# Green City

Rapport du Laboratoire d'informatique ambiante et mobile

UNIVERSITÉ DE NAMUR

Groupe 9 :
DÉSIRON Kevin
LEMAL Amélie
MARCHAL Hugues
SFORZA Syméon

UNIVERSITÉ DE NAMUR
Faculté d'Informatique
Academic Year 2016–2017

# Table des matières

# Chapitre 1

# Description du projet

Dans le cadre du laboratoire d'informatique ambiante et mobile, il nous était demandé de développer une solution connectée pour répondre à une problématique liée aux Smart Cities. Basé sur cette consigne initiale, notre groupe a décidé de s'intéresser au problème de la gestion des espaces verts dans les Smart Cities et plus particulièrement sur la manière de les valoriser au bénéfice des citoyens.

Actuellement, les espaces verts sont souvent pensés de manière à embellir la ville. En effet, ces espaces sont presque exclusivement limités aux parcs et à des emplacements fleuris. Cependant, la plus-value offerte aux citoyens par ces aménagements est très limitée, les parcs offrant tout au plus un cadre de promenade agréable.

Afin de valoriser davantage ces espaces, nous avons proposé un projet de serre intelligente. Dans notre projet de départ, l'idée était de créer des serres pour permettre aux restaurateurs d'avoir des fruits et légumes disponibles à proximité. Néanmoins, l'accès à ces serres intelligentes peut être étendu à l'ensemble des citoyens d'une ville ou à l'ensemble d'une communauté.

Dans ces serres, les plantes sont monitorées via de nombreux capteurs permettant au système de faire des mesures des conditions environnementales. Grâce à ces mesures et une base de connaissance, le système est capable de déterminer si les conditions dans la serre sont optimales pour la plante. Si ce n'est pas le cas, le système peut prendre des mesures correctives comme, par exemple, ouvrir le système d'arrosage pour irriguer la plante ou tendre une toile pour diminuer la luminosité. Dans l'hypothèse où le système ne serait pas capable d'agir sur l'environnement pour corriger les conditions environnementales, il peut alors envoyer une alerte aux administrateurs du système qui sont les jardiniers. De plus, la serre est capable de se connecter à une station météo afin d'interpoler les éventuelles données manquantes suite à une défaillance d'un ou plusieurs capteurs.

Afin d'offrir aux utilisateurs une interface avec le système, nous avons mis au point une application mobile offrant de nombreuses fonctionnalités. Ainsi, elle permet à l'utilisateur d'encoder ce qu'il a planté mais aussi d'en réserver le fruit s'il le désire. S'il le fait, il pourra alors suivre son évolution via l'application jusqu'à son arrivée à maturité. L'application offre aussi des possibilités pour voir les plantes prêtes à la collecte et disponibles pour tous au moyen d'une carte permettant de visualiser ce qui est disponible à proximité. Du côté administrateur, cette application permet aux jardiniers de consulter les alertes émises par le système ainsi que les notifications qu'il envoie lorsqu'il estime une plante à maturité. Ainsi, les jardiniers peuvent facilement voir les problèmes détectés afin de les résoudre et s'assurer de la maturité des plantes avant de les mettre à disposition des utilisateurs.

Finalement, un système d'identification au moyen de tags RFID est mis en place pour contrôler l'accès à la serre. Ceci permet d'identifier les utilisateurs mais aussi aux jardiniers de "déconnecter" une serre du système afin d'y effectuer des opérations de maintenance telles que l'amendement des sols.

# Chapitre 2

# Aspects du projet

Dans le cadre de ce projet, un certain nombre de pré-requis nous étaient demandés sous la forme d'un squelette pour notre projet. Dans ce chapitre, nous passerons en revue chacun de ces pré-requis et exposerons comment nous y avons répondu.

## 2.1 Phidgets

En ce qui concerne les Phidgets, trois points nous étaient demandés :

- Déployer au moins une des cartes des Phidgets connecté au Raspberry Pi
- Récupérer des informations depuis au moins deux capteurs environnementaux et les transmettre
- Lire des informations sur un tag RFID

Les deux premiers points étaient facile à remplir car ils entraient parfaitement dans le cadre de notre projet. En effet, c'est le Raspberry Pi qui est chargé du monitoring d'un emplacement et donc d'une plante. Par conséquent, nous avons déployé l'ensemble des capteurs au niveau du Raspberry Pi via l'interface kit des Phidgets mais aussi les servo-moteurs servant d'actuateur grâce à leur carte spécifique.

Grâce à cette installation, nous pouvions récupérer les mesures des capteurs de température et de luminosité ainsi que celles des capteurs d'humidité et de pH que nous simulions grâce à des sliders. Cependant, le Raspberry n'ayant pas les connaissances nécessaires pour exploiter ces données, elles sont envoyées régulièrement au format JSON au serveur qui indiquera au Raspberry Pi quelle politique adopter pour, éventuellement, corriger l'environnement de la serre.

Afin d'intégrer les tags RFID au projet, nous avons ajouté une identification au moyen de RFID pour accéder à la serre. Cet ajout nous permet d'une part d'ajouter l'aspect de gestion des accès et d'identification à notre projet et, d'autre part, d'intégrer le RFID conformément à ce qui nous était demandé.

## 2.2 Server/Scala

Pour le serveur, partie centrale du projet, nous devions remplir quatre objectifs :

- Définir un protocole de communication entre les différents composants du système
- Inférer des connaissances à partir d'au moins deux types de senseurs différents

— Gérer de manière élégante les connexions et déconnexions soudaines des capteurs
— Créer une API disponible via un webservice permettant à d'autres services de profiter du nôtre

Pour la communication entre les différents composants du système, nous avons décidé d'opter pour l'utilisation d'un webservice RESTful. L'utilisation d'un webservice RESTful nous permet de nous baser sur le protocole HTTP pour la communication entre les différents composants du système, assurant ainsi un excellente interopérabilité entre les différents éléments.

Grâce à ce protocole de communication, notre serveur est capable de recevoir les mesures prises à la fois par notre Arduino servant de station météo et par les Raspberry Pi. A partir de sa base de connaissance et des mesures provenant des capteurs de luminosité, d'humidité, de pH et de température, le serveur va être capable d'inférer les besoins de la plante et sa maturité. Par exemple, à partir du capteur de luminosité et celui de température, le serveur est capable d'inférer les conditions d'exposition de la plante ou, à partir des capteurs d'humidité et de pH, si la plante reçoit une alimentation correcte.

La gestion des déconnexions et reconnexions soudaines des capteurs est, quant à elle, gérée au niveau du Raspberry Pi. Pour ce faire, nous avons utilisé les listeners d'évènements, notamment de connexion et déconnexion. Ceci permet notamment de transmettre l'information au serveur que les capteurs sont perdus.

Finalement, nous avons créé une API disponible via un webservice RESTful permettant d'une part à l'Arduino et au Raspberry d'envoyer leurs données au serveur mais aussi, d'autre part, à l'application mobile d'accéder aux données afin d'offrir une interface du système à l'utilisateur.

## 2.3 Autres aspects

Finalement, deux derniers points nous étaient demandés dans ce projet squelette.

— Communiquer avec l'Arduino au moyen de sa connexion WiFi
— Inclure une application à destination de l'utilisateur en prêtant attention à l'interface utilisateur

Comme nous l'avons expliqué dans la partie sur le serveur, nous utilisons la connexion WiFi de l'Arduino afin d'accéder à un webservice RESTful pour envoyer ses données au serveur.

Nous avons finalement développé une application mobile à destination des utilisateurs utilisant notre API. Grâce à celle-ci, l'utilisateur a accès à de nombreuses fonctionnalités exposées dans les chapitres suivants.

# Chapitre 3

# Architecture matérielle

C'est le moment de détailler la structure matérielle de notre projet. Nous rappelons que ce matériel est en premier temps utilisé pour une maquette, la puissance de certains moteurs ne peut donc pas élever de charges importantes. Nous distinguerons deux structures distinctes. La première, l'Arduino, sert de station météo tandis que la seconde est la serre « connectée », avec ses différentes fonctionnalités. Il est nécessaire de savoir qu'aucune ligne de code ne sera ici ni expliquée ni citée.

## 3.1 Arduino, station météorologique

Pour commencer, l'Arduino fait bande à part car il ne devrait pas être utilisé lors d'un fonctionnement normal du système ; il sert de back-up sur les données météorologique lorsque nos senseurs dans la serre nous font défaut. Ainsi, ce dernier est muni de deux capteurs ; un capteur qui récupère la température et l'humidité, un capteur pour la luminosité.

## 3.2 Serre « connectée »

La serre est une structure en bois avec un toit d'une forte pente. Du plexiglas est placé sur la face gauche et sur le toit, tandis qu'au fond de la serre se trouve un panneau de bois afin d'y accrocher différents éléments.

Concernant les composants électroniques, nous placerons le Raspberry en son centre car c'est lui qui gérera toutes les fonctionnalités de la serre. À ce Raspberry se raccorderont 3 phidgets :

— deux phidgets de contrôle de servo moteur,
— un phidget d'interface kit.

Nous utilisons le phidget "interface kit" dans un premier temps pour récupérer les données de la serre grâce à deux capteurs, un de lumière et l'autre de température. Deux sliders sont utilisés et connectés pour simuler les données de pH et d'humidité pour lesquelles nous n'avons pas les capteurs. Ainsi nous avons de quoi connaître et simuler l'ambiance à l'intérieur de la serre.

Dans un second temps "l'interface kit" nous est utile pour activer deux fonctionnalités de la serre. Premièrement une LED, mise sur la plaque de fond de la serre, est raccordée dans les Output G et 0 de l'interface kit. En activant la borne 0 nous pouvons donc allumer la LED pour éclairer la serre. Deuxièmement, nous effectuons

un branchement avec un moteur DC pour descendre et lever un volet qui occulte la moitié du toit. Pour ce faire, nous raccordons le moteur à un transistor dont la « Gate » est connectée à l'Output 5 de l'interface kit, le circuit passe par la borne 5V et G de l'input pour avoir le courant nécessaire pour faire tourner le moteur. Avec ce branchement, en activant la borne 5, nous activons le transistor qui laissera le courant de 5V passer et fera tourner le moteur pour enrouler la toile. Quand la borne 5 est désactivée, le moteur ne tourne plus et le poids placé au bout de la toile fait descendre cette dernière et occulte une partie de la serre.

Enfin, deux servo-moteurs sont reliés au Raspberry au moyen de phidgets de contrôle afin d'en assurer la pleine maîtrise. Un premier servo, attaché sur le fond de la serre, servira à abaisser un pot pour verser de l'eau dans la serre. Le servo est à l'état de repos et le pot droit - vertical, l'ouverture vers le haut - lorsque celui-ci est à sa position de 180 degrés. Le pot renverse son contenu lorsque sa position est à 60 degrés. Le second servo gère, au moyen d'une corde, l'ouverture de la fenêtre qui se trouve sur la face gauche de la serre. Lorsque le servo est à sa position de 0 degré, la corde est tendue et ferme la fenêtre. Mais lorsque le servo est à sa position de 180 degrés, la corde est lâche et permet à la fenêtre de s'entrouvrir.

# Chapitre 4

# Architecture logicielle

## 4.1 Serveur

### 4.1.1 Fonctionnement général

Les sections ci-après ne traiteront pas de l'implémentation complète du serveur, mais plutôt de son fonctionnement ainsi que de son architecture générale. Pour effectivement prendre connaissance de l'implémentation pointue système, il convient de se référer aux différents fichiers concernés.

Afin de développer le système REST tel qu'expliqué dans le chapitre précédent, le "framework play" a été utilisé dans sa version 2.5. Ce cadre permet de créer efficacement un serveur sous forme d'API REST traitant des requêtes HTTP. Il contient 4 parties principales, à savoir le fichier de routes dans le dossier de configuration, le package "controllers", le package "models" et une base de données MySQL. Une cinquième partie y a été ajoutée sous le nom de "`utils_project`". Afin de bien comprendre le fonctionnement de ce serveur, les parties principales citées ci-dessus sont ici expliquées.

Il est important également de savoir que le serveur a été codé entièrement localement, et a été prévu pour des accès locaux via une connexion WiFi partagée - le serveur tournant sur l'ordinateur partageant la connexion, son adresse IP étant donc 192.168.137.1.

Finalement, comme évoqué au cours, le serveur a bien été implémenté en scala.

### 4.1.2 Routes

Afin de gérer les différentes requêtes émises par l'application mobile, la serre et la station météo, des routes ont donc été ajoutées au fichier "routes". Ces dernières consistent réellement en la fin d'URL, comme par exemple "/login". Ce fichier va donc permettre de lier les requêtes aux méthodes implémentées dans le package "controllers". Pour chaque URL, le fichier doit contenir le type de requête attendu - i.e. POST, GET... -, l'URL, évidemment, ainsi que le chemin complet vers la méthode traîtant la requête à proprement parler.

### 4.1.3 Controllers

Ce package contient les différentes classes contenant elles-mêmes les méthodes traitant les requêtes reçues. Le nombre de classes importantes est de 5, la sixième, "HomeController", auto-générée, n'ayant pas été retirée. Si la méthode en question doit prendre une variable en paramètre, cette variable doit être mentionnée au préalable dans le fichier de routes ainsi que dans l'URL y associée. Voici, ci-dessous, la liste des classes scala contenant les méthodes des "controllers" :

- "HomeController" : auto-générée, sert notamment de page d'accueil par défaut.
- "Meteo" : contient les méthodes de traitement des requêtes de la station météo.
- "Phidgets" : contient les méthodes de traitement des requêtes directement liées aux Phidgets, comme par exemple, une mise-à-jour des valeurs captées.
- "Plante" : contient, comme son nom l'indique, les méthodes de traitement des requêtes liées aux plantes.
- "Raspberry" : contient les méthodes directement liées à la serre sans être liées aux Phidgets, comme la demande de la liste d'emplacements à gérer.
- "User" : contient toutes les méthodes liées aux requêtes émises par des utilisateurs et les concernant directement, comme par exemple, le login ou l'inscription.

Il est important de noter également qu'il existe sur le serveur une notion de session avec cookie. Effectivement, lors du login, un cookie signalant que l'utilisateur effectuant la requête possède une session est créé. Par la suite, lorsque des opérations nécessitant des actions physiques sur les plantes ou sur la serre sont demandées, le serveur vérifie que l'utilisateur est bel et bien connecté - et donc connu - pour traiter cette requête. Si tel n'est pas le cas, aucune action ne peut être effectuée, le serveur renvoyant à l'utilisateur une erreur d'authentification.

Cette dernière phrase soulève d'ailleurs un point très intéressant et pour le moins important : à chaque requête, le serveur répond toujours par un message adéquat (sous forme de réponse HTTP). Si des données doivent être renvoyées, elles le sont toujours dans le corps de la réponse, au format JSON. Il est aussi évident que le type de réponse est variable, par exemple, lorsqu'une erreur s'est présentée sur le serveur, une "InternalServerError" est renvoyée, dans le cas d'une mauvaise requête, une "BadRequest" - ce genre d'erreur est toujours renvoyé dans le cas d'une requête POST ne contenant pas le corps attendu -, dans le cas d'un mismatch de DB, une "NotFound", pour finir par une "Ok" lorsque tout s'est bien déroulé.

Tel qu'expliqué précédemment, le serveur ne fait que répondre aux requêtes envoyées. Toutefois, lorsqu'un utilisateur s'inscrit, le Raspberry doit être notifié du nouveau tag RFID créé, étant donné qu'il contrôle les accès à la serre. Pour ce faire, le serveur doit lui-même prendre l'initiative de contacter les différents Raspberry, ce qui n'est pas possible dans la configuration actuelle. Par conséquent, afin d'éviter de devoir créer un serveur sur les contrôleurs des serres pour seulement une requête, une connexion par socket a été établie. Effectivement, le serveur va ainsi directement aller chercher dans la base de données l'ensemble des adresses des Raspberry pour leur envoyer le nouveau tag sur un port prédéfini, à savoir le 6789.

### 4.1.4 Models

Alors que le dossier "controllers" contient toutes les méthodes de traitement, le package "models" contient quant à lui les classes constituant les objets de données à utiliser. Ce package sert donc uniquement à stocker des données sous forme d'objet et non à les traiter directement.

Dans le cas du projet, une seule classe a été nécessaire ; il s'agit de la classe "Plante". Effectivement, cette dernière permet de stocker, suite à la réception des données quantitatives d'une plante, les données type de la plante en question, dans le but de les comparer. Ce stockage temporaire permet effectivement une meilleure découpe des méthodes, ce qui améliore la compréhension et l'efficacité. Pour rendre ce point plus clair, il est important de situer le contexte d'action de cet objet. En effet, lorsqu'une requête de mise à jour des valeurs des senseurs du Raspberry PI connecté à la serre est reçue, la base de données doit non-seulement être mise à jour, mais également consultée pour vérifier la normalité de ces valeurs - grâce à des valeurs étalon stockées dans cette base de données. Suite à ces consultation et vérification, il est évidemment nécessaire de renvoyer l'état de normalité de ces valeurs dans le but de générer des actions ou non - ce qui permet, le cas échéant, de ramener les valeurs dans une bonne fourchette. Ainsi, par souci de clareté et de respect des principes basiques de programmation autorisant aisément la scission des méthodes en garantissant un accès aux données sensibles, un objet "Plante" a été créé. Ce dernier contient le nom de la plante concernée ainsi que les valeurs minimum et maximum autorisées de température, de luminosité, d'humidité et de pH.

### 4.1.5 Base de données MySQL

Il faut savoir que la base de données, ainsi que le reste, sont développés pour fonctionner sur un réseau local. Comme écrit dans le fichier de configuration du serveur, la base de données "`server_db`" est accessible localement via le port 3306. De plus, l'identifiant ainsi que le mot de passe nécessaires aux accès sont également présents dans ce fichier de configuration, permettant au serveur un accès direct.

La base de données compte 6 tables :

- "emplacement" : contient toutes les informations relatives à un emplacement, comme par exemple son identifiant, ses coordonnées GPS, s'il est accessible, rempli, quel est l'administrateur responsable, le raspberry le gérant...
- "meteo" : contient les données relatives à la station météo globale, à savoir, l'identifiant de la station (son adresse IP sur le réseau) ainsi que les données de luminosité, température et humidité.
- "plante" : contient les données d'une plante effectivement en terre, comme par exemple son nom, l'emplacement où elle est plantée, qui l'a plantée, ses données récoltées par phidget, si elle est prête...
- "`plante_type`" : contient toutes les données d'une plante type référencée par son nom, comme par exemple les valeurs minimales et maximales possibles pour la luminosité et autres, le temps de maturité, les mois de plantation et de retrait...
- "reservation" : contient les données relatives à la réservation d'une plante par l'utilisateur l'ayant mise en terre (et uniquement lui), à savoir la date de réservation, le tag RFID de l'utilisateur (unique), le nom de la plante ainsi que son emplacement.
- "users" : contient l'ensemble des données de l'utilisateur, comme son nom et prénom, son adresse e-mail utilisée comme clé primaire, son tag RFID (unique), son adresse, s'il est administrateur...

Pour plus d'informations, se référer au fichier "`server_db`.sql".

Il est important de savoir que l'identifiant d'un Raspberry est son adresse IP, ce qui permet de toujours savoir le contacter en cas de besoin, sans devoir le référencer outre mesure. De plus, l'identifiant d'un emplacement consiste en la concaténation de l'identifiant du raspberry et d'une chaîne de caractères aléatoires et unique générée par le serveur. En ce qui concerne le tag RFID de l'utilisateur, le serveur s'occupe de

le générer tout en vérifiant qu'il soit unique. Ce tag commencera toujours soit par 'u', soit par 'a' - lorsque l'utilisateur est respectivement soit un utilisateur, soit un administrateur -, pour terminer par une chaîne de 31 caractères parmi les lettres et chiffres.

Finalement, comme introduit ci-dessus, toutes les parties du système ont été développées pour fonctionner localement, et n'ont donc jamais été mises en phase de production. Cela explique les adresses IP fixées ainsi que certaines URL dans presque chaque partie du système.

### 4.1.6 "Utils_project"

Ce dossier du serveur consiste, comme son nom l'indique, en des classes - objets - utilitaires. L'objet "Sec" contient notamment des méthodes servant à vérifier que l'utilisateur est bien identifié sur le système, mais aussi s'il est administrateur ou non. Cet objet est donc principalement élaboré dans le but de garantir une sécurité minimale.

Une autre classe, "FirebaseService" sert à configurer le service Firebase développé par Google et permettant d'envoyer directement à un téléphone des notifications. Bien qu'une configuration ait été implémentée, aucun lien avec les classes et méthodes existantes n'a été créé, par manque de temps. Pour plus d'information, il convient de se référer au dernier chapitre.

## 4.2 Application mobile

L'application mobile joue 2 rôles dans ce projet. D'une part, elle sert d'interface à l'utilisateur afin qu'il puisse interagir avec les serres et les plantes s'y trouvant et, d'autre part, elle sert d'interface administrateur pour les jardiniers. Toutes les fonctionnalités implémentées visent à faire communiquer les utilisateurs de cette application avec la serre connectée (en passant par le serveur qui centralise toutes les données). Dans cette section, nous rappellerons brièvement les fonctionnalités de l'application avant d'expliquer la structure que celle-ci présente.

### 4.2.1 Fonctionnalités

Les fonctionnalités qui sont accessibles à l'utilisateur sont les suivantes :
- Login
- Inscription
- Ajout d'une plante
- Localisation des plantes et emplacements
- Consultation des réservations
- Retrait d'une plante

Les deux premières fonctionnalités de la liste sont assez explicites : la première sert à se connecter sur l'application, à l'aide de son e-mail et de son mot de passe, et la seconde sert à s'inscrire. Lors de l'inscription, il est demandé plusieurs informations à l'utilisateur : son nom, son prénom et son adresse afin de lui envoyer sa carte ou son tag RFID par courrier dès que celui-ci est prêt, son adresse mail qui sert d'identifiant et finalement un mot de passe secret.

L'ajout d'une plante présente deux listes déroulantes. La première contient la liste des plantes qui peuvent être ajoutées. En effet, afin que la serre puisse s'occuper d'une plante, il faut qu'elle contienne les données idéales de chaque plante. Dès lors, une liste des plantes connues est proposée à l'utilisateur. La seconde liste déroulante indique à l'utilisateur la liste des emplacements vides. Ce dernier peut alors sélectionner l'emplacement correspondant à celui où il désire mettre sa plante en terre. De plus, cette fonctionnalité propose à l'utilisateur de réserver ou non la plante qu'il désire ajouter. Cela lui permettra de pouvoir profiter des produits en premier.

La localisation des plantes et emplacements consiste en une carte localisant l'utilisateur (s'il active sa localisation sur son smartphone) et lui montrant les emplacements se trouvant à proximité. Différentes icônes sont visibles sur la carte : pour les emplacements vides, une petite serre vide sera affichée à l'écran, tandis que pour ceux qui possèdent une plante, une icône correspondant à cette dernière sera visible.

L'utilisateur peut aussi choisir de consulter les plantes qu'il a réservées. Deux listes déroulantes s'offriront à lui : d'une part, la liste des plantes qu'il a réservées mais qui ne sont pas encore prêtes à être récoltées et d'autre part, celles qui le sont. Il peut, à partir de ce menu, décider d'abandonner sa réservation sur une plante en cliquant dessus (par exemple, s'il récolte 3 tomates sur un plant de 6, il peut abandonner sa réservation afin que les autres utilisateurs puissent profiter des fruits restants).

La dernière fonctionnalité est le retrait d'une plante. Lorsqu'une plante ne peut plus être récoltée car tous ses produits ont été enlevé, l'utilisateur est responsable de signaler que la plante a été retirée. Il sélectionne alors le type de plante ne pouvant plus être récoltées et, à partir de là, une liste contenant les emplacements où ce type de plante se trouve s'affichera. Il sélectionne alors l'emplacement vide correspondant.

Les fonctionnalités accessibles au jardinier, en plus de celles déjà accessibles à l'utilisateur lambda, sont les suivantes :

— Traitement des plantes
— Inscription d'un nouveau jardinier

Dans le traitement des plantes, le jardinier peut consulter la liste des plantes possédant un problème de pH afin de pouvoir aller apporter un traitement adéquat à chaque plante. Il peut aussi consulter la liste des plantes supposées matures. En effet, le système se base sur des dates théoriques de maturité, mais il ne peut dire si la plante est ou non mature. Le jardinier va donc venir vérifier si la plante est prête ou non. Si c'est le cas, il peut sélectionner la plante sur l'écran et la passer comme prête. Elle sera alors accessible à la récolte par tous les utilisateurs y ayant un droit.

Le jardinier peut aussi ajouter des collègues sur l'application. Nous fonctionnons sur une base de "chain of trust". La fonctionnalité de jardinier n'apporte aucun avantage sur les fonctionnalités de base de l'application et dès lors, nous pouvons considérer que personne ne sera tenté de posséder ce privilège.

### 4.2.2 Structure

Globalement, l'application est séparée en deux packages : d'une part, les activités correspondant aux fonctionnalités décrites ci-dessus et, d'autre part, les outils que les activités utilisent. Un troisième package est présent et contient le service de Firebase, cependant, ce dernier n'a pas été relié avec le reste du projet par manque de temps.

**activités**

Les activités correspondent toutes à un layout et possèdent le même nom que le layout correspondant. Les fonctionnalités ayant été décrites plus tôt, il est inutile de présenter les activités de façon détaillée.

Il est cependant intéressant de noter que la communication au serveur se fait via une librairie spécifique appelée Volley. Cette dernière marque l'implémentation de la communication http et permet alors de travailler avec des requêtes JSON de façon simplifiée. Dès lors, à chaque endroit où il est nécessaire de communiquer avec le serveur, nous trouverons des JSonRequest ou des JSonArrayRequest (selon que la réponse attendue soit unique ou multiple). Ces requêtes demandent de travailler avec des objets finaux, ce qui n'est pas toujours ce qui est souhaité. C'est pourquoi des objets globaux sont employés dans des méthodes qui seront appelées au sein de la requête, lors du traitement de la réponse.

**outils**

Dans les outils, nous trouvons plusieurs classes outils. La première, appelée Tool, présente des méthodes statiques permettant de convertir les noms de plantes de la base de données à une forme un peu plus amicale pour l'utilisateur ou vice-versa ou encore elle possède une méthode permettant le hachage des mots de passe en MD5 (bien que cette méthode soit dépréciée depuis longtemps maintenant).

Nous trouvons aussi un adapteur spécifique utilisé afin d'instancier le contenu des listes déroulantes à l'aide de liste de String.

Il y a aussi une classe permettant de récupérer le token de Firebase. Cependant, par manque de temps, cette classe n'a pas été complètement implémentée. A terme, avec la classe se trouvant dans le package service, ces deux classes auraient permis la réception de notifications du serveur, afin de notifier l'utilisateur quand une plante est prête à être récoltée ou notifier le jardinier lors de problèmes rencontrés sur la plante (mauvais pH, par exemple).

## 4.3 Raspberry PI et capteurs

Tel qu'expliqué dans les sections précédentes, le Raspberry PI sert de contrôleur aux différents Phidgets, moteurs et lecteurs connectés à la serre. Il est donc élémentaire que ce système précis comprend des classes gérant les Phidgets, les servo-moteurs et autres moteurs, le lecteur de tags RFID, aussi bien qu'une base de données locale et un système d'envoi et réception de requêtes et informations.

### 4.3.1 Phidgets

Il est important de savoir que les différents senseurs, ampoule LED et moteur - autre que le servo - sont branchés via un "Interface Kit", relayant les informations et les connexions. Ainsi, il est possible d'avoir plusieurs senseurs branchés simultanément et de pouvoir les gérer indépendamment.

La classe utilisée pour gérer ces senseurs et capteurs n'est autre que la "Main". Effectivement, c'est à cet endroit que l'"Interface Kit" est initialisé et configuré. De

plus, des "Event Listeners" sont implémentés pour gérer les différents évènements générés par un branchement, débranchement de l'interface ou une lecture des données des senseurs. Etant donné que les senseurs sont tous branchés simultanément et qu'ils ne possèdent pas d'identifiant à proprement parler, le seul moyen de les différencier est l'emplacement de branchement sur l'interface. Ainsi, il convient de ne pas intervertir les capteurs, sous peine de valeurs erronées. Les emplacements sont les suivants :

— luminosité : input 0,
— température : input 1,
— pH : input 2,
— humidité : input 3.

De plus, étant donné qu'il est impossible d'obtenir des valeurs directement captées étant égales à zéro - les valeurs captées acceptant beaucoup de chiffres après la virgule -, des évaluations ont été effectuées. Lorsque les valeurs retournées sont tout-à-fait équivalentes à zéro, un message d'erreur est affiché et une requête signalant que la connexion au senseur concerné a été perdue est envoyé au serveur. Une telle requête est également envoyée lorsque l'évènement de détachement de l'interface a été déclenché.

### 4.3.2 Moteurs

Comme son nom l'indique, la classe "ServoMotors" sert à l'utilisation des servomoteurs. Ces derniers sont en effet initialisés et configurés dans cette classe, grâce notamment à leur connexion via leur propre contrôleur. Une fois encore, des "Event listeners" ont été implémentés, de manière équivalente à celle de "l'Interface Kit". De plus, 3 méthodes ont été implémentée :

— closeWindow()
— openWindow()
— arroser()

Le rôle de chacune d'entre-elles est tout-à-fait compréhensible grâce à leur nom, "Window" référençant la fenêtre d'aération de la serre.

Un autre moteur, différent des servo, a lui aussi été utilisé. Il sert, tel qu'introduit précédemment, à ouvrir et fermer le volet de toit de la serre. Il fonctionne simplement ; il tourne d'un côté ou de l'autre suivant le sens de branchement et tourne toujours tant que le circuit est fermé. L'implémentation de ce moteur est donc simple, il suffit de lui fournir du courant quand cela s'avère nécessaire. Les méthodes relatives à ce moteur sont visibles dans la classe "Main" sous le nom de "closeVolet()" et "openVolet()".

### 4.3.3 Base de données et requêtes

Cette partie constitue un élément primordial de l'implémentation du Raspberry. Effectivement, cette partie va permettre d'envoyer les données perçues par les senseurs ; de les envoyer vers le serveur pour permettre leur traitement et de récupérer une réponse donnant des actions à effectuer ou non. Aussi, cette partie permet de stocker tous les utilisateurs ayant accès à la serre ainsi que de recevoir les listes initiales des emplacements à gérer.

**Requêtes**

Comme évoqué ci-dessus, le Raspberry est capable d'envoyer des requêtes de différent type - GET et POST, seuls ces deux types sont nécessaires - grâce à la classe "Transfers", contenant les méthodes d'envoi, de réception et d'écoute par socket. Lors de la mise en route du Raspberry - de l'allumage initial, donc -, ce dernier demande au serveur, au moyen de deux requêtes GET, la liste complète des utilisateurs - liste des tags RFID - ainsi que celle des emplacements à gérer.

Afin de ne pas devoir complètement écrire les requêtes, la librairie "OkHTTP" a été utilisée. Elle présente en effet l'avantage de fournir une interface simple et pratique dans laquelle le besoin de connaissance et de code est limité. De plus, cette librairie permet de traiter à la fois des requêtes synchrones et asynchrones.

Par la suite, à chaque prise de données via les capteurs, une requête POST est envoyée au serveur. Ce dernier, après avoir mis à jour sa base de données et traité les valeurs retourne, dans le corps de la réponse l'ensemble des actions devant être effectuées. Ainsi, dès la réception de cette réponse, la méthode "onResponse" réimplémentée dans "firePostRequest" va en extraire le corps - sous forme d'objet JSON - et le transmettre directement à la classe "Action", plus précisément à la méthode "getActions". Cette méthode va alors appeler à son tour d'autres méthodes au sein de cette classe donnant les actions à effectuer, comme par exemple arroser, ouvrir la lampe, etc. Ces méthodes vont ainsi directement faire appel aux méthodes agissant sur les moteurs et ampoules - les actions pour le pH n'étant pas vraiment réalisables, du moins pour les moyens accordés.

En ce qui concerne la réception des données par socket, un thread spécialement prévu à l'écoute a été créé. La méthode "socketListener" en est d'ailleurs la preuve. La seule information devant passer par ce moyen étant l'ajout d'un utilisateur dans le système, cette information est donc directement traitée et mise en base de données via la méthode "treatIncomingRequest".

**Base de données**

La quantité de données à stocker n'étant pas très importante et la durée de vie de ces données devant coïncider avec la durée durant laquelle le Raspberry est allumé, la base de données n'est pas externe au programme. En effet, il s'agit d'une base de données H2 créée en mémoire. De ce fait, elle est toujours disponible très rapidement, ce qui est utile pour la vérification des tags RFID.

L'initialisation de cette base de données est la première action effectuée au lancement initial du programme. 2 tables sont ainsi créées. La première, "Users" contient l'ensemble des tags RFID des utilisateurs ainsi que si l'utilisateur est administrateur et s'il est dans la serre. Le seconde, "Terrain", contient l'ensemble des emplacements de la serre ainsi que, pour chacun d'entre eux, les données des capteurs - qui sont nulles lors de la création. Il est donc par la suite possible de vérifier si un utilisateur fait partie de la liste tout en vérifiant s'il est administrateur, ainsi que de mettre à jour les valeurs d'un emplacement - sauf son identifiant, bien entendu.

Il est nécessaire de savoir que toutes les opérations ayant pour but un ajout ou une consultation dans la base de données sont effectuées via la classe "Database".

Dans le but de faciliter les échanges de données entre les classes, des objets "User" et "Terrain" ont été créés, chacun contenant exactement les colonnes présentes dans leur table respective.

### 4.3.4 Lecteur RFID

Le lecteur de tags RFID a été implémenté de manière similaire à l'"Interface Kit", grâce à des "Event Listeners" implémentés dans la classe "AdaptRFID". L'initialisation du lecteur est effectuée lors du démarrage du programme, juste après l'initialisation de "l'Interface Kit". Les "listeners" d'attachement et de détachement fonctionnent de manière identique à ceux expliqués précédemment. Lors d'une lecture d'un tag - "TagGain event" -, le tag est comparé à ceux présents dans la base de données pour ensuite afficher un message d'accueil si une correspondance a été trouvée, ou un message d'interdiction dans le cas contraire.

De plus, l'écriture d'un tag sur une puce a été implémenté et testé, mais mis en commentaire, étant donné qu'il n'est pas nécessaire dans notre cas.

## 4.4 Station météo

Comme évoqué précédemment, la station météo, développée grâce à l'Arduino MKR1000 ne sert qu'à apporter une garantie supplémentaire quant à la disponibilité des données sensorielles nécessaires aux différentes plantes. Cette section va s'organiser de la manière suivante : les capteurs utilisés seront présentés, ainsi que leur branchement et finalement, le traitement puis l'envoi des données seront expliqués.

### 4.4.1 Capteurs

Les données nécessaires étant la température, l'humidité ainsi que la luminosité, les capteurs utilisés sont donc les suivants :

- Capteur de température et d'humidité,
- Photorésistance.

Il est nécessaire de noter que le potentiel hydrogène (pH) du sol est également nécessaire au bon traitement des données d'une plante. Néanmoins, cette station météo étant unique et globale pour la ville entière, il n'est ni possible ni concevable d'effectuer des mesures d'une seule parcelle pour les étendre à tous les terrains. La mesure du pH n'est donc pas ici prise en compte. De plus, si aucun branchement spécial n'est nécessaire pour le capteur de température et d'humidité, une résistance de 10K ohms a été ajoutée au circuit de la photorésistance. Cet ajout a pour but d'augmenter la valeur mesurée par l'Arduino et donc de permettre d'inférer une connaissance supplémentaire : la perte de la photorésistance. Aussi, le fil de mesure de données de cette-dernière doit obligatoirement être branché sur le port A0, sous peine d'une variation accrue des valeurs mesurées, la tension mesurée sur les autres ports étant très variable - la tension a été mesurée grâce à un appareil de mesure spécifique.

### 4.4.2 Traitement des données mesurées

Alors qu'aucun traitement spécifique n'est nécessaire pour la mesure de la température et de l'humidité - le capteur fournissant déjà les valeurs dans les unités requises, respectivement en degré celcius et en pourcentage -, ce n'est pas le cas pour la photorésistance. Effectivement, la mesure envoyée par cette dernière ne possède pas d'unité à proprement parler. Il s'agit d'une valeur résultant d'une mesure de tension et de résistance sur le circuit, variant bien évidemment en fonction de la lumière

perçue - une lumière intense diminue la valeur. Il est donc clair que cette donnée brute ne peut être utilisée efficacement sans être traîtée. La photorésistance doit par conséquent être calibrée par le capteur de luminosité branché sur le Raspberry PI, renvoyant des valeurs en "lux".

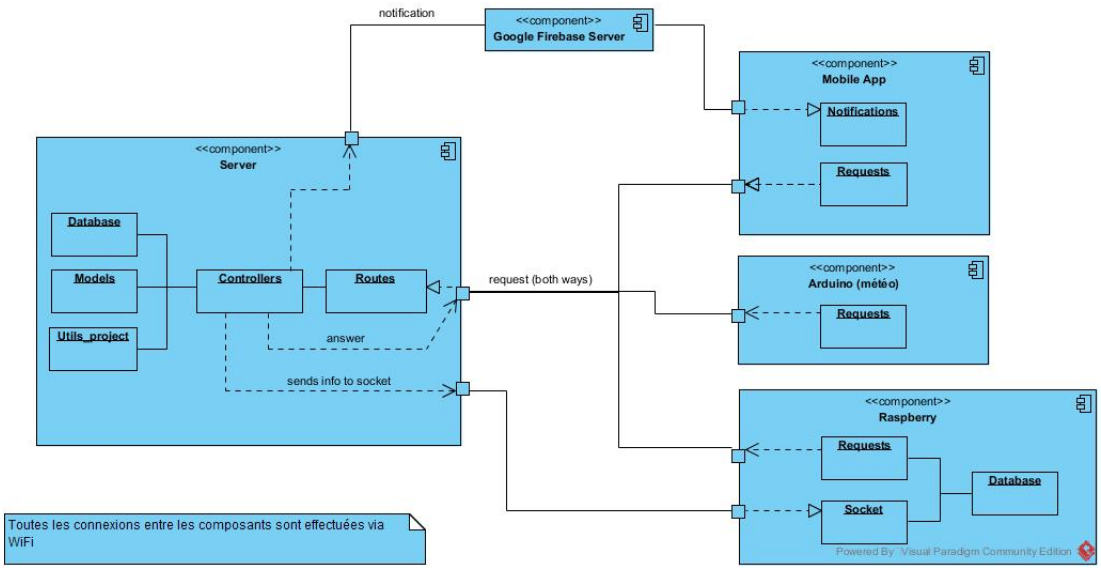
Que ce soit pour récupérer les données de température et d'humidité ou de luminosité, une seule ligne de code Arduino est nécessaire.

### 4.4.3 Envoi des données

Afin de répondre au demandes formulées dans le projet squelette, les données sont envoyées au moyen d'une requête "POST" vers le serveur, via une connexion WiFi telle que détaillée dans la section concernant le serveur. Les données ne sont pas envoyées au format JSON, mais seulement de manière brute, chaque valeur étant séparée par '/' dans le corps de la requête. De plus, lorsque les valeurs sont anormales (-999 pour la température et l'humidité, en-dessous de 100 pour la photorésistance, d'où la nécessité d'une résistance supplémentaire de 10K ohms), une autre requête est envoyée au serveur, signalant la perte des différents capteurs. Dans ce cas, une lampe LED rouge est également allumée. Cette dernière s'éteint automatiquement lorsque des valeurs raisonnables sont à nouveau récoltées.

## 4.5 Schéma global d'interactions

Suite à tous les éléments expliqués dans ce chapitre et les chapitres précédents, voici, un schéma récapitulatif des différentes interactions entre les composants du système global :



# Chapitre 5

# Pistes d'amélioration

## 5.1 Application mobile

L'application, bien que complète, pourrait être améliorée quelque peu afin d'être totalement adoptée par l'utilisateur.

Tout d'abord, comme expliqué dans les chapitres précédents, les notifications à l'utilisateur permettent à ce dernier un confort non négligeable afin qu'il ne se lasse pas de l'application et de l'attente et que celle-ci ne tombe pas dans l'oubli.

Par ailleurs, il est impératif d'instaurer un système de sécurité. Cela commence par une technique de hachage des mots de passe meilleure que le MD5. Nous avons utilisé cette méthode car elle est basique et demande peu de temps pour être implémentée mais nous sommes bien conscients que de meilleurs algorithmes existent, comme nous avons pu le voir au cours de sécurité.

## 5.2 Serveur

Tout comme l'application mobile, la sécurité du serveur pourrait être améliorée. Effectivement, en ce qui concerne la base de données, aucune vérification poussée des données n'a été effectuée.

De plus, tous les accès au serveur, quel que soit le composant émettant les requêtes, sont effectués en HTTP et non en HTTPS. Il s'agit, ici aussi, d'une amélioration sécuritaire possible.

Autrement, il est possible d'essayer de diminuer le temps de réponse du serveur, en mettant par exemple des données fréquemment demandées en cache. Cela éviterait l'accès constant à la base de données.

Finalement, une gestion plus fine des différentes plantes est possible. Pour cela, il conviendrait de réunir des connaissances plus approfondies que celles dont le groupe disposait durant la réalisation du projet.