DULHOSTE Antoine
VIALATOUX Lucas
Master informatique – 1ère année
Septembre 2019

Rapport et modélisation

Mission : Gestionnaire de CV

Université Claude Bernard
Lyon 1
Département informatique

43 boulevard du 11 novembre 1918, 69622 Vileurbanne 04.72.69.21.82

Présentation du projet

Ce projet nommé « cv_search » doit permettre de trier des candidats en appliquant certains filtres, tel que les compétences et/ou expériences recherchées. Il doit également être possible d'affiner les résultats avec différentes stratégies de recherche.
Ce projet s'inscrit dans un apprentissage plus large du langage Java, pour un développement « propre ». C'est-à-dire que nous utilisons différents outils pour écrire du code propre, en faisant des tests, en respectant des codings style, en travaillant en intégration continue ou encore en travaillant à l'aide d'un gestionnaire de version.
Les outils utilisés pour cela sont entre autres : Maven, git, gitLab, ainsi qu'un IDE que nous avons choisi, Netbeans.

Design patterns

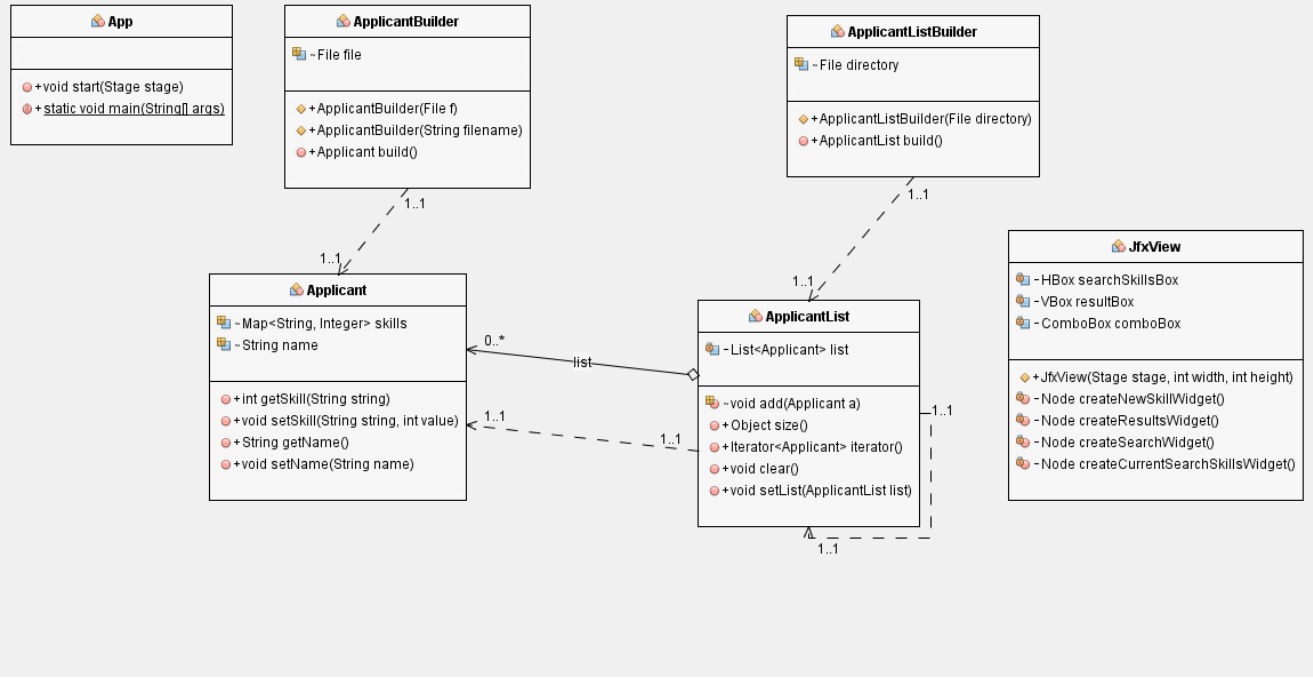


Diagramme de classe avant application du modèle MVC, des design patterns et de GRASP

Modifications apportées (MVC / GRASP / Design patterns)

Nous avons décidé de respecter le modèle MVC et d'appliquer certains patterns GRASP qui nous semblaient cohérent avec la gestion du projet.

Modèle MVC

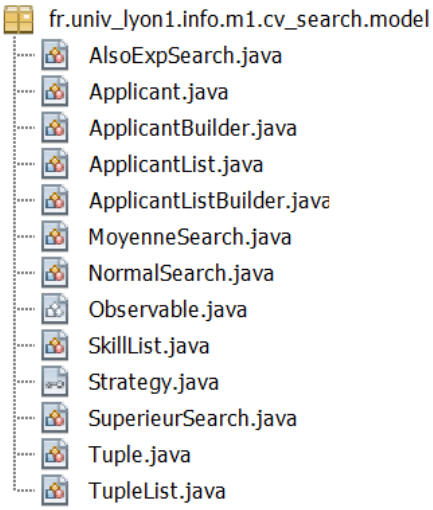
Pour la mise en place du **modèle MVC**, nous avons créé un package « controller » avant toute modification, pour ensuite pouvoir créer une classe **WidgetControler**, qui prendra en

charge la relation entre vue et modèle. Pour ce faire, nous avons également créé plusieurs modèles, par exemple **Tuple** et **TupleList** qui servent à avoir une liste des candidats triée.

Deux fonctions (**createNewSkillWidget** et **createSearchWidget**) géraient des comportements dans la vue, nous les avons donc implémentés dans le « controller » **WidgetController** et dans les modèles appropriés.

Ci-contre la liste des modèles après passage en modèle MVC. Les modèles liés à la gestion des candidats (« Applicants »), sont encore présent, mais ont été légèrement modifiés. Toutes les stratégies de recherche sont également dans le package model.

La finalité de ces changements est de permettre à la vue **JfxView** de gérer uniquement l'affichage, et non pas de gérer également la logique (présente dans les modèles et les contrôleurs).



Liste des modèles du projet

Le modèle MVC mis en place permet à la vue de communiquer avec le controller, qui peut à son tour communiquer avec le(s) modèle(s). Le modèle peut ensuite indiquer à toutes les instances de vues des changements qui ont eu lieu au sein de lui-même.

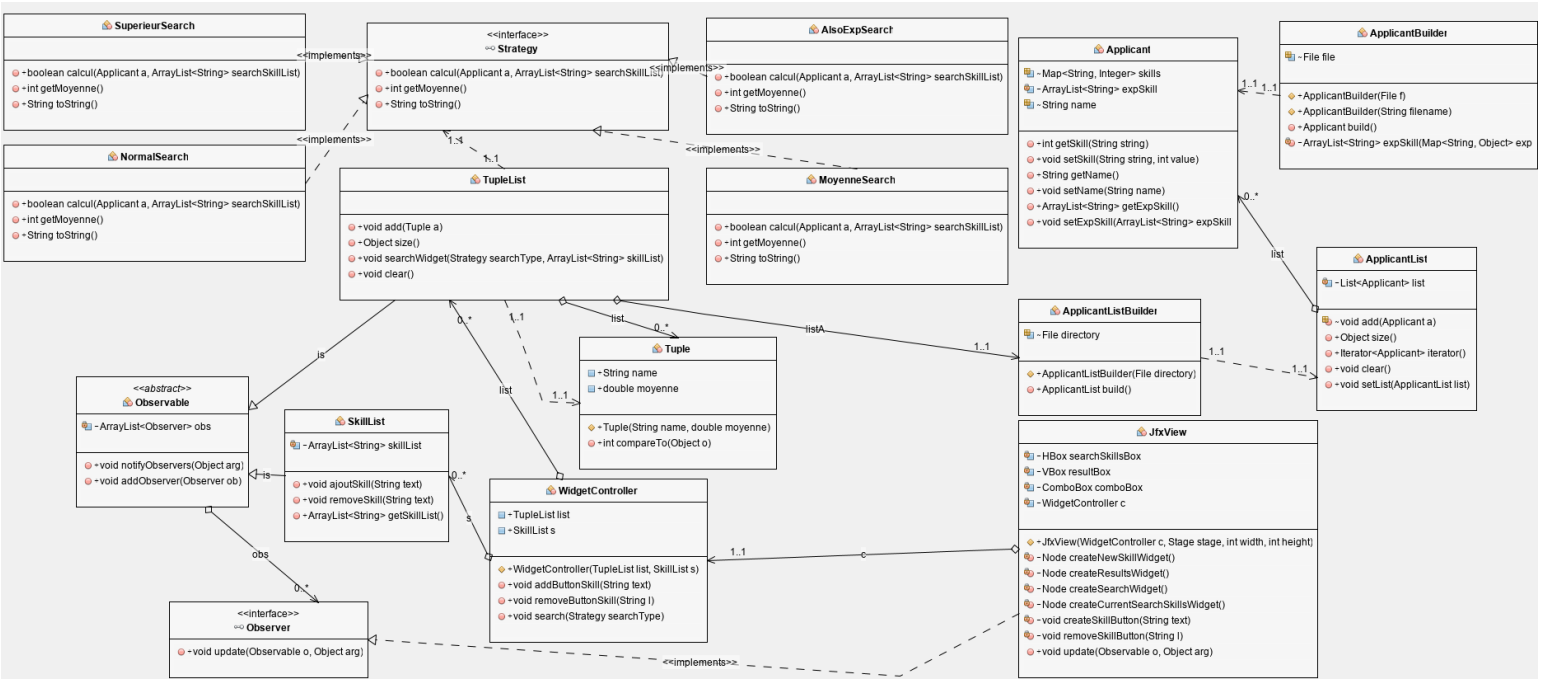


Diagramme de classe après application du modèle MVC, des design pattern et de GRASP

Patterns GRASP

Les modifications citées ci-dessus, nous ont servi à respecter le principe **Expert**, donc à respecter les principes d'encapsulation et de collaboration (POO). La création du système en orienté objet implique un **faible couplage** et une **forte cohésion**, les **classes** mises en place sont ainsi très **génériques** et donc **réutilisables**.

Plus précisément, nous avons réussi à mettre en place une **cohésion fonctionnelle**, c'est-à-dire que chaque classe peut être décrite par une seule phrase.
Par exemple : « SkillList permet la création de liste de compétences » ou encore « NormalSearch permet la recherche supérieure strict à 50 sur les candidats ».

Design patterns

Strategy Design Pattern:
Ce design pattern a été mis en place avec l'interface « **Strategy** » qui est implémentée par les stratégies de recherche (NormalSearch, SuperieurSearch, MoyenneSearch, AlsoExpSearch). Le code utilisant les stratégies n'aura pas besoin d'être modifié suite à la suppression ou l'ajout d'une nouvelle stratégie.

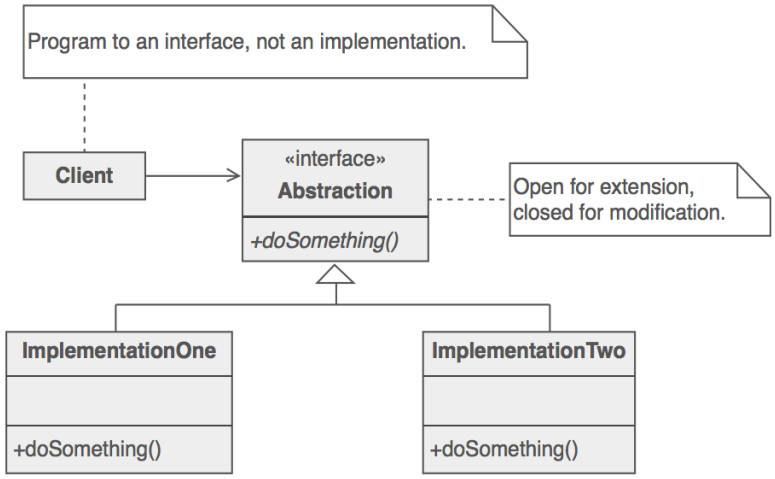


Diagramme de classe illustrant le Strategy Design Pattern

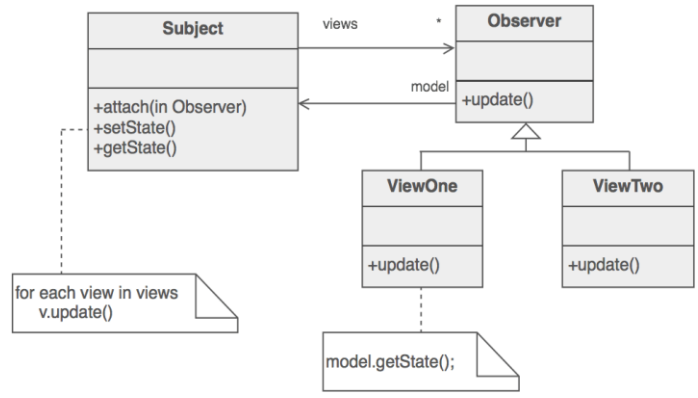


Diagramme de classe illustrant l'Observer Design Pattern

Observer Design Pattern:
Le modèle implémente la classe **Observable** qui va lors de la modification de celui-ci va envoyer les modifications à toutes les instances de vues. La vue JavaFX mise en place, « extend » le modèle Observer, ce qui permet que deux instances de la même vue soient synchrones lorsque l'utilisateur intervient dans l'une d'elle.

Ethique

Stratégies de sélections basées sur les compétences

Nous avons mis en place 4 stratégies de sélections différentes. Les 3 premières sont celles décrites tout au long du projet, c'est-à-dire la stratégie de recherche « **NormalSearch** », qui présente les candidats ayant les compétences (« skills ») sélectionnées **supérieure strictement à 50**.
S'en suit la stratégie de recherche qui fonctionne de la même manière mais qui sélectionne les candidats ayant les compétences sélectionnées **supérieures ou égales à 60**, cette stratégie est nommée « **SuperieurSearch** ».
Toujours basé uniquement sur les compétences, nous retrouvons la stratégie de recherche « **MoyenneSearch** », qui fait une moyenne des compétences sélectionnées par l'utilisateur, et n'affiche que les candidats ayant une **moyenne supérieure ou égale à 50**.

Stratégie de sélection basée également sur les expériences

Ayant eu uniquement des stratégies de recherches basées sur les compétences, nous avons décidé d'implémenter une dernière stratégie alliant compétences et expériences :
« **AlsoExpSearch** ». Cette stratégie permet de chercher des candidats ayant à la fois les **compétences** sélectionnées **supérieures ou égales à 50** mais surtout ayant les langages informatiques également dans les **expériences**. Ainsi ne seront pas que sélectionnés des candidats ayant les compétences recherchées, mais également de l'expérience dans les langages informatiques.

Les stratégies sont-elles suffisantes ?

Bien que les 4 stratégies implémentées permettent de rechercher des candidats ayant soit les compétences ou bien les compétences et l'expérience requises. L'application ne permet pas à l'utilisateur de choisir la limite à laquelle on recherche les candidats. Ainsi si un candidat présente une compétence de 49, il ne sera pas affiché dans les résultats des stratégies suivantes : NormalSearch et SuperieurSearch.

Bien que la recherche par moyenne puisse essayer de venir pallier ce manque, il n'est pas possible de faire de recherche de candidats ayant uniquement les langages informatiques dans les expériences. Il faudrait donc implémenter une recherche qui se base uniquement sur les expériences des candidats.

L'outil de sélection de CV, indispensable mais imparfait

Dans le monde réel, nombre d'exemple de grandes entreprises recevant des milliers de CV par semaine sont disponible (ex : GAFAM = Google / Apple / Facebook / Amazon / Microsoft). Il est donc compréhensible qu'un outil de sélection de CV permette de trier les CV des candidats, tant ce nombre est élevé. Mais il ne doit pas remplacer les êtres humains, qui auront un œil critique sur les CV, qui prendront notamment en compte le poste recherché, les compétences dites cruciales et celles qui sont « un plus ».

Un tel outil se doit de présenter un éventail plus large de CV retenu, donc de ne pas avoir de recherche trop stricte, pour que l'être humain puisse ensuite faire une sélection plus poussée.

Peut-être que l'intelligence artificielle viendra épauler les grandes entreprises et les directions des ressources humaines, pour trouver le candidat idéal pour un poste disponible.

Tests

Tests manuels

Tout au long du développement de l'application, nous avons réalisé différents tests manuels. Ces tests permettaient notamment de tester le bon fonctionnement des boutons et des résultats affichés. Pour ce faire, nous nous mettions à la place d'un recruteur, qui entre les langages informatiques pour lesquels il recherche un candidat, puis nous appliquions différentes stratégies de recherche.

Nous avons également, lorsque nous ne comprenions pas d'où venait une erreur, utilisé la console en écrivant à différents endroits des « System.out.println() ». Ces tests, bien que sommaires, permettent de trouver une erreur lorsque nous étions en train de développer.

Pour tester les limites que nous fixions dans les différentes stratégies, nous avons également modifié les valeurs des applicant1.yaml et applicant2.yaml pour voir si les candidats ressortaient bien dans tel ou tel stratégie. Mais modifié ces valeurs nous permettaient également d'effectuer des tests pour savoir si nous cherchions bien toutes les compétences et les expériences des CV.

Sources - Sitographie

Maven (logo)
https://medium.com/nycdev/java-get-started-with-apache-maven-a71f4f907cb3
Git (logo)
https://fr.wikipedia.org/wiki/Fichier:Git-logo.svg
GitLab (logo)
http://www.telecom-valley.fr/11-decembre-2018-tech-workshop-gitlab/
Strategy Design Pattern
https://sourcemaking.com/design_patterns/strategy?fbclid=IwAR0Tgy80HcPnODShswkjmlyOQfO3RzDPeqicx06QmfcKUffSgU_D8UGjtaI
Observer Design Pattern
https://sourcemaking.com/design_patterns/observer?fbclid=IwAR2cYgg11XNw6ZT1v4yJVeWi6EqtevKBgFlEMxV5TB5EvCTCyyjaASXQ6eU
Netbeans (logo)
https://cwiki.apache.org/confluence/display/NETBEANS/NetBeans+Logo