que la route soit strictement correspondante à `/easygarden`.

```

  @File garden.component.html
  pwa > src > app > easygarden > components > garden > garden >   garden.component.html
    Go to component
1   <app-navbar></app-navbar>
2   <router-outlet></router-outlet>
3   <div *ngIf="router.url === '/easygarden'" class="div-wrapper">

```

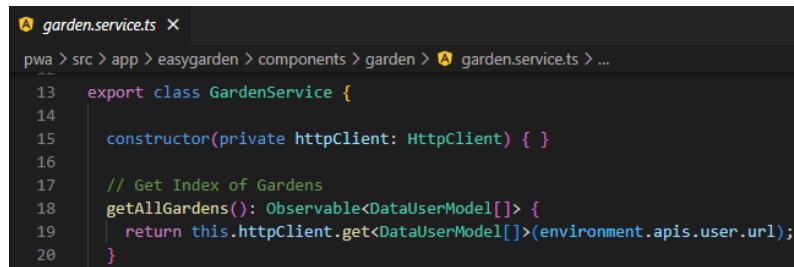
Afin d'hydrater le tableau contenant les jardins possédés par l'utilisateur je crée une méthode `fetchGardens()` dans le fichier `garden.component.ts`. Cette même méthode appellera la méthode `getAllGardens()` dans le service que j'ai créée pour les jardins, j'appelle ici l'URI `/users` afin de récupérer toutes les datas liés à l'utilisateur par le biais du `UserDataProvider` que j'ai mis en place dans le back-end.



```

  37  ngOnInit(): void {
  38    |  this.fetchGardens()
  39  }
  40
  41  // Display Gardens
  42  fetchGardens(): void {
  43    |  this.gardenService.getAllGardens()
  44    |  .subscribe(
  45    |    (res:any) => {
  46      |      if (res.hasOwnProperty('hydra:member'))
  47      |        this.users = res['hydra:member']
  48    }
  49  }
  50

```



```

  13  export class GardenService {
  14
  15    constructor(private httpClient: HttpClient) { }
  16
  17    // Get Index of Gardens
  18    getAllGardens(): Observable<DataUserModel[]> {
  19      return this.httpClient.get<DataUserModel[]>(environment.apis.user.url);
  20    }

```

Je peux dès lors développer le code pour le tableau dans le fichier garden.component.html. J'utilise la balise <th> afin de spécifier les éléments d'entêtes de mon tableau. Puis dans la balise <tbody> j'utilise la balise <ng-container> et la directive *ngFor à deux reprises afin de boucler jusqu'à la seconde dimension du tableau users que j'ai récupéré avec le service. De plus un pipe est inséré dans la seconde boucle afin de paginer les résultats. Cette opération est rendu possible par le module ngx-pagination déclaré dans les imports du easygarden.module.ts. Je défini que trois résultats seront affichés par page grâce à l'option itemsPerPage.

J'utilise ensuite l'interpolation sur les éléments de cellule du tableau (balise <td>) afin d'afficher le nom du jardin ainsi que son propriétaire. Puis dans une troisième cellule j'affiche deux boutons contenant une icône FontAwesome, ceux-ci sont destinés:

- Pour le premier: à modifier le nom du jardin et load le composant editGarden. J'utilise pour ce faire le routerLink vers l'URL souhaité et dans laquelle j'interpole l'id de l'équipement sélectionné.
- Pour le second: à supprimer le jardin (et donc tout les équipements lui étant associés, cf l'option orphanRemoval présente dans le back-end). Une méthode **confirmDialog()** activant une modale de confirmation de suppression a de plus été développé. Je lui passe en paramètre l'id de l'équipement ainsi que son nom (afin de l'afficher dans la modale).

Un tooltip a été implémenté sur les icônes afin d'améliorer l'expérience utilisateur et l'accessibilité. J'ai donc utilisé le module MatTooltipModule d'Angular Material. Celui-ci fournit un ensemble de composants d'interface utilisateur réutilisables.



Un dernier bouton est ajouté afin de load le composant addGarden. J'utilise pour ce faire le routerLink vers l'URL souhaité et dans laquelle j'interpole l'id de l'utilisateur (un jardin devant appartenir à un utilisateur j'ai absolument besoin de fournir cette valeur comme clé étrangère afin de créer un nouveau jardin).

```

<div class="table form-border form-shadow">
  <table class="t-hover">

    <thead>
      <tr>
        <th>Nom</th>
        <th>Utilisateur</th>
        <th>Action</th>
      </tr>
    </thead>

    <tbody>
      <ng-container *ngFor="let user of users">
        <tr *ngFor="let garden of user.garden | paginate: { itemsPerPage: 3, currentPage: p }">
          <td>{{ garden.name }}</td>
          <td>{{ user.pseudo }}</td>
          <td>
            <button type="button"
                  aria-label="Modifier"
                  mat-raised-button
                  matTooltip="Modifier"
                  matTooltipPosition="before"
                  matTooltipShowDelay="500"
                  routerLink="/easygarden/garden/{{garden.id}}">
              <fa-icon [icon]="faPen"></fa-icon>
            </button>
            <button type="submit" class="trash"
                  aria-label="Supprimer"
                  mat-raised-button
                  matTooltip="Supprimer"
                  matTooltipPosition="after"
                  matTooltipShowDelay="500"
                  (click)="confirmDialog(garden.id, garden.name)">
              <fa-icon [icon]="faTrash"></fa-icon>
            </button>
          </td>
        </tr>
      </ng-container>
    </tbody>
  </table>
</div>
<div class="pagination">
  <pagination-controls (pageChange)="p = $event"
    previousLabel=""
    nextLabel=""
    [autoHide]="true"
    [responsive]="true"
    aria-label="Pagination">
  </pagination-controls>
</div>
<ng-container *ngFor="let user of users">
  <div class="submit">
    <button type="button"
           id="submit"
           class="button-shadow"
           routerLink="/easygarden/add/{{user.id}}">Ajouter</button>
  </div>
</ng-container>
</div>

```

La méthode pour supprimer un jardin avec sa méthode de service:

```
// Delete Garden
confirmDialog(id: string, name: string): void {
  const message = 'Êtes-vous certain de vouloir supprimer l\'équipement "'+ name +'" ?';
  const dialogData = new ConfirmDialogModel("Confirmer l'action!", message);
  const dialogRef = this.dialog.open(ConfirmDialogComponent, {
    maxWidth: "400px",
    data: dialogData
  })

  dialogRef.afterClosed().subscribe(dialogResult => {
    this.result = dialogResult;
    if (this.result === true) {
      this.gardenService.deleteGarden(id).subscribe(
        () => {
          this.fetchGardens()
        }
      )
    }
  })
}

// Delete Garden
deleteGarden(id: string): Observable<GardenModel> {
  return this.httpClient.delete<GardenModel>(environment.apis.garden.url+'/'+id)
}
```

L'interface GardenModel:

```
gardenModel.ts ×
pwa > src > app > easygarden > components > garden
1  export interface GardenModel {
2    id: number,
3    name: string,
4    user: {
5      id: string,
6      pseudo: string
7    }
8  }
9
10 export interface DataGardenModel {
11   data: GardenModel[]
12 }
```

Le composant editGarden sera quand à lui un formulaire fonctionnant sur le même principe que celui de login pour ce qui est des contrôles de saisies, néanmoins j'ai implémenté dans le **ngOnInit()** une méthode **getGarden()** faisant appel à mon service afin de préremplir le champ de saisie avec la valeur souhaitée.

```
ngOnInit(): void {
  window.scrollTo(0, 0);
  let gid = this.activated.snapshot.paramMap.get('id')
  this.gardenService.getGarden(gid).subscribe(
    data => {
      this.garden = data
      this.value = this.garden.name;
    }
  )
}
```

```
// Get Garden
getGarden(gid: string | null): Observable<GardenModel>{
  return this.httpClient.get<GardenModel>(environment.apis.garden.url+'/'+gid)
}
```

La méthode **onSubmit()** pour modifier le nom d'un jardin et sa méthode de service:

```
// Submit button
onSubmit() {
  this.submitted = true;
  if (this.editGardenForm.invalid) {
    return;
  } else {
    const typedEditGardenForm: GardenModel = this.editGardenForm.value;
    this.success = JSON.stringify(typedEditGardenForm);
    let gid = this.activatedRoute.snapshot.paramMap.get('id')
    this.gardenService.updateGarden(typedEditGardenForm, gid).subscribe()
    this.router.navigateByUrl('/', {skipLocationChange: true}).then(() => {
      this.router.navigate(['/easygarden']);
    });
  }
}
```

```
// Update Garden
updateGarden(garden: GardenModel, gid: string | null): Observable<DataGardenModel[]> {
  return this.httpClient.put<DataGardenModel[]>(environment.apis.garden.url+'/'+gid, garden)
}
```

Le composant addGarden sera lui aussi un formulaire fonctionnant sur le même principe que celui de login pour ce qui est des contrôles de saisies. Voici la méthode **onSubmit()** pour ajouter un jardin et sa méthode de service:

```
// Submit button
onSubmit() {
  this.submitted = true;
  if (this.addGardenForm.invalid) {
    return;
  } else {
    const typedAddGardenForm: GardenModel = this.addGardenForm.value;
    this.success = JSON.stringify(typedAddGardenForm);
    this.gardenService.addGarden(typedAddGardenForm).subscribe(
      () => {
        this.router.navigate(['/easygarden'])
      }
    )
  }
}
```

```
// Add Garden
addGarden(garden: GardenModel) {
  const json = {
    name: garden.name,
    user: garden.user
  };
  return this.httpClient.post(environment.apis.garden.url, json)
}
```

Et pour finir voici la manière dont j'ai crée le formulaire: la propriété user devant avoir comme valeur l'URI api/users/+l'id de l'utilisateur (ici gid récupéré dans l'URL avec `activated.snapshot.paramMap.get('id')`).

```
constructor(private formBuilder: FormBuilder,
            private customValidator : FormValidationService,
            private router: Router,
            private location: Location,
            private activated: ActivatedRoute,
            private gardenService: GardenService) {
  let gid = this.activated.snapshot.paramMap.get('id')
  this.addGardenForm = this.formBuilder.group({
    name: [
      '',
      [
        Validators.required,
        Validators.minLength(3),
        Validators.maxLength(20),
        this.customValidator.validEquipmentName()
      ]
    ],
    user: [
      'api/users/'+gid
    ]
  })
}
```

Pour ce qui est des entités représentant les équipements (Arrosage-Eclairage-Bassin-Tondeuse-Portail) il m'a fallu implémenter un bouton toggle dans les cellules du tableau correspondant afin d'éteindre et d'allumer l'équipement en question. Voici la méthode `updateStatus()` sa méthode de service et son HTML:

```
// Update Status
updateStatus(id: number, status: boolean): void {
  if (status === true) {
    status = !status;
    // console.log(id, status)
    this.wateringService.updateStatus(status, id)
      .subscribe(
        (res:any) => {
          this.status = res
          // console.log(status)
          this.fetchWaterings()
        }
      )
  } else if (status === false) {
    status = !status;
    // console.log(id, status)
    this.wateringService.updateStatus(status, id)
      .subscribe(
        (res:any) => {
          this.status = res
          // console.log(status)
          this.fetchWaterings()
        }
      )
  }
}
```

```
// Update Status
updateStatus(status: boolean, id: number): Observable<DataWateringModel[]> {
  return this.httpClient.put<DataWateringModel[]>(environment.apis.watering.url+'/'+id, {status})
}
```

```
<td class="power">
  <button type="submit"
    aria-label="On"
    *ngIf="watering.status === true"
    (click)="updateStatus(watering.id, watering.status)"
    <fa-icon [icon]="'faPowerOff'></fa-icon>
  </button>
  <button type="submit" class="btn-off"
    aria-label="Off"
    *ngIf="watering.status === false"
    (click)="updateStatus(watering.id, watering.status)"
    <fa-icon [icon]="'faPowerOff'></fa-icon>
  </button>
</td>
```

J'ai aussi utilisé la directive de classe [ngClass] afin d'injecter une propriété CSS box-shadow en inset sur l'équipement si celui-ci est allumé et présente une valeur inférieur à celle attendue. Le tout afin d'alerter l'utilisateur qu'un équipement présente sûrement un problème matériel.

```
<td *ngIf="watering.status === true" [ngClass]="{'alert': watering.flowSensor < '1800'}">{{ watering.flowSensor }}</td>
<td *ngIf="watering.status === false">-</td>
<td *ngIf="watering.status === true" [ngClass]="{'alert': watering.pressureSensor < '2bars'}">{{ watering.pressureSensor }}</td>
<td *ngIf="watering.status === false">-</td>
```

Sur l'équipement Portail, il m'a fallu développer un pipe afin de transformer la valeur booléenne reçue de l'API en un format interprétable par l'utilisateur.

```
presence-sensor.pipe.ts ×
pwa | src | app | easygarden | modules | portal | pipe | presence-sensor.pipe.ts
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'presenceSensor'
5  })
6  export class PresenceSensorPipe implements PipeTransform {
7
8    transform(value: any, ...args: any[]): any {
9      return value ? "Attention!" : "-";
10   }
11 }
12 }
```

```
<td *ngIf="portal.status === true" [ngClass]="portal.presenceSensor ? 'alert' : ''">{{ portal.presenceSensor | presenceSensor }}</td>
<td *ngIf="portal.status === false">-</td>
```

J'ai de plus implémenté, en utilisant Angular Material, une balise <select> bouclant sur les jardins possédés par l'utilisateur, afin de pouvoir lier le nouvel équipement ajouté au jardin sélectionné.

```

<mat-form-field appearance="fill">
  <mat-label>Jardin</mat-label>
  <mat-select formControlName="garden"
    name="garden"
    class="form-select">
    <ng-container *ngFor="let user of users">
      <mat-option [value]="'api/gardens/'+garden.id"
        selected
        *ngFor="let garden of user.garden">
        {{garden.name}}
      </mat-option>
    </ng-container>
  </mat-select>
</mat-form-field>

```

```

this.addWateringForm = this.formBuilder.group({
  name: [
    '',
    [
      Validators.required,
      Validators.minLength(3),
      Validators.maxLength(20),
      this.customValidator.validEquipmentName()
    ]
  ],
  garden: [
    '',
    [
      Validators.required
    ]
  ]
})

```

Angular possède aussi son propre gestionnaire de dépendances, le fichier package.json décrit la configuration du projet et permet de modifier les versions de nos dépendances.

```

"dependencies": {
  "@angular/animations": "~13.2.0",
  "@angular/cdk": "^13.3.9",
  "@angular/common": "~13.2.0",
  "@angular/compiler": "~13.2.0",
  "@angular/core": "~13.2.0",
  "@angular/forms": "~13.2.0",
  "@angular/material": "^13.3.9",
  "@angular/platform-browser": "~13.2.0",
  "@angular/platform-browser-dynamic": "~13.2.0",
  "@angular/router": "~13.2.0",
  "@fortawesome/angular-fontawesome": "^0.10.2",
  "@fortawesome/fontawesome-svg-core": "^6.1.1",
  "@fortawesome/free-brands-svg-icons": "^6.1.1",
  "@fortawesome/free-solid-svg-icons": "^6.1.1",
  "animate.css": "^4.1.1",
  "jwt-decode": "^3.1.2",
  "ngx-pagination": "^6.0.1",
  "rxjs": "~7.5.0",
  "tslib": "^2.3.0",
  "zone.js": "~0.11.4"
},

```

Le fichier angular.json permet quant à lui d'importer des fichiers ou le contenu d'un dossier.

```

"assets": [
  "src/assets"
],
"styles": [
  "src/styles.scss",
  "src/assets/scss/animations.scss",
  "src/assets/scss/responsive.scss",
  "src/app/easygarden/modules/panel.scss",
  "node_modules/animate.css/animate.min.css"
],

```

J'importe ici le dossier assets contenant lui-même:

- un dossier fonts dans lequel j'ai placé la font Charlotte Southern, je ne pouvais pas me servir d'un CDN ou d'un import cette police étant peu commune.

- un dossier img dans lequel j'ai placé les images de l'application ainsi que le background, l'icône et le logo.

J'importe aussi différents fichiers de styles scss:

- le fichier styles.scss.
- le fichier animations.scss contenant des animations développées par moi-même.

```

// Spin
@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}
@-webkit-keyframes spin {
  0% { -webkit-transform: rotate(0deg); }
  100% { -webkit-transform: rotate(360deg); }
}
.spin {
  -webkit-animation: spin 3.5s linear 0s infinite reverse;
  animation: spin 3.5s linear 0s infinite reverse;
}

```

- le fichier responsive.scss.
- le fichier panel.scss exclusivement dédié aux tableaux.
- la librairie animate.min.scss.

Je vais présentée ici la manière dont j'ai développé et structuré le scss de l'application. J'ai en premier lieu travaillé sur le fichier styles.scss afin de définir différentes variables pour les couleurs ainsi que les polices d'écritures et leurs graisses.

```

styles.scss X
pwa > src > styles.scss > ...
35  /*-----Color-Palette-----*/
36  :root {
37    /*-----Main Colors-----*/
38    --ang-color-green: #95cb11;
39    --ang-color-green-rgba: rgba(149, 203, 17, 0.55);
40    --ang-color-white: #ffffff;
41    --ang-color-grey: #666666;
42    --ang-color-alert: #ff0f0f;
43    --ang-color-alert-rgba: rgb(255, 143, 0, 1);
44    --ang-color-alert2: #eb8100;
45    --ang-color-focus: #2fa6ff;
46    --ang-color-background: #e5e5e5;
47
48  /*-----Background-----*/
49  --bg-color: var(--ang-color-background);
50
51  /*-----Text-----*/
52  --ang-color-text-dark: #000000;
53  --ang-color-text-white: #ffffff;
54  --ang-color-text-light: rgba(60, 60, 60, 0.66);
55  --color-heading: var(--ang-color-text-light);
56  --color-text: var(--ang-color-text-dark);
57
58  /*-----Buttons-----*/
59  --ang-color-text-button: #c1c1c1;
60  --ang-color-border-button: #C4F253;
61
62  /*-----Title-----*/
63  --ang-color-title-tab-rgba: rgba(0, 0, 0, 0.5);
64
65  /*-----Hovers-----*/
66  --ang-color-input-hover: #e1ff95;
67  --ang-color-table-hover: #e0f7a1;
68  --color-border-hover: var(--ang-color-green);
69
70  /*-----Shadows-----*/
71  --ang-color-title-form-shadow: #858585;
72  --ang-color-button-shadow-rgba: rgba(0, 0, 0, 0.4);
73  --ang-color-form-shadow-rgba: rgba(0, 0, 0, 0.3);
74
75  /*-----Borders-----*/
76  --color-border: var(--ang-color-border-button);
77
78  /*-----Dividers/HR-----*/
79  --ang-color-divider-light: rgba(0, 0, 0, 0.29);
80 }

```

```

styles.scss ×
pwa > src > styles.scss > ...
82  /*-----Semantic Fonts-----*/
83  :root {
84    /*-----Roboto-----*/
85    --ang-font-body-text: 'roboto';
86    --ang-body-font-weight-thin: 100;
87    --ang-body-font-weight-light: 300;
88    --ang-body-font-weight-regular: 400;
89    --ang-body-font-weight-medium: 500;
90    --ang-body-font-weight-bold: 700;
91
92    /*-----Roboto Condensed-----*/
93    --ang-font-button-text: 'roboto condensed', sans-serif;
94    --ang-button-font-weight-light: 300;
95    --ang-button-font-weight-regular: 400;
96    --ang-button-font-weight-bold: 700;
97
98    /*-----Roboto Serif-----*/
99    --ang-font-serif-text: 'roboto serif', serif;
100   --ang-serif-font-weight-thin: 100;
101   --ang-serif-font-weight-extra-light: 200;
102   --ang-serif-font-weight-light: 300;
103   --ang-serif-font-weight-regular: 400;
104 }
105
106 @import url('https://fonts.googleapis.com/css?family=Roboto:wght@100;300;400;500;700&display=swap');
107 @import url('https://fonts.googleapis.com/css?family=Roboto+Condensed:wght@300;400;700&display=swap');
108 @import url('https://fonts.googleapis.com/css?family=Roboto+Serif:opsz,wght@..144,100;..144,200;..144,300;..144,400&display=swap');
109
110 @font-face {
111   font-family: "Charlotte Southern";
112   src: url("assets/fonts/Southern/Southern.ttf") format("ttf");
113 }

```

J'ai ensuite effectué un reset des plus basiques.

```

/*-----Reset-----*/
*,
*:before,
*:after {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

```

Puis j'ai développé maintes classes utilisables dans toute l'application ainsi que du style défini et appliqué sur différentes balises HTML.

```

/*-----Body-----*/
body {
  height:100%;
  color: var(--color-text);
  font-family: var(--ang-font-body-text);
  font-weight: var(--ang-body-font-weight-light);
  font-size: 15px;
  line-height: 1.6;
  background: var(--color-background);
  transition: color 0.5s, background-color 0.5s;
  text-rendering: optimizeLegibility;
  overflow-x: hidden;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  .container-custom {
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;
    margin-right: auto;
    margin-left: auto;
  }
}

```

```
/*-----Title-----*/
h1 {
  font-size: 2.5rem;
  line-height: 1.2;
  letter-spacing: 1.4px;
}
.form-title {
  font-family: "Charlotte Southern";
  font-size: 3.5rem;
  font-weight: 900;
  color: var(--ang-color-green);
  margin-bottom: 1.5rem;
}
.form-title-content {
  position: relative;
  color: var(--ang-color-green);
  text-transform: uppercase;
  border-bottom: 2px solid var(--ang-color-green);
  top: -2.2rem;
  .p-form-title-content {
    font-family: "Charlotte Southern";
    font-size: 1.7rem;
    font-weight: 700;
    letter-spacing: 1.2px;
    padding-bottom: 2rem;
  }
}
```

```
/*-----Border-----*/
.form-border {
  border: 2px solid var(--ang-color-green);
  border-radius: 10px;
  -webkit-border-radius: 10px;
}
/*-----Shadows-----*/
.button-shadow{
  box-shadow: 0 8px 10px -6px var(--ang-color-button-shadow-rgba);
  -webkit-box-shadow: 0 8px 10px -6px var(--ang-color-button-shadow-rgba);
  &:hover {
    box-shadow: 0 8px 20px -3px var(--ang-color-button-shadow-rgba);
    -webkit-box-shadow: 0 8px 20px -3px var(--ang-color-button-shadow-rgba);
  }
}
.input-inset-shadow{
  box-shadow: inset 0 0px 20px 7px var(--ang-color-green-rgba);
  -webkit-box-shadow: inset 0 0px 20px 7px var(--ang-color-green-rgba);
}
.title-form-shadow {
  text-shadow: 0px 5px 20px var(--ang-color-title-form-shadow);
  letter-spacing: -0.05em;
  user-select: none;
  transition: all 0.25s ease-out;
  &:hover {
    text-shadow: 0px 5px 20px var(--ang-color-green);
  }
}
.form-shadow {
  -webkit-box-shadow: 0 30px 60px 0 var(--ang-color-form-shadow-rgba);
  box-shadow: 0 30px 60px 0 var(--ang-color-form-shadow-rgba);
}
```

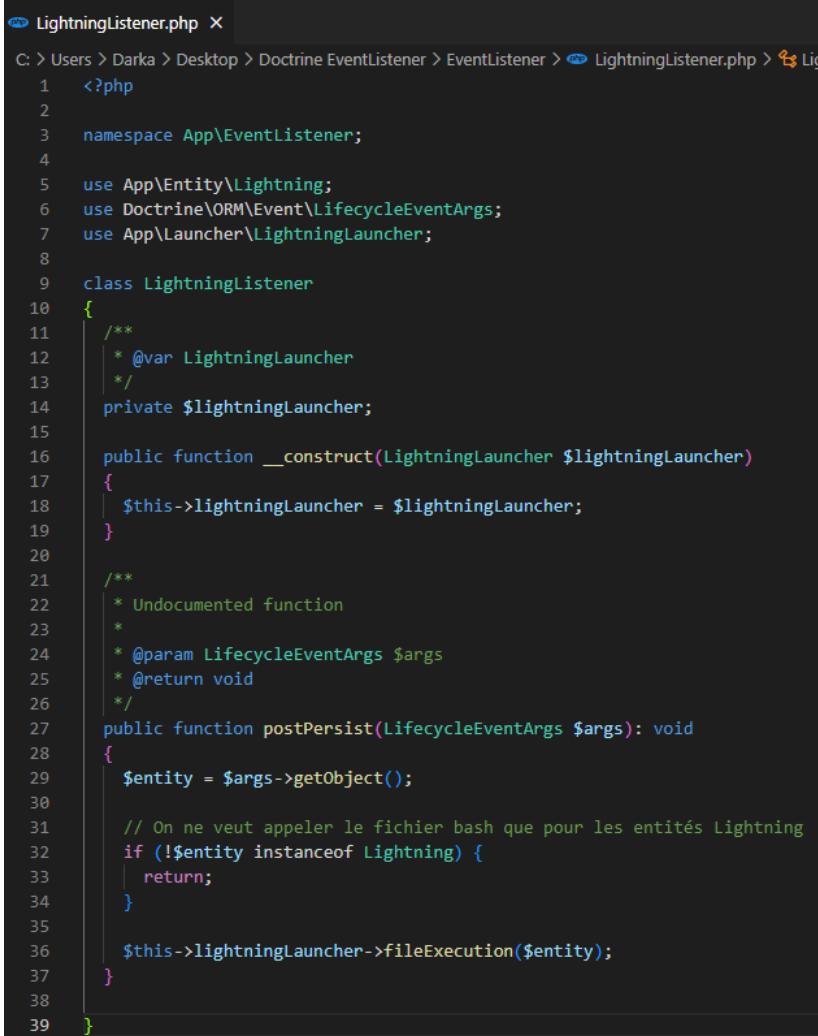
Comme énoncé plus haut j'ai créé un fichier panel.scss pour le style des tableaux. Les tableaux n'étant présent qu'une fois connecté à l'application je l'ai donc inclus dans le dossier easygarden. J'ai aussi utilisé les fichiers .scss générés lors de la création d'un composant afin de n'appliquer du style que sur celui-ci, par exemple sur le composant Home où j'ai cette fois utilisé le système de grid au lieu de flexbox. Ces deux extraits sont disponibles dans les annexes.

5.4 Hardware

La partie hardware est quant à elle encore en cours de développement, je ne présenterai donc ici que la manière dont j'envisage de développer et faire fonctionner celle-ci. Elle est composée de deux parties distinctes que j'expose ici séquentiellement:

Une partie dite descendante:

- 1/ Action sur l'IHM de l'utilisateur (allumer ou éteindre un équipement).
- 2/ Créer un EventListener Doctrine sur la propriété **status** de chaque équipement.



```

LightningListener.php ×
C: > Users > Darka > Desktop > Doctrine EventListener > EventListener > LightningListener.php > ⌂ Li
  1  <?php
  2
  3  namespace App\EventListener;
  4
  5  use App\Entity\Lightning;
  6  use Doctrine\ORM\Event\LifecycleEventArgs;
  7  use App\Launcher\LightningLauncher;
  8
  9  class LightningListener
 10 {
 11     /**
 12      * @var LightningLauncher
 13      */
 14     private $lightningLauncher;
 15
 16     public function __construct(LightningLauncher $lightningLauncher)
 17     {
 18         $this->lightningLauncher = $lightningLauncher;
 19     }
 20
 21     /**
 22      * Undocumented function
 23      *
 24      * @param LifecycleEventArgs $args
 25      * @return void
 26      */
 27     public function postPersist(LifecycleEventArgs $args): void
 28     {
 29         $entity = $args->getObjectType();
 30
 31         // On ne veut appeler le fichier bash que pour les entités Lightning
 32         if (!$entity instanceof Lightning) {
 33             return;
 34         }
 35
 36         $this->lightningLauncher->fileExecution($entity);
 37     }
 38 }
 39 }
```

- 3/ Déclencher un scriptLauncher contenant un string avec l'instruction.

```

LightningLauncher.php X
C: > Users > Darka > Desktop > Doctrine EventListener > Launcher > LightningLauncher.php > ...
1  <?php
2
3  namespace App\Launcher;
4
5  use App\Entity\Lightning;
6
7  class LightningLauncher
8  {
9
10     public function fileExecution(Lightning $status)
11     {
12         dump($status);
13         shell_exec('/bin/sh -c /var/www/e_reveil/script/sbin/appele_cron.sh');
14     }
15
16 }

```

4/ Envoyer ce script sur le Raspberry de manière sécurisé (SSH).

5/ Déterminer l'action souhaité par l'utilisateur et la router vers le bon Arduino.

6/ L'Arduino effectue l'action demandée.

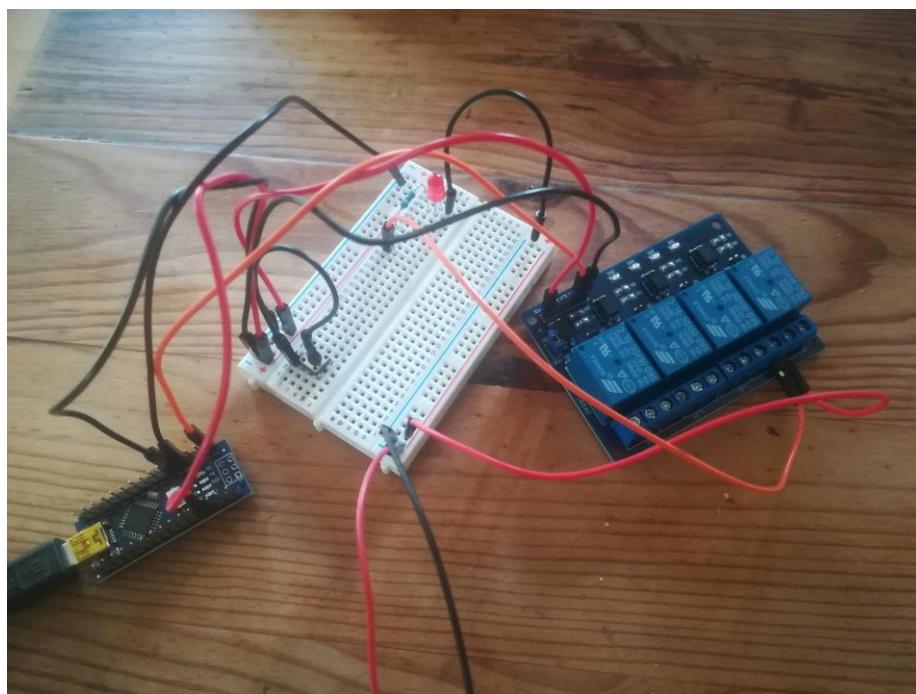
Une partie dite montante:

1/ Renvoyer un message de confirmation d'exécution et d'autre part les données des capteurs au Raspberry.

2/ Depuis le Raspberry se connecter au back-end pour persister les données en BDD.

J'ai inclus dans les annexes un screen du logiciel Arduino Ide dans lequel j'ai développé le code permettant d'allumer ou éteindre une LED

Voici une photo du montage arduino.



6. Remerciements

Je tiens en premier lieu à remercier Mr Thomas Delalbre mon formateur pour son temps passé à nos côtés malgré la charge administrative qui lui incombe.

Mon maître de stage Mr Nicolas Clément pour m'avoir accueilli au sein de son association et fait découvrir le monde de la robotique et de la domotique de la manière la plus pédagogique qui soit. A sa femme Mme Christelle Villeneuve (conseillère en communication et design graphique) pour nos échanges au sujet du design, de l'ergonomie ou encore de l'expérience utilisateur.

Je remercie également le professeur de PHP et JavaScript Mr Thomas Mouchelet, le professeur de HTML/Css Mr Axel Gromat ainsi que tout mes camarades de formation et plus particulièrement Mr Sofiane Khlafi pour son aide et ses conseils précieux.

Un grand merci à vous tous pour votre soutien et nos échanges enrichissants.

7. Conclusion

En conclusion cette année fut très intense et m'a réellement permis d'enrichir mes connaissances ainsi que mes compétences qui n'était en fin de compte jusque là que sommaire.

Aborder ce projet d'un bout à l'autre m'a permis de commencer à mettre en place une méthodologie complète et une rigueur absolument nécessaire à la production d'une application multicouche.

Le monde du développement n'a donc pas fini de ma passionner et je compte à présent continuer à me former dans ce domaine en effectuant un master par alternance à la rentrée.

8. Annexes

Maquette zoning:

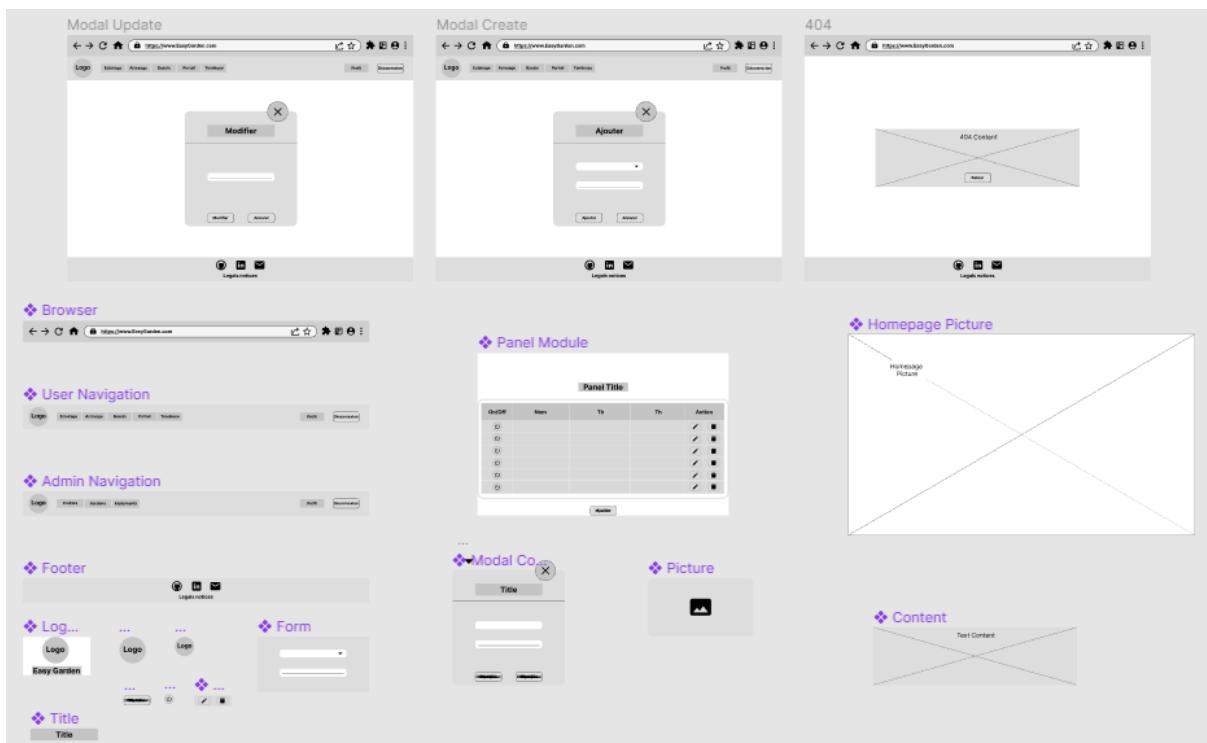


Diagramme d'activité Login:

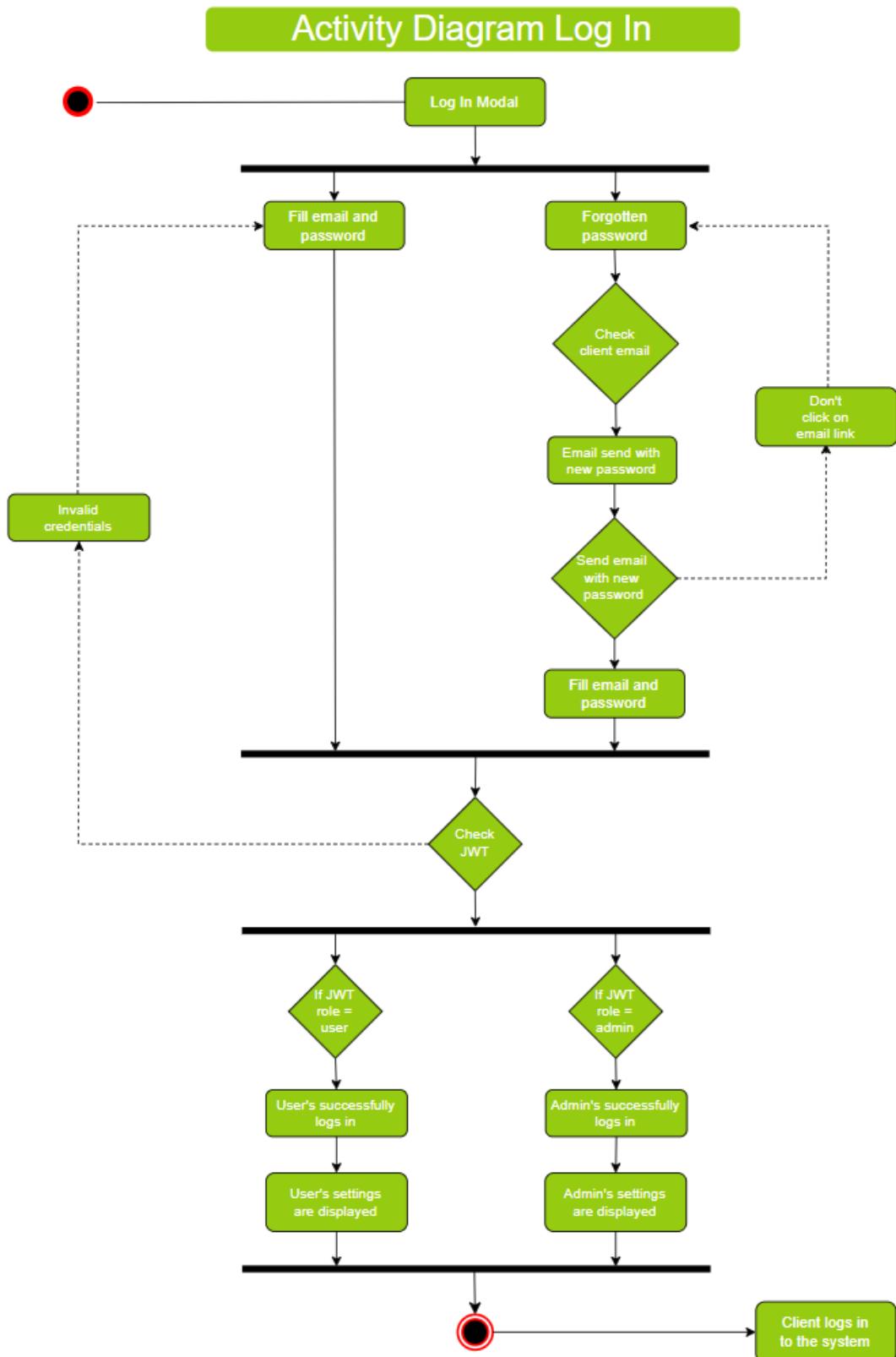


Diagramme d'activité Profile:

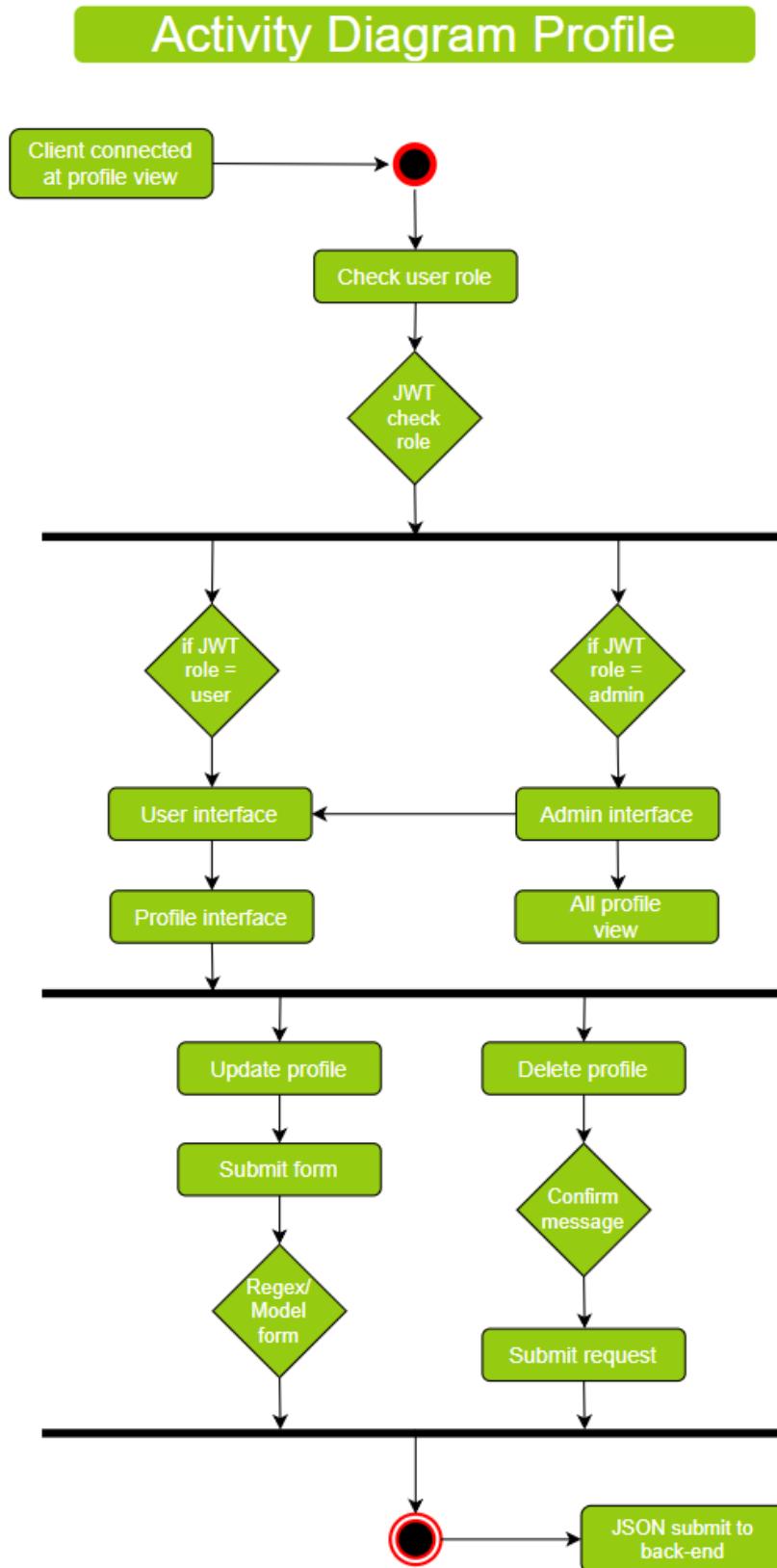
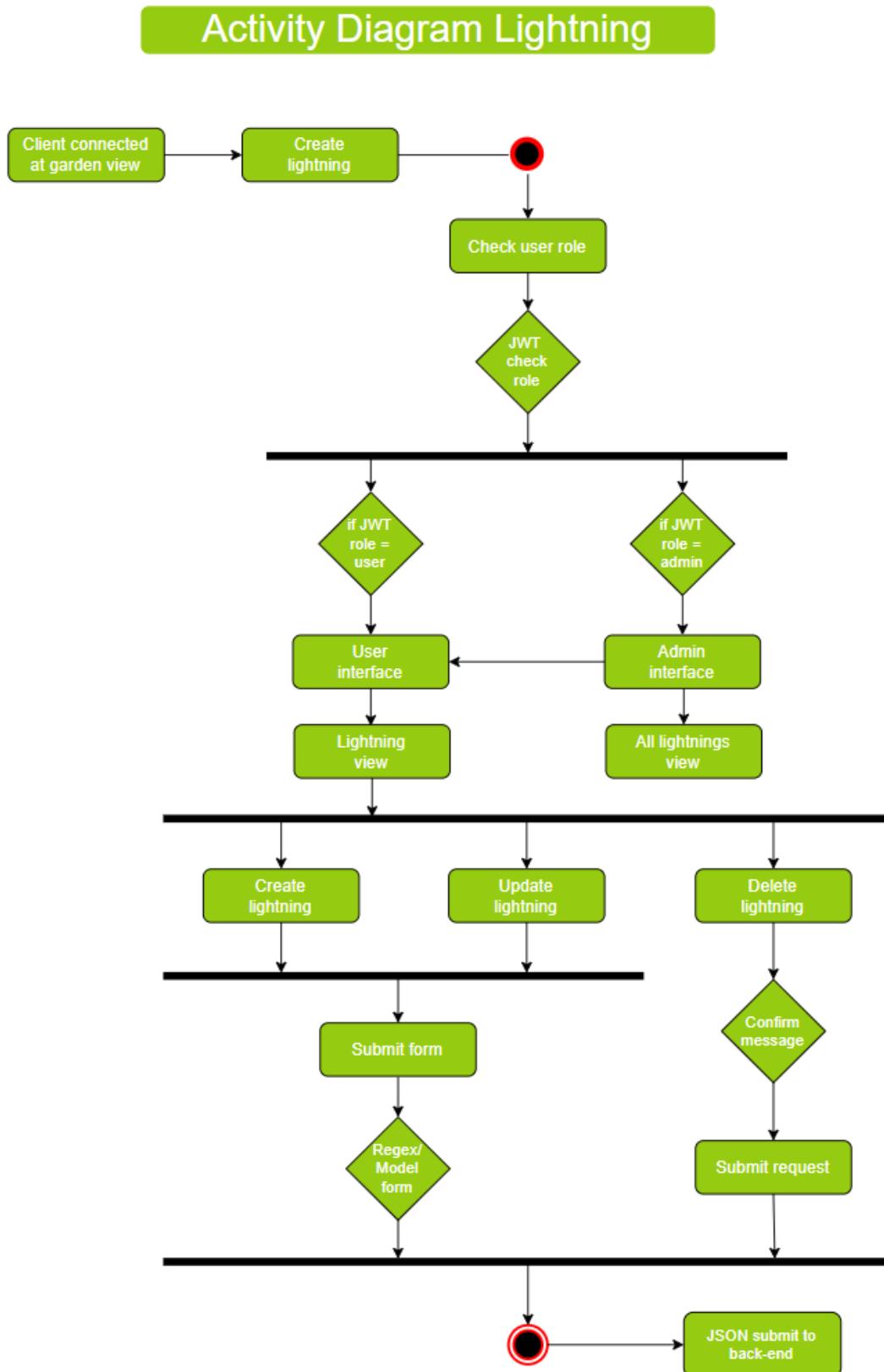


Diagramme d'activité Eclairage:



form-validation.service.ts:

```

import { Injectable } from '@angular/core';
import { AbstractControl, ValidatorFn } from '@angular/forms';

@Injectable({
  providedIn: 'root'
})
export class FormValidationService {

  passwordMatch(password: string, confirmPassword: string):ValidatorFn {
    return (formGroup: AbstractControl): { [key: string]: any } | null => {
      const passwordControl = formGroup.get(password);
      const confirmPasswordControl = formGroup.get(confirmPassword);

      if (!passwordControl || !confirmPasswordControl) {
        return null;
      }

      if (
        confirmPasswordControl.errors &&
        !confirmPasswordControl.errors['passwordMismatch']
      ) {
        return null;
      }

      if (passwordControl.value !== confirmPasswordControl.value) {
        confirmPasswordControl.setErrors({ passwordMismatch: true });
        return { passwordMismatch: true }
      } else {
        confirmPasswordControl.setErrors(null);
        return null;
      }
    };
  }

  strongPassword(): ValidatorFn {
    return (control: AbstractControl): { [key: string]: boolean } | null => {
      if (control.value == '') return null;

      let re = new RegExp('^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#$%^&*])[A-Za-z\d:;.-!@#$%^&*]{8,40}$');
      if (re.test(control.value)) {
        return null;
      } else {
        return { strongPassword: true };
      }
    };
  }

  validEmail(): ValidatorFn {
    return (control: AbstractControl): { [key: string]: boolean } | null => {
      if (control.value == '') return null;

      let re = new RegExp(`^([a-zA-Z-.%+-]+@[a-zA-Z-.]+\.[a-zA-Z]{2,4})$`);
      if (re.test(control.value)) {
        return null;
      } else {
        return { validEmail: true };
      }
    };
  }

  validPhoneNumber(): ValidatorFn {
    return (control: AbstractControl): { [key: string]: boolean } | null => {
      if (control.value == '') return null;

      let re = new RegExp(`[- .()0-9]{8,12}`);
      if (re.test(control.value)) {
        return null;
      } else {
        return { validPhoneNumber: true };
      }
    };
  }

  validName(): ValidatorFn {
    return (control: AbstractControl): { [key: string]: boolean } | null => {
      if (control.value == '') return null;

      let re = new RegExp(`^([a-zA-Z]{3,20})$`);
      if (re.test(control.value)) {
        return null;
      } else {
        return { validName: true };
      }
    };
  }

  validPseudo(): ValidatorFn {
    return (control: AbstractControl): { [key: string]: boolean } | null => {
      if (control.value == '') return null;

      let re = new RegExp(`^([0-9a-zA-Z]{3,10})$`);
      if (re.test(control.value)) {
        return null;
      } else {
        return { validPseudo: true };
      }
    };
  }

  validEquipmentName(): ValidatorFn {
    return (control: AbstractControl): { [key: string]: boolean } | null => {
      if (control.value == '') return null;

      let re = new RegExp(`^([0-9a-zA-Z ])*`);
      if (re.test(control.value)) {
        return null;
      } else {
        return { validEquipmentName: true };
      }
    };
  }
}

```

home.component.scss:

```

#homePage {
  background: linear-gradient(rgba(0, 0, 0, 0.40),rgba(0, 0, 0, 0.40
)), url("../ ../../assets/img/KuangSi.jpg");
  background-size: cover;
  background-repeat: no-repeat;
  height: 87vh;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 0.2fr 0.3fr 0.6fr;
  grid-gap: 2rem 1rem;
  .connexion {
    grid-area: 1/4/2/5;
    display: grid;
    grid-template-columns: 1fr 1fr;
    place-items: center;
    grid-gap: 3rem 1rem;
    .icon {
      grid-area: 1/2;
      margin-top: 2vh;
      .button-circle {
        font-size: 3rem;
        color: var(--ang-color-white);
        border: 2px solid var(--color-border);
        border-radius: 50%;
        margin: 0px;
        padding: 0px;
        &:hover {
          color: var(--ang-color-green);
          background: var(--ang-color-white);
        }
      }
      fa-icon {
        display: flex;
        align-items: center;
        justify-content: center;
      }
    }
    p {
      color: var(--ang-color-green);
      font-size: 1rem;
      font-weight: bold;
      display: flex;
      justify-content: center;
    }
  }
}

.logo {
  grid-area: 2/1/3/5;
  margin: auto;
  font-family: "Charlotte Southern";
  -webkit-text-stroke: 1px var(--color-border);
  font-weight: 200;
  font-size: 10rem;
  color: var(--ang-color-green);
}

.resume {
  grid-area: 3/2/4/4;
  color: var(--ang-color-white);
  text-align: center;
  button {
    margin-top: 10vh;
  }
}

```

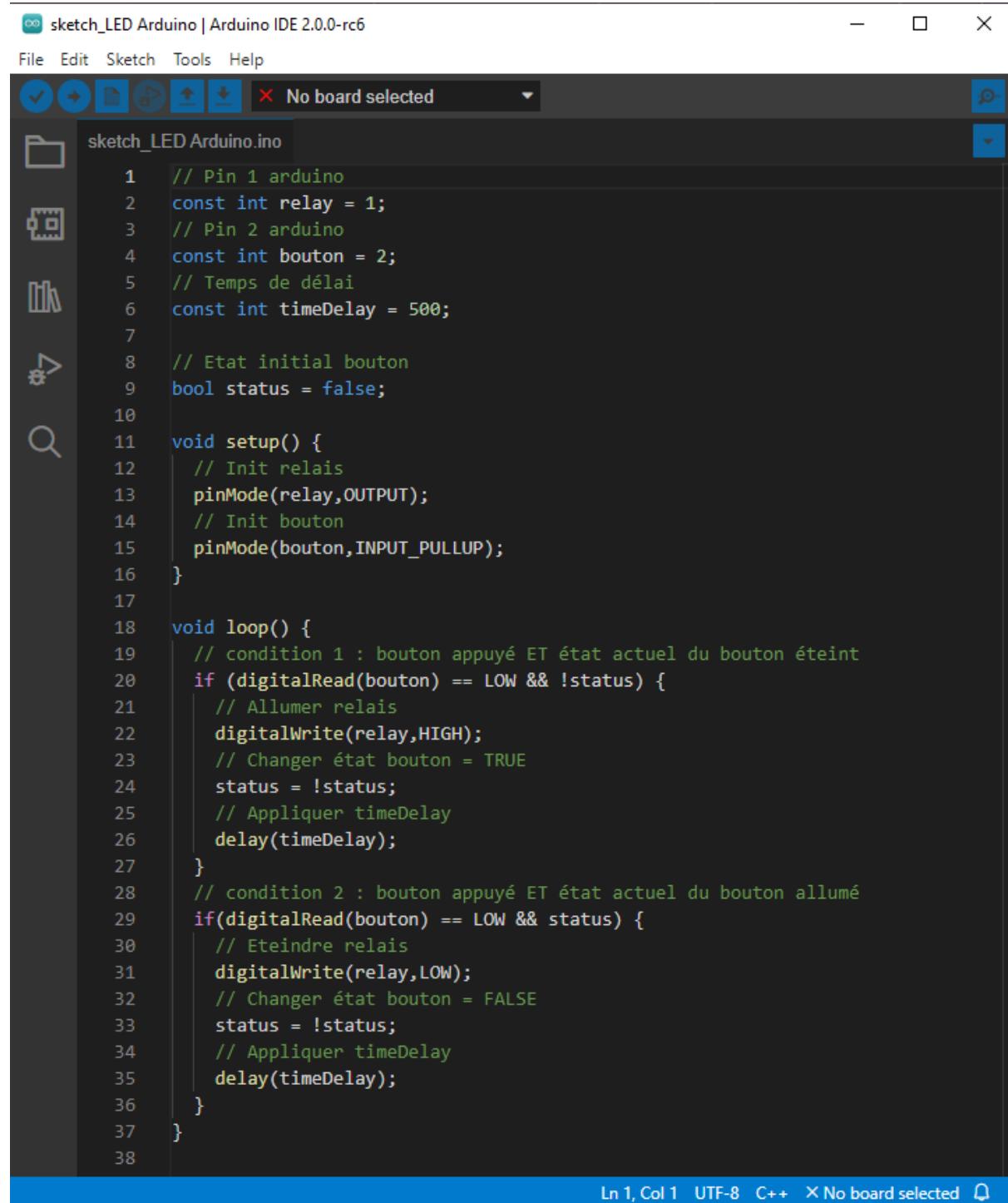
panel.scss:

```

div {
  .logo-panel {
    width: 110px;
    height: 110px;
    margin-bottom: 1vh;
  }
  hr {
    width: 60%;
    margin-top: 4px;
    margin-left: 20%;
    border: 0;
    height: 2px;
    background: #95c11b;
    background: -webkit-linear-gradient(left, #eef9d2, #cdec79,
#b4e236, #95c11b 50%, #b4e236, #cdec79, #eef9d2);
    background: -moz-linear-gradient(left, #eef9d2, #cdec79,
#b4e236, #95c11b 50%, #b4e236, #cdec79, #eef9d2);
    background: -ms-linear-gradient(left, #eef9d2, #cdec79,
#b4e236, #95c11b 50%, #b4e236, #cdec79, #eef9d2);
    background: -o-linear-gradient(left, #eef9d2, #cdec79,
#b4e236, #95c11b 50%, #b4e236, #cdec79, #eef9d2);
  }
  h1 {
    font-family: var(--ang-font-body-text);
    font-weight: var(--ang-body-font-weight-medium);
    font-size: 40px;
    color: #008080;
    -webkit-text-stroke: 1.5px;
    -webkit-text-stroke-color: var(--ang-color-green);
    text-shadow: 0 0 0.5em var(--ang-color-title-tab-rgba);
  }
  .table {
    margin: 3vh 0 1vh 0;
    padding: 3px;
    table {
      border-collapse: collapse;
      thead {
        th {
          font-family: var(--ang-font-serif-text);
          font-weight: var(--ang-serif-font-weight-regular);
          font-size: 1.1rem;
          color: var(--ang-color-white);
          background-color: var(--ang-color-green);
          border: 2px solid var(--ang-color-white);
          height: 6vh;
        }
      }
      tbody {
        font-family: var(--ang-font-body-text);
        font-weight: var(--ang-body-font-weight-thin);
        font-size: 14px;
        &:hover {
          background-color: var(--ang-color-table-hover);
        }
        tr:nth-child(odd) {
          background-color: #e0e2e5;
        }
        tr:nth-child(even){
          color: #666666;
          background-color: #f7f7f7;
        }
      }
      td {
        border: 2px solid var(--ang-color-white);
        height: 5vh;
        button {
          background-color: var(--ang-color-border-button);
          border: 2px solid var(--ang-color-green);
          margin: 0 0.8rem;
          padding: 0.3rem 0.7rem;
          &:hover {
            background-color: var(--ang-color-white);
          }
          fa-icon {
            font-size: 1.2rem;
            color: var(--ang-color-grey);
          }
        }
        button:hover fa-icon {
          color: var(--ang-color-green)
        }
        button.trash {
          &:hover {
            border-color: var(--ang-color-alert);
          }
        }
        button.trash:hover fa-icon {
          color: var(--ang-color-alert)
        }
      }
    }
  }
}

th, td {
  width: 10rem;
  max-width: 15rem;
  text-align: center;
}
table.t-hover tr:hover {
  background-color: var(--ang-color-table-hover);
}
table>thead>tr>th.power,
table>tbody>tr>td.power {
  width: 7rem;
  button {
    border-radius: 50%;
    &:hover {
      background-color: var(--ang-color-alert-rgba);
      border-color: var(--ang-color-alert2);
      -webkit-animation: pulse-red 3s infinite;
      animation: pulse-red 3s infinite;
    }
    fa-icon {
      color: var(--ang-color-grey);
    }
  }
}
button.btn-off {
  background-color: rgba(193, 193, 193, 0.5);
  border-color: rgba(255, 15, 15, 0.2);
  &:hover {
    background-color: var(--ang-color-border-button);
    border-color: var(--ang-color-green);
    -webkit-animation: pulse-green 3s infinite;
    animation: pulse-green 3s infinite;
  }
}
button:hover fa-icon {
  color: var(--color-text);
}
table>tbody>tr>td.alert {
  color: var(--ang-color-alert);
  box-shadow: inset 0 0px 6px 2px var(--ang-color-alert);
  -webkit-box-shadow: inset 0 0px 10px 2px var(--ang-color-alert);
  background-color: #f7f7f7;
}
div.submit {
  button {
    font-size: 1.1rem;
    width: 7rem;
    margin: 1rem 0 1rem 0;
  }
}
div.pagination {
  margin-top: 0.5vh;
  pagination-controls {
    nav {
      padding: 0.3vh 0.2vh;
      ul {
        margin-bottom: 0;
        li {
          font-size: 14px;
          margin: 0 2px;
          a {
            &:hover {
              color: var(--ang-color-white);
              background-color: var(--ang-color-green);
              border-radius: 6px;
            }
          }
        }
      }
    }
  }
  pagination-controls .current {
    background-color: var(--ang-color-green);
    border-radius: 6px;
  }
}

```

Arduino ide:


The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** sketch_LED Arduino | Arduino IDE 2.0.0-rc6
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Includes icons for Open, Save, Print, and Upload.
- Status Bar:** No board selected
- Code Editor:** Displays the following C++ code for a sketch named "sketch_LED Arduino.ino".

```

1 // Pin 1 arduino
2 const int relay = 1;
3 // Pin 2 arduino
4 const int bouton = 2;
5 // Temps de délai
6 const int timeDelay = 500;
7
8 // Etat initial bouton
9 bool status = false;
10
11 void setup() {
12     // Init relais
13     pinMode(relay,OUTPUT);
14     // Init bouton
15     pinMode(bouton,INPUT_PULLUP);
16 }
17
18 void loop() {
19     // condition 1 : bouton appuyé ET état actuel du bouton éteint
20     if (digitalRead(bouton) == LOW && !status) {
21         // Allumer relais
22         digitalWrite(relay,HIGH);
23         // Changer état bouton = TRUE
24         status = !status;
25         // Appliquer timeDelay
26         delay(timeDelay);
27     }
28     // condition 2 : bouton appuyé ET état actuel du bouton allumé
29     if(digitalRead(bouton) == LOW && status) {
30         // Eteindre relais
31         digitalWrite(relay,LOW);
32         // Changer état bouton = FALSE
33         status = !status;
34         // Appliquer timeDelay
35         delay(timeDelay);
36     }
37 }
38

```

The status bar at the bottom indicates: Ln 1, Col 1 UTF-8 C++ X No board selected Q