RAPPORT
CV Search

FROISSART Kévin – TOUHARDJI Hamza | MIF01 | 2021 - 2022

Sommaire

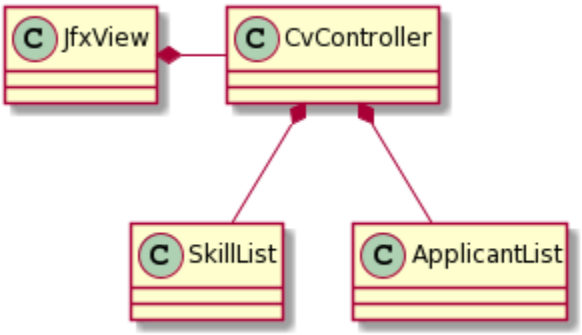
Table des matières

Présentation du projet

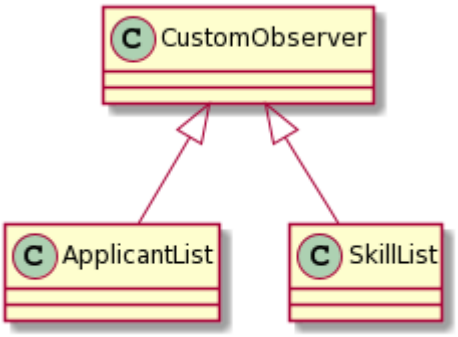
CV Search est un Applicant Tracking System, une application permettant de digitaliser le processus de recrutement en sélectionnant des CV en fonction de certains critères. Parmi ces critères, nous avons la possibilité de trier les candidats en fonction de leur niveau de maîtrise ainsi que de leur nombre d'années d'expérience. L'application nous renvoie alors une liste de potentiels candidats, triée du plus compétent au moins compétent.

Design Patterns

Pour ce projet, nous avons implémenté quatre patrons de conception. Le premier est le **Modèle-Vue-Contrôleur**. Ce patron consiste à diviser le fonctionnement d'une application en trois groupes d'entités distinctes. Les composants front JavaFx forment la ou les Vues. Les classes d'objets permettant le bon fonctionnement de l'application et les traitements métier forment les Modèles. Finalement, les Contrôleurs permettent la communication entre les vues et les modèles. Cette architecture permet d'organiser le code, de le structurer et offre une meilleure réutilisabilité ainsi qu'une meilleure maintenabilité.

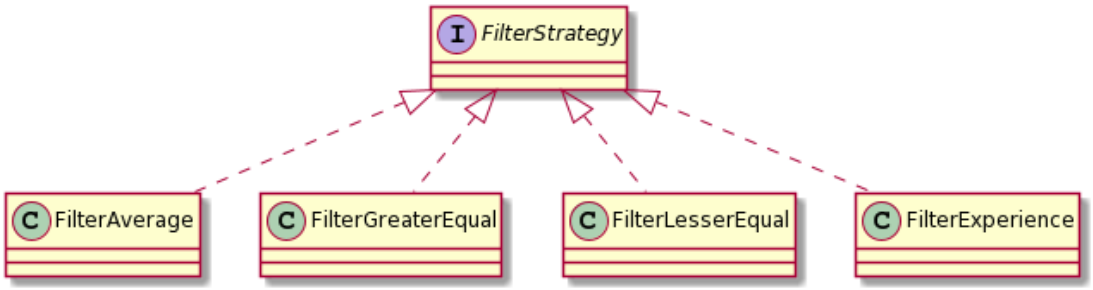


Nous avons ensuite implémenté le patron **Observateur** qui vient se greffer à notre implémentation du MVC. Celui-ci permet en effet de mettre en place un mécanisme de souscription de manière à notifier les vues lorsque les modèles observés par celles-ci subissent des modifications. Pour favoriser la réutilisabilité du code, nous avons implémenté une classe template nommée CustomObservable qui utilise un PropertyChangeListener. De cette manière, les objets qui nécessitent d'être observée par la vue peuvent hériter de cette classe et envoyer des notifications lorsque des changements sont effectués.

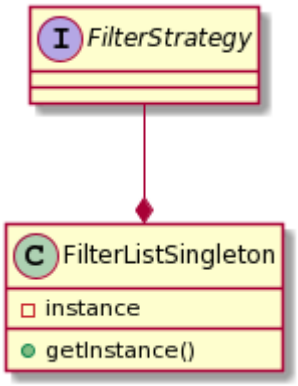


Le troisième patron de conception que nous avons mis en place est le patron **Stratégie**. Ce patron de conception permet de définir une famille d'algorithme interchangeable. Les différentes stratégies de sélection de

candidats sur leur niveau de compétence et sur leurs expériences professionnelles sont modélisées par des classes polymorphes qui implémentent toute la même interface.



Le dernier patron de conception est le patron **Singleton**. Celui-ci permet de garantir l'unicité de l'instance d'une classe au sein de toute l'application. Nous avons décidé d'implémenter ce patron de conception dans l'utilisation de nos stratégies de manière à rendre accessibles toutes les stratégies instanciées aux différentes vues de l'application.



Ces UML ont été généré via l'outil Plantuml. Un UML du projet complet se trouve à la racine du projet sous le nom cv-search_UML.png et a été généré via l'outil de génération automatique d'UML d'Intellij.

Éthique

Lors du développement de l'application, nous avons décidé de respecter deux points essentiels :

Dans un premier temps, notre application sélectionne des candidats de manière objective en prenant en considération seulement des critères dits professionnels. Le tri s'appuie sur la maîtrise du langage de programmation et sur le nombre d'années d'expérience du candidat.

Dans un deuxième temps, l'application affiche seulement le nom et la note moyenne du candidat. Nous avons fait le choix d'afficher aucune autre information (l'âge, le sexe, l'origine ou le lieu d'habitation) pour ne pas influencer le choix des RH. En effet, d'après l'étude ÉCONOMIE ET STATISTIQUE N° 464-465-466 postée par l'INSEE en 2013, un candidat d'origine étrangère a moins de chances de trouver un emploi qu'une autre personne à compétences égales. Le problème s'étend aussi au niveau de l'âge lorsque les personnes s'approchent de la retraite ou chez les femmes mettant en cause le congé maternité ou plus globalement leur genre.

De ce fait, l'ensemble des candidats sont sélectionnés de la même manière, aucun jugement n'est fait. L'algorithme ne pourra pas être accusé d'être discriminant.

Cependant, cet outil vient seulement aider le travail des RH. Il ne prend pas en compte les compétences humaines, mais les compétences utiles pour bâtir une équipe. Ainsi, un ou plusieurs entretiens devront être fait pour trouver le développeur parfait.

En effet, l'algorithme peut être mis en défaut dans un cas où par exemple, un candidat posséderait d'excellentes notes sur 4 des 5 technologies triées dans l'une de nos stratégies, mais la 5^{e} se trouve un tout petit peu en dessous du minimum requis. L'expérience du candidat sur les 4 premières technologies aurait largement compensé ce qui manquait sur la 5^{e} et l'algorithme nous aurait fait rater une pépite !

Les tests

Nous avons choisi de tester plusieurs fonctionnalités de notre code. Chacun des tests s'organisent en trois parties. La partie *given*, dans laquelle on instancie un objet. La partie *when*, où l'on appelle une fonction de l'objet et enfin la partie *then* où l'on teste la fonction en question.

Dans un premier temps, nous avons testé si notre application est capable de lire les données depuis des fichiers « yaml » et de créer des candidats à partir de ceux-ci. Pour créer un candidat, son nom doit être renseigné ainsi qu'une ou plusieurs compétences et une ou plusieurs expériences.

Partie de cette base, nous avons testé si les différentes stratégies renvoyaient les bons candidats en fonction des compétences sélectionnées.