CY TECH

2023-2024

PROJET ORIENTATION
RAPPORT DE PROJET

ING1 GMF2

Naïm Zoghlami

Oscar Mesnildrey
Léo Renault
Noa Ducaroix--Lasseur
Mathieu Simond
Arthur Durand

Table des matières :

I - Introduction

Après avoir dans un premier temps, élaboré une solution algorithmique au problème d'orientation, en effectuant une recherche bibliographique de ce qui se faisait dans la littérature, et élaboré le cahier des charges, notamment les fonctionnalités que nous souhaitions intégrer à notre application et les éventuelles solutions des futurs problèmes que nous allions rencontrer. Nous nous sommes attelés au développement de notre solution au problème posé dans le premier rapport.

Ce rapport a pour objectif de détailler le développement de notre application dans sa globalité. Ainsi que les méthodes de travail en équipe et les stratégies de développement mise en place pour réussir sereinement le projet. Enfin il apportera un regard critique sur notre travail grâce à une étude de performance de notre application.

Le développement de l'application se décompose en 3 grandes parties :

1. Création de la maquette avec les différentes vues, ainsi que du diagramme UML.
2. Développement du "cœur" de l'application i.e. des classes, des méthodes, de l'algorithme d'affectation expliqué dans le rapport 1et des classes de tests.
3. Développement des interfaces graphiques, i.e. de la solution finale.

II – Méthodologie de travail

Nous avons mis en place la méthode Agile au sein de l'équipe. Une méthode largement utilisée dans le monde professionnel notamment dans le développement logiciel. La méthode Agile est une approche de gestion de projet flexible et collaborative, divisée en sprints courts pour des livraisons fréquentes. Avec utilisation d'un backlog, une liste priorisée de tâches, qui est constamment révisé pour s'adapter aux besoins changeants. Agile favorise l'interaction avec les parties prenantes et l'amélioration continue. En effet, nous tenions un backlog à jour où était spécifié les fonctionnalités principales de notre application mais aussi celles "bonus" qui amélioreraient l'UX (user experience).

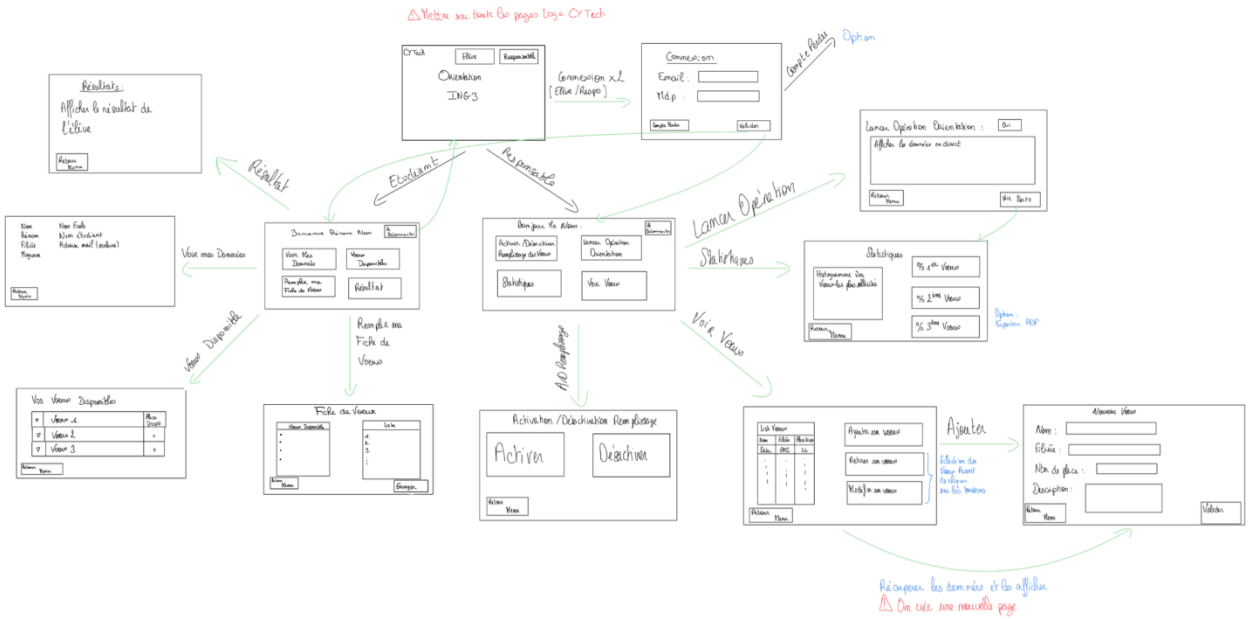
Pour la méthodologie de développement, nous avons fait de la rétro-ingénierie, en imaginant la solution finale nous avons pu prévoir (en partie) les points de difficultés auxquels nous allions nous confronter. Par exemple, pour la fonctionnalité "Activer/Désactiver l'option remplissage des vœux" nous avons pu imaginer la solution (fichier json avec un booléen) avant même de se heurter au problème lors du développement, permettant un gain de temps considérable en évitant des réunions pour un seul problème. D'autres part, les classes de tests de l'application ont été développé avant la création des classes dans l'optique une nouvelle fois de gagner du temps.

III – Maquette de l'application

La mise en place de la méthodologie de rétro-ingénierie nous a conduit à d'abord penser la solution finale, c'est à dire l'application graphique. Nous avons donc commencé par dessiner la maquette de cette dernière pour nous orienter tout le long du développement. L'enjeu ici était d'avoir un maximum de flexibilité dans cette maquette car la moindre erreur aurait impacté lourdement la flexibilité de notre solution. En effet, cette maquette a été la base de tout notre travail. C'est pourquoi nous nous sommes efforcés d'imaginer de multiple cas d'utilisation lors de l'élaboration de cette maquette.

Quelques exemples des questions que nous nous sommes posées :

- Et si une filière est amenée à être modifiée ? Supprimée ?
- Et si c'est une autre école qui utilise notre application ?
- Et si plusieurs promo (MI, MF...) ont accès à des mêmes filières ?
  Exemple : Si l'IA est accessible par la voie MI et MF. Comment affecter les élèves ? Nous les classons ensemble ? Séparément ? Pour avoir une solution générique nous avons décidé de laisser le choix à l'admin.



Maquette de notre application (maquette disponible sur le GitHub : MaquetteApp.pdf)

IV – Implémentation

A – Choix du Langage

Pour le choix du langage de programmation de notre application, nous avons longuement hésité entre Java et Python. Après une analyse approfondie des avantages et des inconvénients de chaque langage, nous avons finalement opté pour Java.

En effet, bien que Python présente des avantages indéniables en termes de simplicité et de rapidité de développement. Grâce à son typage statique et à sa gestion stricte des types, Java offre une plus grande robustesse et une meilleure performance surtout pour du développement logiciel. Java était donc selon nous le meilleur compromis entre du Python et du C. Nous sommes convaincus que ce choix permet de réaliser une application fiable et accessible, répondant pleinement aux besoins des étudiants et des enseignants.

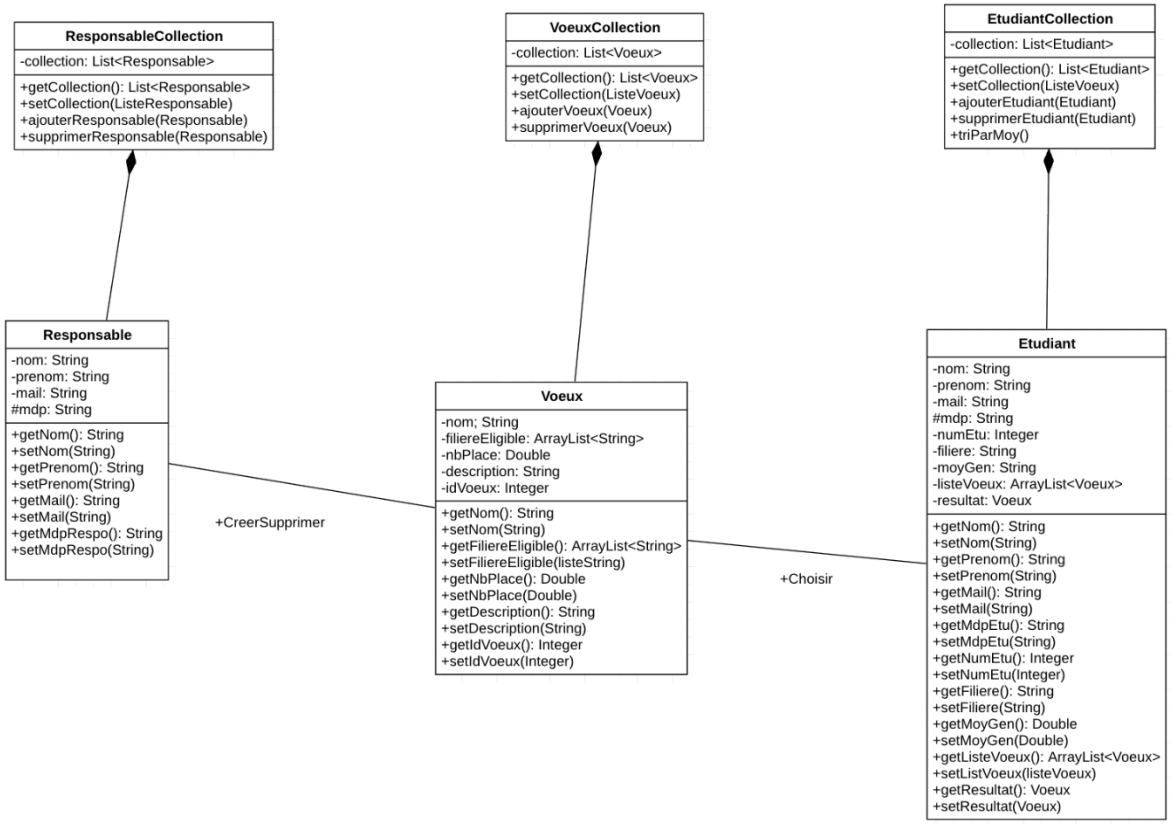
Pour la construction de notre jeu de donnée, nous avons utilisé d'une part de l'intelligence artificielle pour générer des données notamment des étudiants avec leur différents attributs et leur liste de vœux. Mais cette option étant assez limités en termes de quantité nous avons par la suite utilisés Python qui est très pertinent pour de la manipulation de données.

Enfin, pour la création des graphiques montrant la performance de notre application nous avons utilisé une nouvelle fois Python et sa bibliothèque MatPlotLib.

Outils utilisés :
- Star UML pour le diagramme UML
- Eclipse comme IDE pour JAVA et JAVA FX (interface graphique)
- Spyder via Anaconda IDE pour Python
- Scene Builder pour la création des vues (interface graphique)

B – Diagramme des classes UML

Le diagramme UML ci-joint représente la structure et le fonctionnement de notre application. Il inclut les principales classes, leurs attributs et méthodes, ainsi que les relations entre ces classes. Il offre une vue d'ensemble de la structure de notre application, facilitant ainsi la compréhension de son fonctionnement et de son architecture. Il sert également de référence pour le développement futur et la maintenance de l'application.



GitHub : Diagramme des classes.mdj

C - Implémentation de l'algorithme

À la suite de la création du diagramme UML qui représente la partie 1 du développement de l'application. Nous avons fait la partie 2 et 3.

La deuxième partie a été effectuée rapidement, elle consistait à créer les différentes classes, leur attributs et méthodes. Ainsi que les classes de tests permettant d'exécuter pour la première fois notre algorithme d'affectation sur un faible nombre d'étudiants instancié à la main. Une fois la phase de test validée nous avons pu passer au développement de l'application graphique.

Dans cette troisième partie qui fut la plus longue, nous avons commencé par créer les différentes vues de notre application avec SceneBuilder à l'aide de la maquette élaborée auparavant. Enfin nous avons créé les "controllers" des différentes vues. Et effectué des simulations d'affectation sur nos bases de données, tester toutes les fonctionnalités pour chercher d'éventuels bugs et valider l'application.

D - Problèmes rencontrés

Au cours de la réalisation de l'application, nous avons prévu et/ou rencontré plusieurs problèmes auxquels nous avons dû trouver une solution. En voici quelques exemples :

Problème : Garder en mémoire le fait que l'accès au remplissage de la fiche de vœux soit activé ou non. C'est à dire, lorsque l'élève veut remplir sa fiche de vœux comment savoir si l'option a été activée par l'admin.

Solution : Création d'un fichier JSON avec un booléen, lorsque l'admin active l'option, le booléen passe à TRUE. Lorsque l'élève souhaite remplir sa fiche de vœux on regarde dans le fichier JSON si l'option est activée.

Problème : Comment actualiser le nombre de places restantes dans les vœux/filières ?

Solution : Lors du lancement de l'algorithme d'affection, on crée une liste de compteur où on stipule le nombre de places de chaque vœu. Ainsi lorsque qu'un étudiant est affecté, on décrémente le compteur correspondant dans la liste de compteur. Cette solution permet d'actualiser les places restantes sans devoir créer un nouvel attribut de vœu qui sera inutile après l'affectation. En effet puisque la somme des places disponibles de chaque vœu est égale au nombre d'étudiants à affecter (cf. Cahier des charges), cet attribut sera nul à la fin du processus d'affectation.

Problème : Comment effectuer l'affectation lorsque qu'un même vœu à plusieurs filières éligible ?

Solution : Deux solutions sont possibles, soit on classe les étudiants de plusieurs filières entre eux (classement inter filière) et donc on a un nombre de place unique. Soit on définit un nombre de place PAR filière et donc on classe les étudiants au sein de leur filière (classement intra filière). Notre application laisse le choix à l'admin, il peut soit créer un unique vœu et stipuler que plusieurs filières sont éligibles (classement inter filière), soit créer un vœu par filière éligible avec un nombre de place distinct par filière (classement intra filière). Ce choix a été motivé par la nécessité d'avoir une application générique : flexible.

V – Lien GITHUB

https://github.com/Leho777/Application-Orientation

Pour tester notre application :

1. Télécharger : OrientationProjectCode.zip
2. Dézipper dans un dossier
3. Importer le dossier “OrientationProject” via Eclipse
4. Dans “run configuration” puis arguments puis VM Arguments insérer : --module-path "C:\Divers\javafx\lib" --add-modules javafx.controls,javafx.fxml
5. Assurez-vous d’avoir les bonnes librairies, adapter les chemins des librairies à votre ordinateur (S'il y a des librairies manquantes voir dans : Clic-droit du projet -> Build Path -> Configure Build Path).
6. Compiler le main l’application devrait se lancer !

VI – Analyse de l'application & Test en condition réelle

A – Cahier des charges remplis ?

L'ensemble des fonctionnalités présente dans le cahier des charges (cf. Rapport 1) est disponible dans notre application. Voici quelques précisions sur certains points :

La principale consigne qui nous a guidé à travers le développement de l'application est le fait de tendre faire une solution générique, flexible qui pourra être utilisée par n'importe quel organisme. C'est pour cela que, bien que dans le cahier des charges nous avions spécifié que seul le classement intra-filière serait disponible, finalement nous avons ajouté l'option inter-filière car il suffisait de modifier la classe vœu et notamment son attribut filières éligibles qui auparavant était un type String et est devenu un type ArrayList<String>.

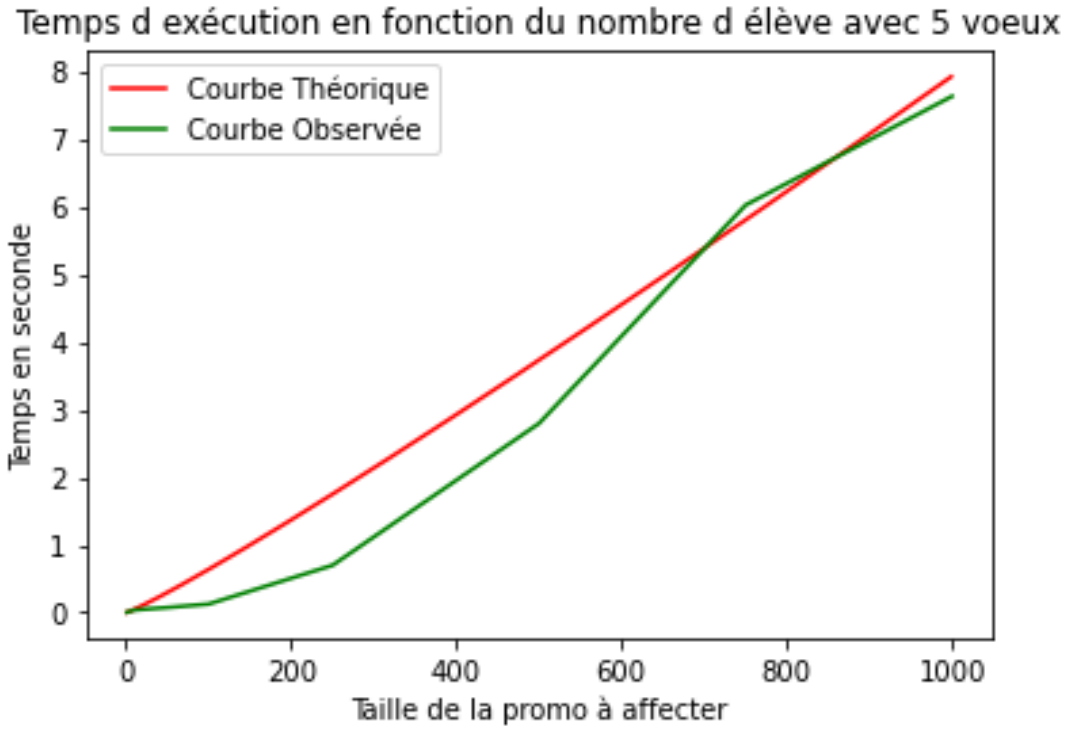
B – Performance de notre application : Théorie VS Pratique

Le but ici est de mesurer la performance de notre application et notamment la vitesse d'exécution de l'opération d'affectation. Cette étape est cruciale car c'est l'un des points principaux sur laquelle une application peut être jugée : son efficacité.

Nous avons donc testé l'application sur plusieurs tailles de promotions i.e. sur plusieurs groupes d'étudiants à affecter, et mesurer le temps que met notre application à fournir le résultat. Bien que notre application fonctionne sur quelques dizaines d'étudiants rentré à la main. En est-il de même pour une promo de 100, 500 ou 1000 étudiants ?

L'enjeu ici est de générer ces étudiants.

Pour se faire nous avons créé un script Python prenant en entrée un entier n (nombre d'étudiant souhaiter) et qui génère n étudiants avec la bonne syntaxe en format json. Que nous pouvons utiliser directement sur l'application et ainsi calculer son temps d'exécution. Pour éviter des biais et avoir une complexité moyenne, il a fallu éviter des cas particuliers. Par exemple il faut que les étudiants aient une liste de préférences générée aléatoirement (ce qui est le cas) et aussi paramétrer correctement les nombres de places des filières disponibles car s'il y a trop de place tout le monde sera pris sur son premier vœu, et donc la complexité sera dans le meilleur des cas. Or, nous voulons une complexité moyenne qui reflète bien la réalité. Cela nous a permis d'avoir une modélisation qui s'approche davantage du réel où tous les étudiants ne sont pas pris sur leur 1er vœu.



Voici un graphique montrant nos résultats :

En théorie, nous avions démontré dans notre Rapport 1 que notre algorithme avait une complexité en : O(n*(log(n) +m)) où n était le nombre d'étudiant à affecter et m le nombre de vœux. Asymptotiquement les courbes ont tendance à se rapprocher ce qui tend à valider notre modèle. Ainsi notre application à bien la complexité annoncée dans le rapport 1 et est donc plus performante qu'un algorithme n'ayant pas simplifié celui de Gale et Shapley.

Par exemple pour une promotion de 248 étudiants notre application met 0.6 secondes pour affecter les étudiants. Cet exemple est à prendre avec précaution car très dépendante des capacités de l'ordinateur. En effet, pour plus de 1000 étudiants nos ordinateurs n'avaient plus la capacité d'effectuer l'affectation et donc les temps explosent. Cependant, l'important est la complexité, car passé d'une complexité quadratique à une complexité quasi-linéaire représente un gain de temps conséquent sur un grand nombre d'étudiant.

C – Regard critique

Plusieurs points auraient pu être amélioré dans notre application, notamment des réductions de complexité spatiale et des ajouts de fonctionnalités intéressantes :

Dans les fichiers json des étudiants leur liste de vœux contiennent toutes les informations de chaque vœu. Or on aurait pu se contenter de l'attribut id du vœu pour gagner en complexité spatiale. Mais ce gain aurait eu un coup en complexité temporelle car avec seulement l'id cela aurait ajouté des parcours de collection pour retrouver le vœu.

Notre application nécessite le respect d'un protocole précis stricte :

1. L'admin créer les vœux,
2. Ouvre la possibilité de remplir sa fiche de vœux
3. Les élèves font leur fiche de vœux
4. L'admin ferme la possibilité de remplir sa fiche de vœux
5. L'admin lance l'opération et consulte les résultats

Le non-respect de ce protocole entraînera des erreurs dans notre application. Notamment si l'admin ouvre la possibilité de choisir ses vœux puis ferme, ajoute un nouveau vœu puis réouvre. Il y aurait des erreurs pour les étudiants ayant déjà effectué leur liste de vœux. Ce cas aurait pu être géré par notre application mais aurait impacté notre complexité avec des parcours régulier de collection pour les actualiser en temps réelles. Ainsi après ouverture, plus de modification possible des vœux disponibles ou alors il faut réinitialiser chaque liste de vœux de chaque étudiant et recommencer la procédure du début.

Avec plus de temps voici les fonctionnalités que nous aurions souhaité intégrer :

- Permettre à l'admin de choisir lui-même l'algorithme d'affectation, avec des algos par défaut (notamment le nôtre) mais aussi un outil de création d'algorithme simplifié. Permettant ainsi de rendre l'application encore plus générique.
- Ajout d'indice de sélectivité des vœux, un indice qui montre si le vœu et plus ou moins demandé. Permet à l'élève de savoir quelles sont les vœux les plus demandés (sur base des années précédentes).
- Créer un exécutable de notre application (.exe)
- Gérer les cas d'égalité : conflit entre deux étudiants ayant la même moyenne est souhaitant le même vœu. Dans notre cas c'est le premier étudiant dans la liste qui l'emporte mais il serait intéressant de lever un avertissement pour l'admin pour montrer que l'affectation de ces deux étudiants mériterait une étude plus approfondie de leur dossier.

VII – Bilan

Le véritable point fort de notre démarche a été notre capacité à anticiper et planifier chaque étape du développement de notre application. En élaborant une solution algorithmique dès le départ et en établissant un cahier des charges détaillé, nous avons pu naviguer sereinement à travers les différentes phases du projet. Ce qui nous a vraiment guidés lors du développement de l'application, c'était l'objectif d'avoir une application aussi générique et flexible que possible.

L'utilisation de la méthode Agile a également été un facteur clé de notre succès. Cette approche nous a permis de rester flexible et réactifs face aux défis rencontrés, tout en maintenant un rythme de travail constant grâce aux sprints et à un backlog régulièrement mis à jour. La rétro-ingénierie, en nous permettant de penser d'abord à la solution finale avant de commencer le développement, a été particulièrement efficace pour anticiper les points de difficulté et les résoudre de manière proactive.

Notre objectif de flexibilité et de généricité s'est manifesté à plusieurs niveaux. Par exemple, nous avons conçu l'application de manière qu'elle puisse être utilisée par différents types d'établissements scolaires, en permettant à l'administrateur de configurer les options d'affectation selon les besoins spécifiques. Nous avons également intégré des fonctionnalités permettant de choisir entre des classements intra-filière et inter-filière, afin de rendre l'application adaptable à divers scénarios d'utilisation.

Le choix de Java comme langage principal a apporté la robustesse et la performance nécessaires pour le développement de notre application, tandis que Python a été un outil précieux pour la génération et la manipulation de données afin de tester en condition réelle l'efficacité de notre application.

Les tests et les simulations ont montré que notre application respecte les exigences du cahier des charges et fonctionne efficacement même avec un grand nombre d'étudiants, confirmant la complexité quasi-linéaire de notre algorithme. Cependant, nous avons également identifié des améliorations possibles, notamment des fonctionnalités pertinentes pour rendre l'application d'avantage flexible, ainsi que l'utilisation d'ordinateur plus performant pour tester l'application sur un plus grand nombre d'étudiants.

En conclusion, nous sommes satisfaits des résultats obtenus jusqu'à présent. Notre application est fonctionnelle et répond aux attentes définies au début du projet. Nous voyons également des pistes pour des améliorations futures et des fonctionnalités supplémentaires, que nous sommes impatients d'explorer pour rendre notre application encore plus utile et adaptable.