

Agenda Étudiant Jardinage

Introduction

Le but de ce projet est de réaliser un agenda d'horticulture. Cet agenda doit être composé de trois parties principales : une partie agenda, qui contient les fonctions primaires d'un agenda classique (événements, filtres, navigation dans le temps), une partie qui donne les informations météorologiques et pour finir, une partie qui permet d'enregistrer des plantes et des informations sur celles-ci, ainsi que plusieurs photos, notes, et aussi de créer des événements liés à ces plantes. Notre client est une école d'horticulture, et souhaite fournir à ses étudiants cet agenda, afin qu'ils puissent planifier leurs journées et leurs activités liées à la jardinerie. Ainsi, à l'aide des trois parties qui composent notre application, ces besoins seront traités.

Analyse Globale

Nous allons réaliser ce projet en Java, et plus particulièrement en utilisant la bibliothèque JavaFX. Les trois parties composant le projet seront traitées dans un package qui leur est propre, mais elles ne restent pas pour autant séparées dans le code, car chacune de ces parties sont liées par la même scène, et il est rapidement possible de passer de l'une à l'autre. De plus, il existe des liens entre elles dues aux fonctions à implémenter. Cependant, à l'intérieur de chaque package, il existe différentes fonctionnalités qui seront développées dans la section Plan de Développement qui sont, elles, propres au package à laquelle elles appartiennent à une exception près.

Plan de Développement

Dans cette partie, nous allons énoncer toutes les fonctionnalités qui vont être mises en place dans ce projet. La répartition de ces fonctionnalités est faite en deux catégories distinctes : d'un côté, les fonctionnalités que nous avons réussi à réaliser, et de l'autre celles que nous n'avons pas réussi à réaliser, par manque de temps, ou par difficulté trop importante.

Fonctions implémentées :

- Dans la partie agenda :
 - un système de filtre
 - un système d'événements
 - ajout de filtre
 - ajout d'événement en cliquant sur une cellule
 - sélection d'horaires de début/fin
 - choix de la couleur de filtre
 - afficher ou non certains filtres

- naviguer dans les semaines/mois/années
- lien entre plante et événement
- ajout d'un événement depuis une plante
- Dans la partie plante :
 - l'affichage de la liste des plantes
 - l'ajout d'une nouvelle plante dans la liste
 - la création de fiche plante regroupant les informations détaillées d'une plante
 - l'ajout de photo pour une plante
 - visualisation des images d'une plante lors d'un clique sur l'image
 - l'accès à d'autre fiche plante à partir d'une fiche plante
 - définition des dates de : plantation, rempotage, arrosage, entretien/coupe, récolte
 - l'ajout de mesure
 - une partie pour des notes et observations
 - planifier des événements spécifiques à partir d'une plante
- Dans la partie météo :
 - l'accès à tout un panel d'informations météorologique et climatologiques en temps réel, ainsi que les prévisions pour la journée
 - l'accès à l'heure, en temps réel
 - prévision météorologique et climatologique pour les 10 prochains jours
 - accès à un changement de ville pour voir la météo ailleurs

Fonctions non implémentées :

- Dans la partie agenda :
 - la récurrence des événements (l'option est là mais n'a aucun effet)
 - lors de l'ajout d'un événement via une plante, l'heure de fin d'événement peut être inférieure à celle de départ, nous avons connaissance de ce soucis et aurions pu le régler en ajoutant un prompt pour l'heure de début avant d'afficher l'interface d'événement, mais cela aurait grandement surchargé l'affichage et aurait pu perdre l'utilisateur qui aurait fait face à 3 prompts pour ajouter un événement.
 - accès à une pré-visualisation rapide de la météo du jour
 - modification d'un événement
- Dans la partie plante :
 - la page Graphes des Plantes
 - La fonction de tri pour la liste des Plantes
 - La date pour chaque photo et mesures
 - message d'erreur lorsqu'une plante déjà existante veut être créée par l'utilisateur
 - Redimensionnement des images d'une plante
 - uniformisation des unités de mesure
- Dans la partie météo :
 - le calendrier des saisons
 - l'utilisation d'icônes

Conception Générale

Ce projet est constitué d'une scène dont le contenu change en fonction de l'outil utilisé. Il existe ainsi trois affichages différents : l'agenda, la partie plante, et la météo.

Le package *Agenda* contient les classes nécessaires à l'utilisation de l'agenda :

- Agenda : il s'agit de la classe principale de cette partie, elle permet l'affichage de l'agenda en lui-même, ainsi que des fonctionnalités sur les côtés telles que les filtres, le petit agenda qui sert à naviguer dans le temps. De plus, l'agenda se met aussi à jour en fonction des interactions de l'utilisateur. C'est aussi cette classe qui contient l'ensemble des événements de l'utilisateur ainsi que ses filtres.
- Cell : il s'agit des cellules de l'agenda. Chaque cellule qui n'est pas un jour de la semaine est cliquable afin de pouvoir ajouter un événement sur cette même cellule ainsi que sur les cellules du dessous en fonction de la durée de cet événement. De plus, la couleur du fond de la cellule change pour prendre la couleur du filtre de la cellule si celle-ci en possède un.
- Event : il s'agit de la classe définissant qu'est-ce qu'un événement. Elle contient tous les paramètres et toutes les informations de celui-ci.
- Filter : il s'agit de la classe définissant qu'est-ce qu'un filtre. Elle contient tous les paramètres et toutes les informations de celui-ci.

Le package *PopUp* est un package utilitaire qui permet de gérer les pop-ups à l'écran, qu'il s'agisse aussi bien de l'ajout d'un événement ou d'un filtre, que des pop-up informatifs pour l'utilisateur, notamment pour lui signaler qu'une action a bien été réalisée (heuristique numéro une de Nielsen). Il contient les classes suivantes :

- PopUpPane : il s'agit d'une classe abstraite dont hérite les différents types de pop-ups. Cela permet de réduire ce qui peut être passé en paramètre à la classe qui affiche le pop-up en plus de pouvoir définir certaines méthodes et constructeurs communs à tous les pop-ups.
- PopUp : il s'agit de la classe qui permet l'affichage du pop-up. On y passe en paramètre quel type de pop-up on veut afficher, et celui-ci s'affiche.
- PromptPopUp : il s'agit d'un pop-up purement informatif, utilisé pour informer l'utilisateur de l'état du système.
- EventPopUp : il s'agit d'un pop-up qui gère la création d'un nouvel événement.
- FilterPopUp : il s'agit d'un pop-up qui gère la création d'un nouveau filtre.
- MesurePopUp : il s'agit d'un pop-up qui gère l'ajout d'une mesure pour une plante
- PlantePopUp : il s'agit d'un pop-up qui gère la création d'une nouvelle Fiche Plante
- CityPopUp : il s'agit d'un pop-up qui gère le choix de la ville dans la partie météo.

Le package *Plante* contient les classes nécessaires au fonctionnement de l'herbier :

- FichePlante : La classe contenant les informations d'une plante
- Overlay : La classe qui permet d'afficher en overlay certains éléments
- Plante : La classe qui s'occupe de la gestion de la liste des plantes
- TransparentButton: il s'agit d'une classe qui hérite de la classe ButtonSkin permettant de modifier l'apparence d'un bouton

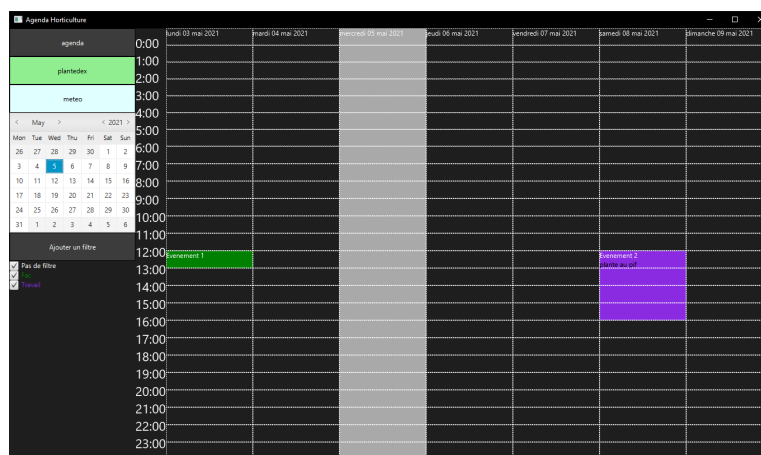
Le package *Weather* contient les deux classes nécessaires au fonctionnement de la partie météorologique de notre projet. Ainsi, elle contient :

- Weather : il s'agit de la classe principale de la météo, qui gère l'affichage ainsi que la récupération des données météorologiques via un API.
- XmlDomParser : il s'agit d'un parser de fichiers DOM XML (Document Object Model) qui est ce que nous récupérons avec l'api. Nous récupérons de ce fichier XML les informations que nous désirons via cette classe.

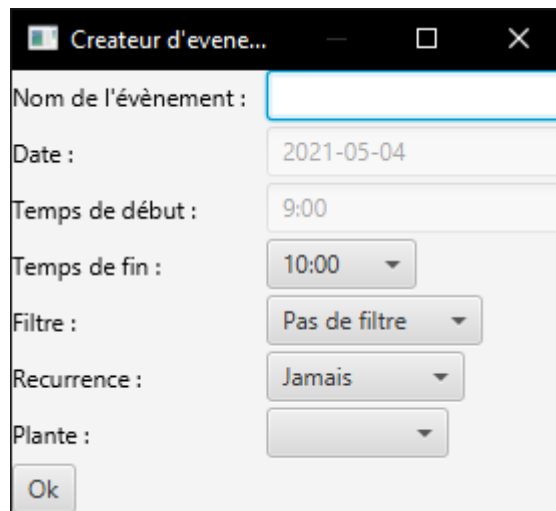
Conception Détaillée

Agenda

La partie agenda est graphiquement simple : il s'agit d'un agenda comme nous avons l'habitude d'en voir habituellement. Il y a, sur le côté, les trois boutons pour naviguer entre les différentes parties du projet, en dessous, un petit agenda qui permet de naviguer dans le temps et qui change la semaine affichée dans l'agenda. Encore en dessous, se situe le bouton pour ajouter les filtres ainsi que les filtres à afficher qui sont sous la forme d'une checkbox. L'agenda est un HBox composé de VBox composés de cellules (class cell). Chaque cellule est cliquable pour ajouter un événement sur celle-ci, et sur les cellules d'en dessous si l'événement ne dure pas une heure. La classe cell extends Region, ce qui lui donne toute les propriétés des nodes, tout en nous offrant une plus grande liberté de customisation.



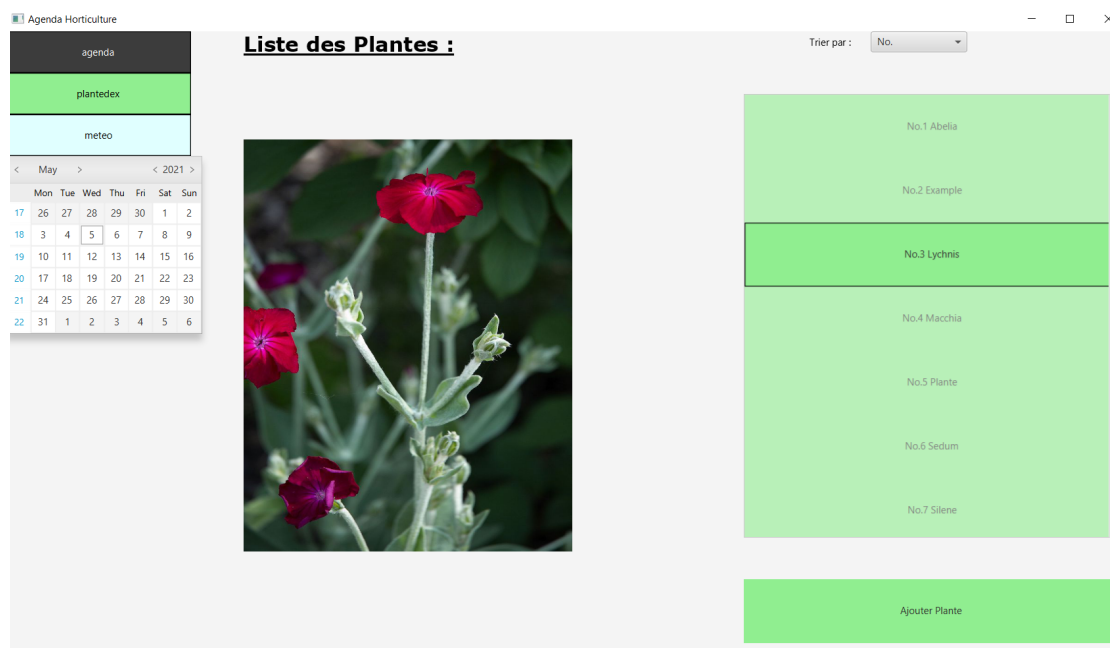
Lorsque l'on ajoute un évènement, une fenêtre pop-up apparaît, gérée par la classe EventPopUp. Ce pop-up contient les informations à remplir pour créer l'évènement. Il y a ainsi : le nom de l'évènement, la date qui est déjà sélectionnée car l'utilisateur a cliqué sur une cellule, donc la date de l'évènement est la date de la cellule, une heure de début qui est celle de la cellule cliquée, un temps de fin, choisi par l'utilisateur, un filtre, avec la possibilité de ne mettre aucun filtre, une récurrence, même si comme précisé plus haut, le bouton est ici mais il n'a aucun effet et pour finir, l'option de lier l'évènement à une plante de la partie herbier.



Quand l'évènement est créé, l'utilisateur est signalé par une fenêtre pop-up indiquant que l'évènement a bien été créé pour respecter l'heuristique de visibilité de l'état du système. L'ajout de filtre fonctionne de manière similaire, il y a seulement comme choix le nom du filtre ainsi que sa couleur, et bien évidemment, un pop-up indiquant que le filtre a bien été créé pour indiquer à l'utilisateur l'état du système. Les événements apparaissent sur l'agenda si leur filtre est coché (donc affiché) en temps que cellule colorée, par la couleur de leur filtre correspondant. Le jour actuel est aussi surligné en gris afin qu'il ressorte, donnant une information visuelle rapide à l'utilisateur pour lui indiquer la journée actuelle.

Plante

Dans la partie Plante, la première page est l'affichage de la liste des plantes. Cet affichage est géré par la classe Plante qui hérite d'un BorderPane pour l'agencement des différents éléments de la fenêtre.



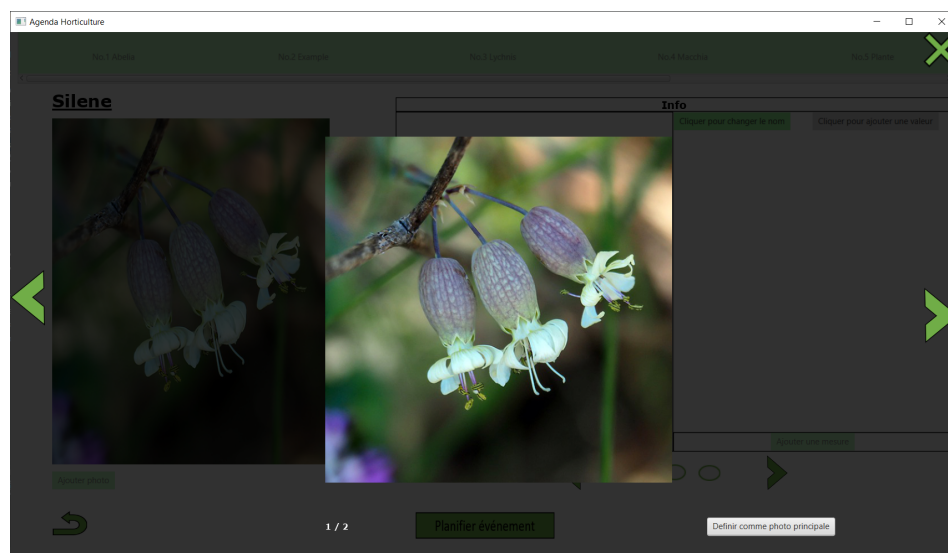
Tout d'abord, à droite se situe la liste des plantes agencées dans une VBox, chaque Bouton représente une FichePlante et emmène sur la page détaillée. En passant la souris sur chaque bouton de la liste, l'image de prévisualisation de la plante change à gauche. cette image est crée par le biais d'une ImageView qui va être mise à jour lorsque la souris passe sur l'un des boutons de la liste en ajoutant un EventHandler à chaque bouton de la liste, la méthode setOnMouseEntered() permet de savoir lorsque la souris passe au dessus d'un bouton ce qui va changer l'image de prévisualisation. Un bouton de tri est affiché au-dessus de la liste mais nous n'avons pas pu le faire fonctionner sur les boutons de la liste.

En dessous de la liste se situe le bouton pour ajouter une plante, en cliquant sur ce bouton une fenêtre pop-up est créée par la classe PlantePopUp et invite l'utilisateur à taper le nom de la nouvelle plante qu'il veut ajouter, malheureusement nous n'avons pas pu faire en sorte qu'une erreur soit créée si l'utilisateur tape le nom d'une plante qui est déjà présente. L'utilisateur écrit dans un TextArea, ce qui permet de récupérer la valeur qu'il tape et de l'assigner en tant que nom de FichePlante. Une fois la plante créée, un message de confirmation est affiché par le biais de la classe PromptPopUp pour respecter l'heuristique de visibilité de l'état du système.

Cliquer sur un des boutons de la liste de Plante permet d'afficher les détails de chaque plante. Une FichePlante hérite de la classe StackPane ce qui permettra plus tard de visualiser des éléments en overlay.

Tout d'abord, dans une fiche, on retrouve l'image de la plante à gauche qui est affichée par le biais d'une ImageView créée dans la méthode previewPlante(). Cette imageView possède un EventHandler, détectant un clic souris, qui permet d'afficher toutes les images de la plante en overlay. Lors d'un clic sur l'image, un objet de la Classe Overlay est crée et est ajouté à la StackPane de la Fiche ce qui permet de faire le fond noir au dessus de la fiche.

Ensuite, une nouvelle BorderPane est créée pour l'agencement des éléments sur l'overlay: au centre on retrouve une imageView, à gauche et à droite deux flèches qui permettent de défiler les photos. Chaque image de la plante possède un indice dans l'Arraylist images, en cliquant donc sur les flèches, l'indice de l'image affichée change ce qui met à jour l'imageView. En haut à droite se situe un bouton fermer, qui va supprimer l'overlay et la borderPane de la StackPane FichePlante.



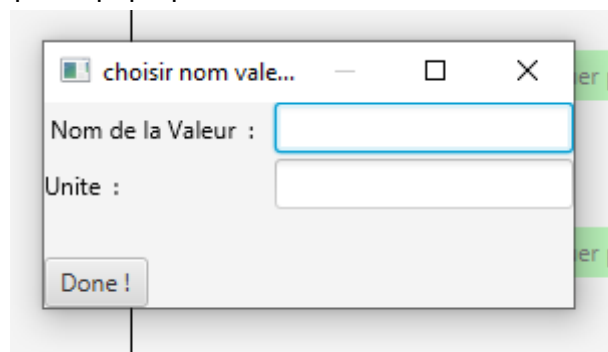
Une exception que nous n'avons pas pu gérer est que l'image n'est pas redimensionnée et peut ne pas être centrée à l'écran lorsqu'elle est trop large. Nous n'avons également pas pu implémenter le fait d'associer une image à une date, cela aurait pu être possible en créant un bouton qui va ouvrir un calendrier pour que l'utilisateur puisse choisir la date de la photo.

On peut retrouver aussi en haut de chaque FichePlante, une ScrollPane contenant les autres FichePlante, ce qui permet facilement d'accéder aux autres fiches par un clique. La fiche Plante où se situe l'utilisateur est de couleur plus foncée pour facilement se repérer. La boîte d'information est créée par le biais de 3 méthodes, chacune gère une page : la méthode infoBox pour la première page avec les mesures et les dates clés, la méthode observation qui crée un TextArea pour les notes de l'utilisateur et la méthode graphes() que nous n'avons pas pu implémenter.

La première page d'information est scindée en deux, la partie des dates clés et la partie des mesures qui sont affichées respectivement par les méthodes dateClef() et mesure(), ces deux méthodes prennent en paramètre une borderPane qui sera ici la BorderPane associée à la page info.

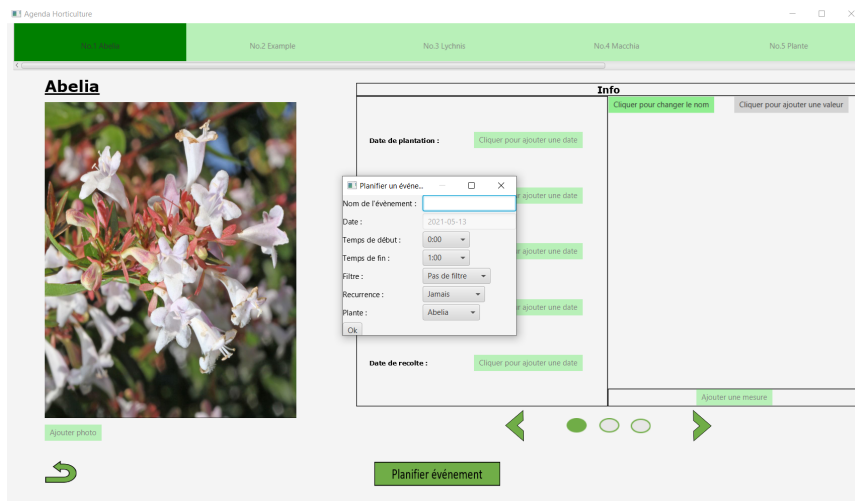
Pour la partie des dates clés, l'utilisateur peut cliquer sur le bouton "cliquer pour ajouter une date", ce qui va appeler la méthode ajoutDate(). Cette méthode crée l'overlay et affiche un calendrier créé à partir de la classe DatePicker. L'utilisateur clique sur une date et un pop-up qui va créer un nouvel événement est affiché avec la classe EventPopUp, le nom de l'événement est pré rempli et la plante associée aussi.

Pour la partie des mesures, l'utilisateur peut ajouter une nouvelle mesure à partir du bouton "ajouter une mesure", l'utilisateur clique ensuite sur le nouveau bouton pour changer le nom et définir l'unité par le pop-up.



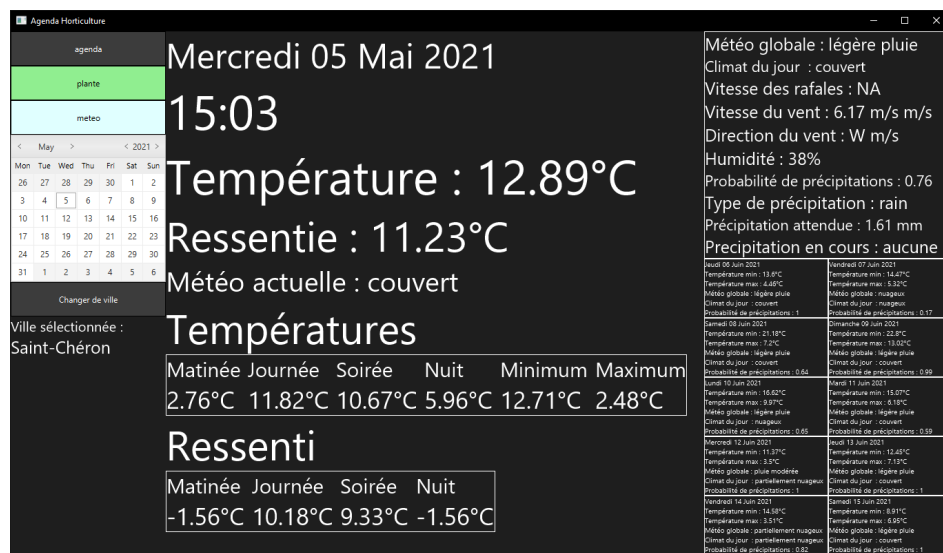
Les boutons associés à une mesure sont arrangés dans une VBox pour pouvoir les empiler les uns sur les autres. Chaque nom de valeur est associé à un bouton "Cliquer pour ajouter une valeur" qui va ouvrir un pop-up créé par la classe MesurePopUp et qui invite l'utilisateur à entrer une valeur. Les noms des valeurs sont modifiables mais nous n'avons pas pu rendre les valeurs tapées modifiables.

En bas de la fenêtre se trouve un bouton "planifier un événement", ce qui va permettre de planifier un événement à partir de la page de la plante. Un calendrier est affiché et invite l'utilisateur à cliquer sur une date, une fois la date cliquée, le jour est récupéré et un nouvel objet de la classe EventPopUp est créé avec la plante associée déjà rempli et la date récupérée selon la sélection de l'utilisateur. Un nouvel événement sera ainsi donc créé dans l'agenda associé à la plante.



Météo

Pour finir, la partie météo de notre projet comporte une multitude d'informations météorologiques et climatiques. Nous aurions voulu ajouter des icônes pour en faciliter la lecture, et moins submerger l'utilisateur d'informations pour respecter l'heuristique de design minimaliste mais nous n'avons pas eu le temps.



Nous avons quand même fait en sorte que les informations principales soient centrées, avec une taille de police plus importante afin d'attirer l'œil. Il est ainsi facile de lire la température du jour, l'heure, la date, la météo actuelle ainsi que la température ressentie. Il y a cependant plein d'autres informations retrouvées à deux endroits : deux tableaux en dessous, qui informent sur les températures de la journée ainsi que les températures ressenties de la journée et enfin le panneau de droite. Ce panneau regorge de plein d'informations : la météo globale de la journée ainsi que son climat, la vitesse des rafales de vent actuelles (si il y en a), la vitesse du vent ainsi que sa direction, le taux d'humidité, la probabilité de précipitation dans la journée, le type de précipitation attendue dans la journée

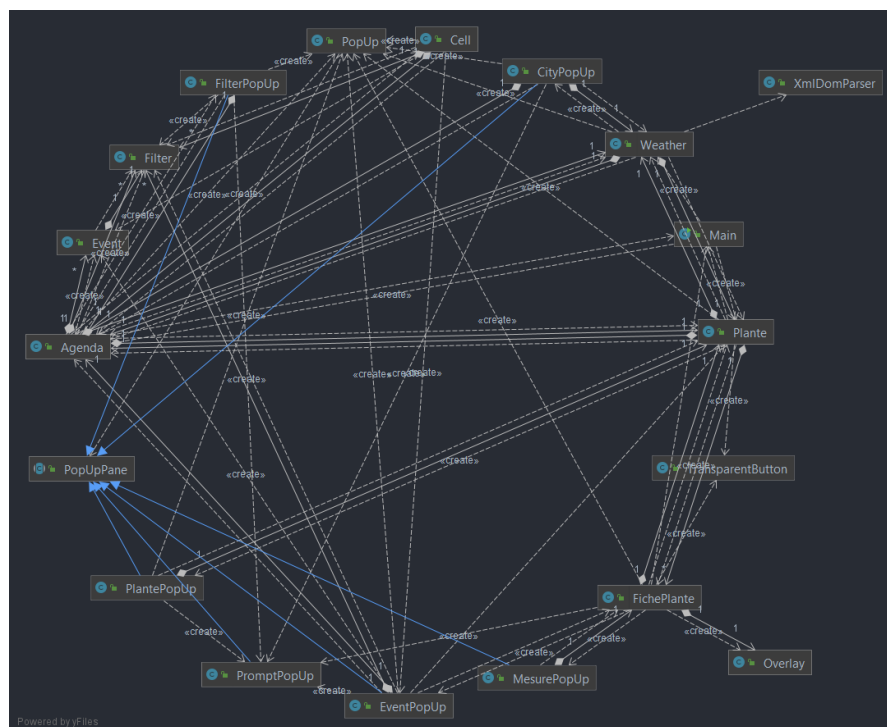
(neige ou pluie), le volume de précipitation attendu dans la journée et pour finir, les précipitations en cours.

En dessous de ce panel informatif, se trouvent les informations essentielles concernant les 10 prochains jours. Cette partie est assez importante car pour une école d'horticulture, les informations météorologiques et climatiques sont importantes pour s'occuper des plantes, c'est pour cela que nous avons préféré rajouter quelques informations supplémentaires au risque de submerger un peu l'utilisateur d'informations. Notre projet n'est pas grand public, mais spécialisé pour notre client, l'école d'horticulture, ainsi nous pouvons nous permettre de donner plus d'informations pour que nos clients puissent mieux prévoir la météo pour leurs plantes. Sur le côté gauche, nous retrouvons les trois boutons et le petit agenda habituel, en plus d'un bouton permettant de changer la ville dont les données météorologiques sont prises, ainsi que l'affichage de la ville actuellement sélectionnée.

Toutes les données météorologiques sont acquises via l'api [openweathermap](https://openweathermap.org/). Nous faisons une requête internet et stockons le résultat sous la forme d'un fichier xml. Cette requête est effectuée à l'aide de la classe HttpURLConnection, le fichier est construit à l'aide d'un FileWriter qui écrit le résultat de la requête. Il y a deux requêtes faites à chaque fois : une requête pour avoir la météo actuelle et une autre pour avoir les prévisions pour les 10 prochains jours. Ces deux fichiers sont alors parsés à l'aide de la classe XmlDomParser qui parse le fichier xml via les classes DocumentBuilderFactory, Document et DocumentBuilder, puis il y a une itération sur l'ensemble des nodes du fichier, en récupérant les informations des nodes que ne souhaitons. Ces données sont alors injectées dans la classe weather sous la forme d'une hashmap String, String. Nous avons alors les données avec les keys que nous avons défini, et la valeur correspondante sous la forme de String. Il est ensuite simple de juste faire un affichage en récupérant les données à tout moment dans la hashmap.

Notre projet peut être résumé par ce diagramme de classe :

Diagramme des différentes classes de notre projet :



Il est à noter que nous avons choisi délibérément de ne pas afficher les méthodes et les paramètres de chaque classe afin d'en améliorer la lisibilité.

Documentation Utilisateur

Pour pouvoir utiliser notre logiciel, il y a quelques petites choses à savoir. Tout d'abord, un IDE Java est nécessaire (ou Java seul si vous avez fait un export .jar exécutable), avec la version 11 de la JDK de Java ainsi que la version 11 de la SDK de JavaFX. Si vous êtes dans le premier cas et que vous avez un IDE Java, il vous faut importer le projet dans votre IDE, sélectionner la classe Main, ajouter les options de VM et rajouter `--module-path [path to JavaFX] --add-modules=javafx.controls,javafx.fxml`, en prenant soin de remplacer [path to JavaFX] par le path au dossier lib de la bibliothèque JavaFX sur votre ordinateur. Ensuite, il vous faut aller dans la structure du projet, et ajouter dans la catégorie "bibliothèques" la bibliothèque JavaFX. Une fois cela fait, il ne vous reste plus qu'à lancer l'application en cliquant sur Run as Java Application. Si vous êtes dans le second cas et possédez une version .jar, il vous suffira de cliquer sur le fichier exécutable. En haut à gauche, peu importe où vous vous situez dans le logiciel, il y aura toujours les 3 boutons permettant de naviguer au travers des trois parties du projet.

Conclusion et Perspectives

Nous avons réussi à implémenter de nombreuses fonctionnalités donnant au logiciel une utilisation basique possible, ainsi que quelques options supplémentaires. Il reste tout de même une bonne quantité de travail si nous voulons rendre ce logiciel entièrement utilisable par notre client. Cependant, nous avons rencontré quelques difficultés telles que la manipulation de certains types d'objets assez complexe, l'agencement des différentes Pane dans une fenêtre, ainsi que la création à partir de zéro d'un agenda. Ce projet nous a permis d'apprendre à utiliser JavaFX ainsi que tout un panel de type d'objets différents, ainsi qu'une nouvelle manière de programmer les interfaces. Il nous a aussi permis de nous mettre face à la réflexion pour la création de la meilleure interface possible pour nos futurs utilisateurs. Ainsi, nous en ressortons une expérience non négligeable qui nous aidera sans l'ombre d'un doute dans notre avenir. Il reste, cela dit, encore une certaine quantité de fonctionnalités à implémenter dans ce projet telles que l'ajout de graphique dans les fiches plantes, ainsi que la possibilité de modifier les valeurs des mesures rentrées. Les boutons latéraux pour accéder aux différentes parties pourraient aussi être plus soignés, par exemple en affichant simplement un logo pour réduire l'effort mnésique de l'utilisateur et aussi respecter l'heuristique de standardisation. De plus, plus de pop up de confirmation ou d'erreur devraient s'afficher pour respecter l'heuristique de visibilité de l'état du système.