Projet POOA
Optimisation de l'affectation des modules électifs

Yuchen MO, Hengshuo LI, Peiyao LI
04 Janvier 2023

Introduction

Comme le système de l'affectation nécessaire pour le GM5, les étudiants doivent choisir des modules dans son parcours préférentiel et d'autre parcours. Donc , il faut un système de l'affectation qui satisfaite autant que possible aux voeux de tous les étudiants et qui soit relativement équitable. Pour notre projet, chaque étudiant doit suivre 4 modules dans son parcours préférentiel et 2 modules de l'autre parcours, c'est-à-dire 6 modules en tout.

Table des matières

1 Description détaillée

Les étudiants saisiront leurs vœux sur un logiciel client qui se connectera à un serveur d'affectation. Chaque parcours on a 10 modules dans le premier version. Chaque étudiant donne alors ses vœux par ordre de préférence

— 8 possibilités dans son parcours préférentiel
— 4 possibilités dans l'autre parcours

Les étudiant peut demander l'affectation à tout moment. En ce cas, le serveur indiquera l'affectation courante à tous les clients (même à ceux qui ne sont pas à l'origine de la demande). Les étudiants sont affectés aux modules électifs en tenant compte

— de leurs préférences (leurs vœux)
— de contraintes sur les effectifs

1.1 La liste des fonctions principales

Il existe deux partie : client et serveur.
Pour client
— Connexion
— Login (saisir le nom)
— Choisir (voeux des parcours, modules)
— Demander (demander l'affectation de module)

Pour seveur
— Connexion
— Identification (enregistrer le nom de client)
— Choix (enregistrer de parcours préférentiel et des voeux des modules choisi)
— Optimisation (Optimisation pour obtenir une affectation de cours répondant le plus possible aux vœux de les étudiants)
— Envoyer(l'affectation)

1.2 Les différents utilisateurs et leurs caractéristiques

Nos utilisateurs cibles sont les étudiants qui doivent choisir des modules. Notre objectif est de créer un serveur qui recueille les voeux de nos étudiants et trouver une affectation des étudiants à des modules électifs optimale.

1.3 Les contraintes matérielles et logicielles

— Afin de faciliter la mise en œuvre de notre projet, dans ce projet nous ne considérerons que les options 'full' (pas les demi-options) et seulement deux parcours.
— Il existe dix modules dans chaque parcours pour le premier version, en attente du choix de l'élève.
— Effectifs : L'effectif optimum dans un module correspond à 8 étudiants. On pourrait associer une préférence (un coût) aux effectifs différents de 8. Pour simplifier cette étude, on considérera simplement qu'il s'agit de contraintes fortes : on accepte des effectifs jusqu'à MaxParModule = 10. On ne peut pas avoir des effectifs inférieurs à MinParModule = 5.
— Pour le cas réalise pas les contraintes des effectifs, on va faire l'affectation quand même mais faire un remarque provisoire à derrière.
— Le serveur calculera la meilleure affectation des étudiants aux modules avec les informations dont il dispose à un instant donné. Attention, le calcul de l'affectation peut prendre du temps, et pendant ce temps le serveur doit gérer les informations nouvelles que d'autres étudiants pourraient lui faire parvenir.

— Si le serveur se déconnecte, les affectations en cours sont perdues (pas besoin de stocker des informations des vœux des étudiants sur le disque dur du serveur).
— Le serveur pourra accueillir jusqu'à 20 utilisateurs maximum, qui pourront saisir leurs vœux en tout moment de la durée d'exécution du serveur.

2 Conception préliminaire

Nous allons présenter trois parties différentes : les spécifications fonctionnelles, les spécifications d'interfaces et les spécifications opérationnelles basées sur les diverses images UML.

2.1 Diagramme de cas d'utilisation

À l'aide du diagramme de cas d'utilisation, nous verrons les principales fonctions que le système doit réaliser.

Tout d'abord, l'étudiant a besoin de la fonction d'inscription pour donner son nom au serveur. Cela nous permet de distinguer les différents choix des différents étudiants. Ensuite, les étudiants doivent sélectionner le parcours qu'ils souhaitent le plus. Cette fonction inclut la capacité pour eux d'envoyer leur choix de parcours préférentiel au serveur. Lorsque les étudiants sélectionnent un parcours préférentiel, leur autre parcours a déjà été sélectionné (puisqu'il n'y a que deux parcours au total), nous n'avons donc besoin que d'une fonction pour sélectionner un parcours préférentiel.

Une fois que les étudiants ont choisi leur parcours, nous devons également savoir quels modules ils veulent choisir, nous avons donc besoin d'une fonction pour choisir les modules dans son parcours préférentiel et d'une fonction pour choisir les modules dans l'autre parcours. Ces 2 fonctions consistent à envoyer les choix des modules au serveur. Les étudiants auront également besoin d'une fonction demander l'affectation pour leur montrer les modules qu'ils ont finalement obtenus.

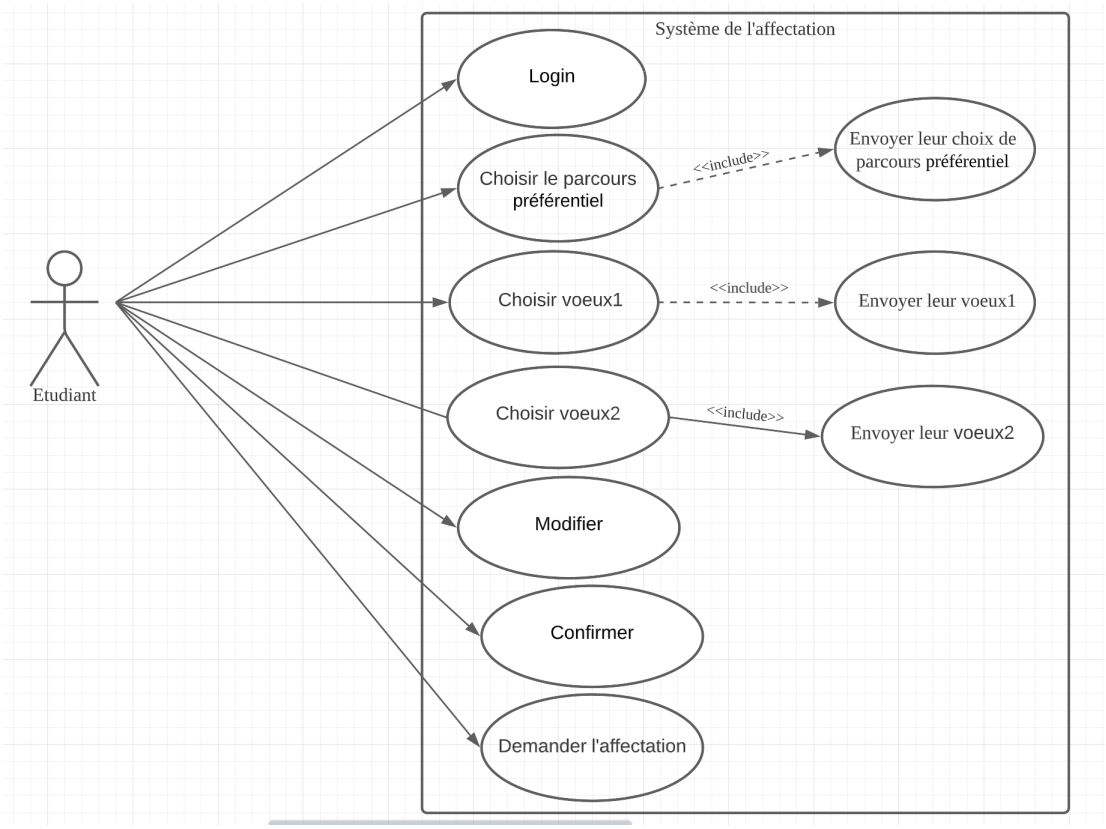


FIGURE 1 – Diagramme de cas d'utilisation

2.2 Diagramme de séquence système

Le diagramme de séquence système nous permet de voir le processus de sélection des cours par les étudiants. Les détails sont les suivants

1. Le client(étudiant) va connecter vers le système au début(c'est un option nécessaire, donc qui est un événement se caché de client), après le réalisation de connexion(vérifié le contrainte($\leqslant$20)), le système renvoie à client le état de connexion(réussie ou échoue).
2. Le client souhait s'inscrire dans le système qui est alimenté par le données nom. le système après enregistre le nom de client, va revoie le nom de client et le liste de nom de parcours.
3. Le client choisie les parcours préférentiel, le système après de enregistre le parcours préférentiel de client, lui retourne la liste des modules qui appartient la parcours préférentiel.
4. Ensuite le client choisie les 8 voeux dans la parcours préférentiel, le système quand même les enregistre et lui retourne ensemble des modules dans d'autre parcours.
5. Puis le client choisie les 4 voeux dans l'autre parcours , le système les enregistre encore une fois et lui retourner tous les choix il fait. (info) c'est à dire que le parcours préférentiel, les voeux dans le parcours préférentiel et les voeux dans l'autre parcours.
6. Si le client souhaite fait une modification alors il choisie le option modifier(confirme 0), le système va lui renvoyer le message(confirme 0), il se retrouve dans le début de boucle, et recommence les étape suivant. à la fin de boucle ,une fois le client confirme les options il fait(confirme 1), le système va enregistre les information de client dans un database, et puis renvoyer le message(confirme 1), il va sauter le boucle.
7. Le client souhaite demande l'affectation actuel, le système va lui répondre l'affectation après fait le calcule.

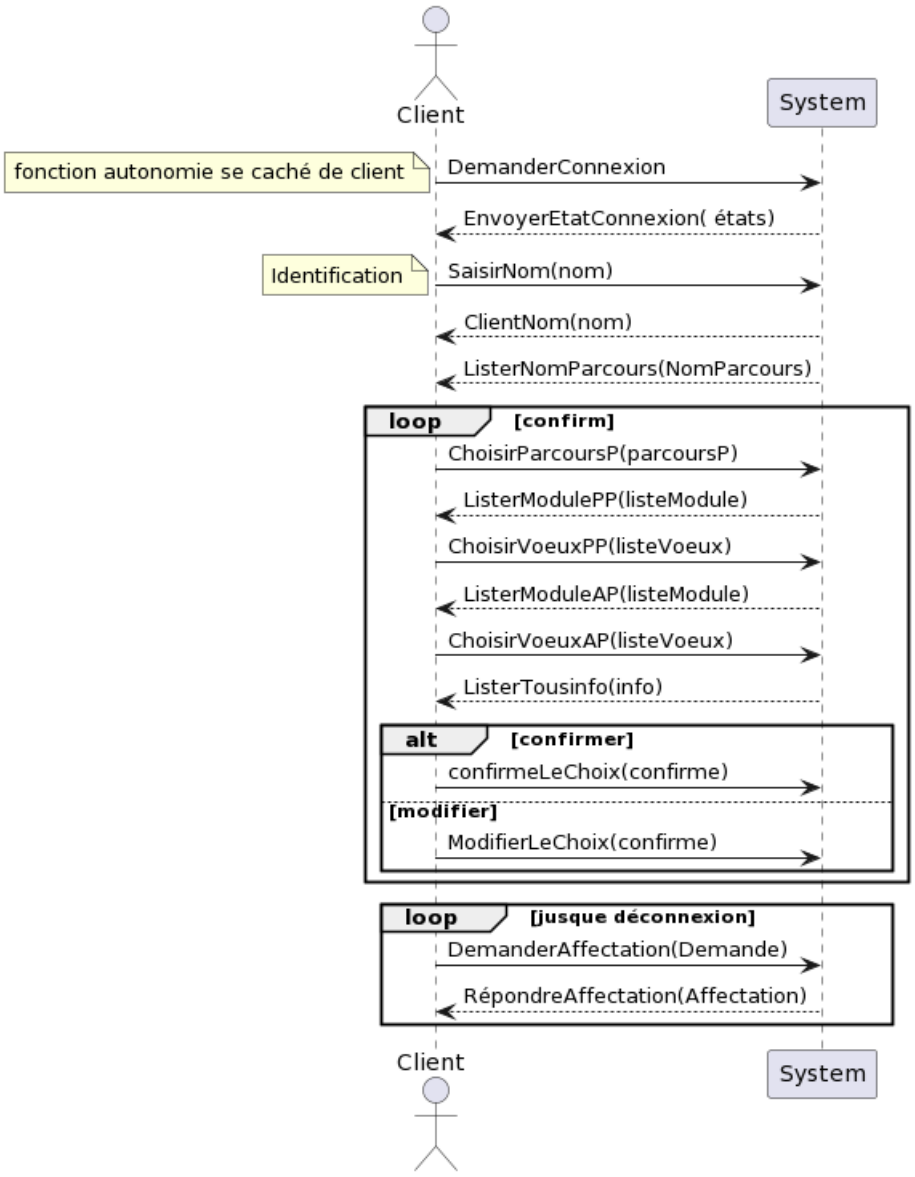


FIGURE 2 – Diagramme de séquence système

2.3 Diagramme de modèle du domaine

Ensuite, nous analysons ce dont nous avons besoin côté serveur afin de répondre aux besoins de nos utilisateurs. On a besoins de 4 classes : Module, Etudiant, Parcours, Database

- Module : Il s'agit du représentant du cours que nous avons mentionné dans le contexte.Pour attributs il contient un nom de lui même et le numéro de identification.
- Etudiant : C'est là que sont stockées toutes les informations importantes sur nos chaque étudiant. il existe cinq variable important :nom , parcours préférentiel, voeux1 =8 module, voeux2=4 module,l'affectation =6 module.
- Parcours : il contient liste de Module dans lui-même, et liste de étudiants qui choisissent cette parcours comme préférentiel(après confirmer son choix), le nom de lui même et numéro de

identification.

— Database : c'est une classe le plus important dans notre programme. il contient que liste de Parcours (déja stocker tous information de module et étudiant).

D'abord, il servie à être l'interface pour tous les appels de données de nos programmes. (c'est à dire que si on veut un liste nom de parcours, on appeler méthode de cette classe, et d'autre aussi) et puis, il existe méthode pour optimiser l'affectation.

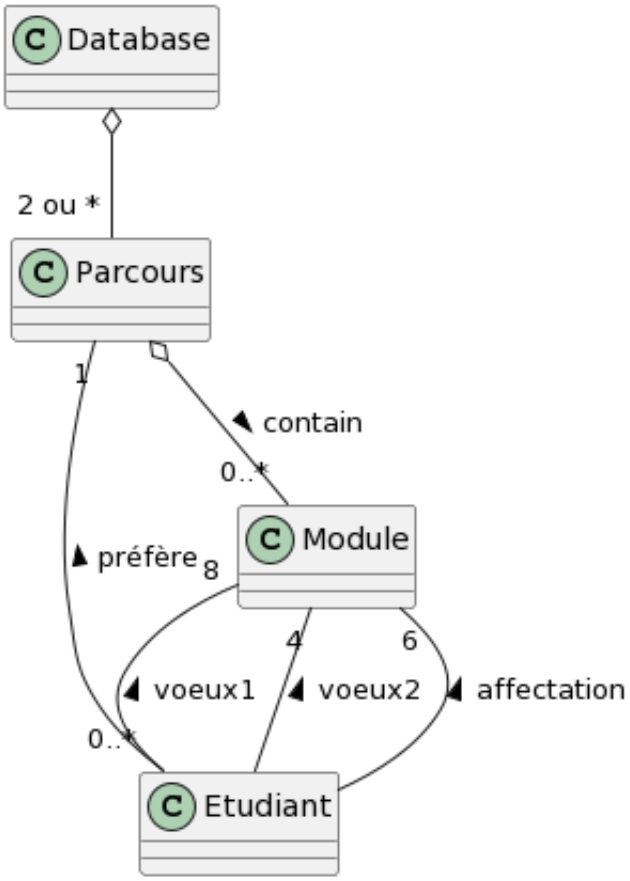


FIGURE 3 – Diagramme de modèle du domaine

2.4 Diagramme de maquettes

Ensuite le Diagramme de maquette, où nous allons simuler les effets que les utilisateurs voient réellement à l'écran, et le déroulement de leurs actions.

Dans l'écran initial, nous pouvons voir que nous nous connectons à un système de sélection de cours. Lorsque nous cliquons sur "demander la connexion", nous passons à l'écran suivant (c'està-dire l'écran de connexion).

Une fois que nous avons saisi notre nom d'utilisateur et que nous nous sommes connectés, cliquez sur "Entrer" et nous serons dirigés vers l'écran suivant.

Dans le coin supérieur gauche de cet écran, l'identifiant avec lequel nous nous sommes connectés s'affiche.Si nous cliquons sur "x" dans le coin supérieur droit, le processus termine. Cet écran nous donne également deux choix pour sélectionner le parcours préférentiel et lorsque nous l'avons fait, en cliquant sur suivant, nous passons à l'écran suivant.

Encore une fois, dans le coin supérieur gauche, nous verrons l'ID de la personne qui s'est connectée, et dans le coin supérieur central, nous marquerons qu'il s'agit de l'écran de sélection pour Parcours1. En bas, nous sommes invités à sélectionner un module : la première option est le module le plus souhaité, et ainsi de suite, pour sélectionner les 8 modules que vous souhaitez le plus suivre. Si l'on clique sur "suivant" après avoir sélectionné les huit options, on passe à l'écran suivant.

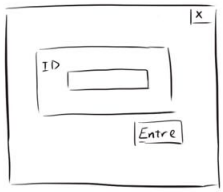
Dans cet écran, nous allons sélectionner les modules de parcours 2. Comme dans l'écran précédent, si on clique sur "suivant", on passe à l'écran suivant.

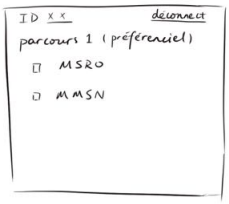
Lorsque nous arrivons à l'écran de confirmation, nous pouvons voir tous les modules que nous avons sélectionnés et si nous cliquons sur "Modifier", nous retournerons à l'écran de sélection de Parcours 1. Si nous cliquons sur "confirmer", nous passerons à l'écran suivant (l'écran d'attente).

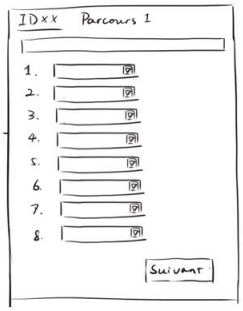
Cet écran montre notre sélection de modules et comporte un bouton pour demander l'affectation. Lorsque nous cliquons sur ce bouton, nous passons à l'écran suivant.

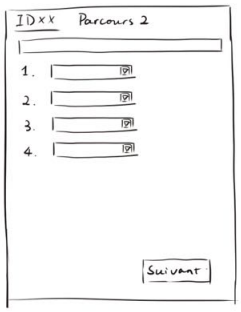
Cet écran affiche l'affectation en cours et comporte également un bouton "Demander l'affectation" permettant de vérifier les modifications apportées à l'affectation.

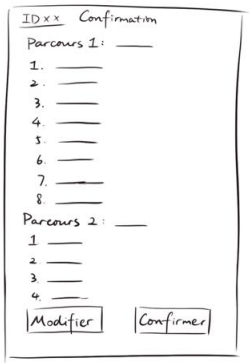

Système de sélection des cours
Demander la connexion

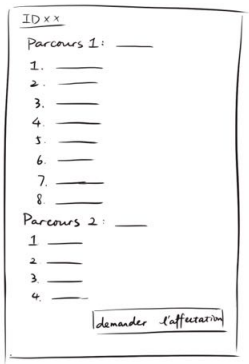

X
ID
Entre

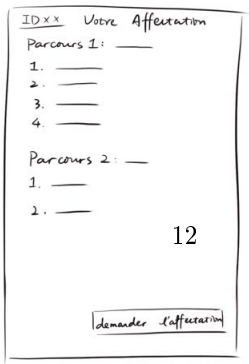

ID xx
déconnect
parcours 1 (préférentiel)
MSRO
MMSN


IDxx Parcours 1
1.
2.
3.
4.
5.
6.
7.
8.
Suivant


IDxx Parcours 2
1.
2.
3.
4.
Suivant


IDxx Confirmation
Parcours 1 :
1.
2.
3.
4.
5.
6.
7.
8.
Parcours 2 :
1
2
3
4
Modifier
Confirmer


IDxx
Parcours 1 :
1.
2.
3.
4.
5.
6.
7.
8
Parcours 2 :
1
2
3
4
demander l'affectation


IDxx Votre Affectation
Parcours 1 :
1.
2.
3.
4.
Parcours 2 :
1.
2.
demander l'affectation

2.5 Diagramme de navigation

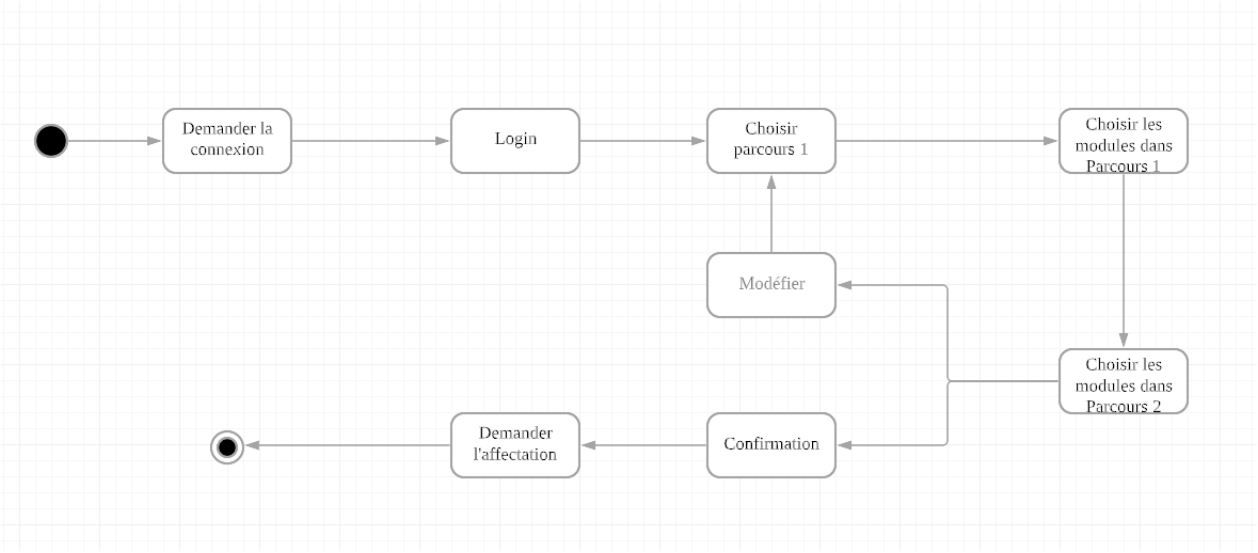


FIGURE 5 – Diagramme de navigation

2.6 Diagrammes d'interaction

2.6.1 Scénerio 1

On va aborder par rapport notre premier fenêtre (premier cas de utilisation, premier maquette)

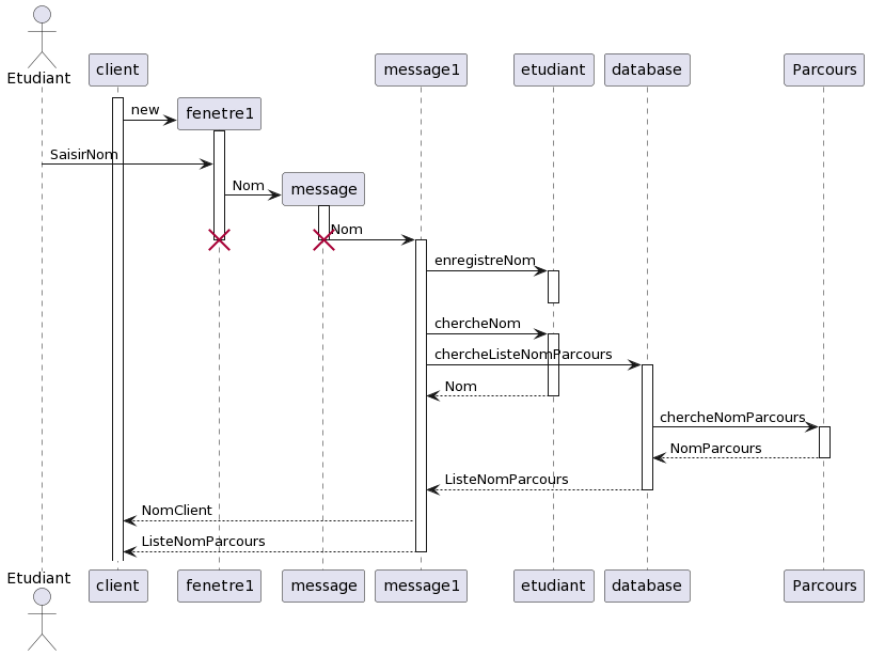


FIGURE 6 – Scénerio 1

Comme vous avez vu précédament ,le nom de chaque entité est très familier, Client comme un gestion d'ordre de interface graphique(une série de IHM), va tous d'abord , créer une fenêtre1, il y un text field pour saisir le nom, une fois Etudiant ,notre utilisateur appuyer le "save" button , le fenetre1 va créer message pour gérer le nom et mettre prefix de type de message, et envoyer le contenue à la message1 dans le serveur.

Message1 une fois reçois le message de type 1, il va évoquer le premier méthode pour enregistre le nom dans client, et puis message1 va appliquer deuxième méthode pour retourner Nom de étudiant et Liste de Nom parcours à la client.

2.6.2 Scénerio 2

On va aborder par rapport notre fenêtre2

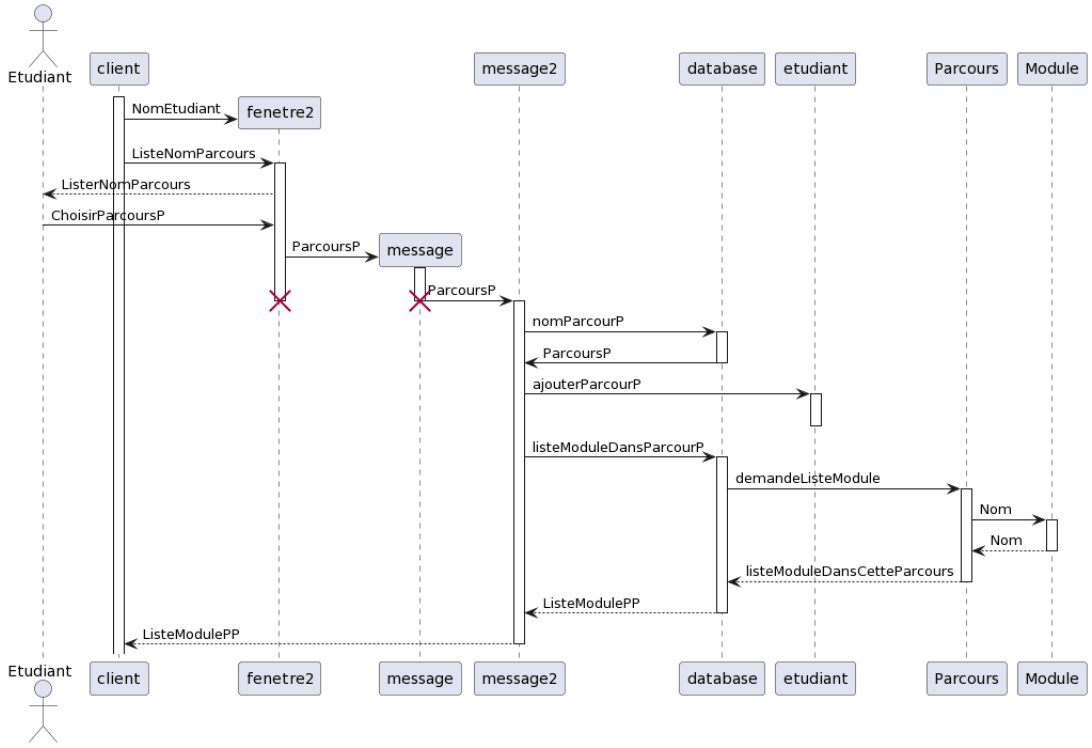


FIGURE 7 – Scénerio 2

Fenetre2 est crée par information Nom de client et la liste de Nom de Parcours il peut choisie, cette fois il faut Etudiant choisi le Parcours Préférentiel, une fois choisi est fait, on créer message pour traiter le message, et puis utiliser OutputStream de socket dans fenetre2 le envoie ver coté serveur et une fois message2 reçois le message , il va évoquer le premier méthode enregistre le parcours préférentiel dans etudaint et puis évoquer le deuxième méthode pour obtenir le liste de module dans cette parcours, et le retourner via socket.

2.6.3 Scénerio 3

On va aborder par rapport notre fenêtre3

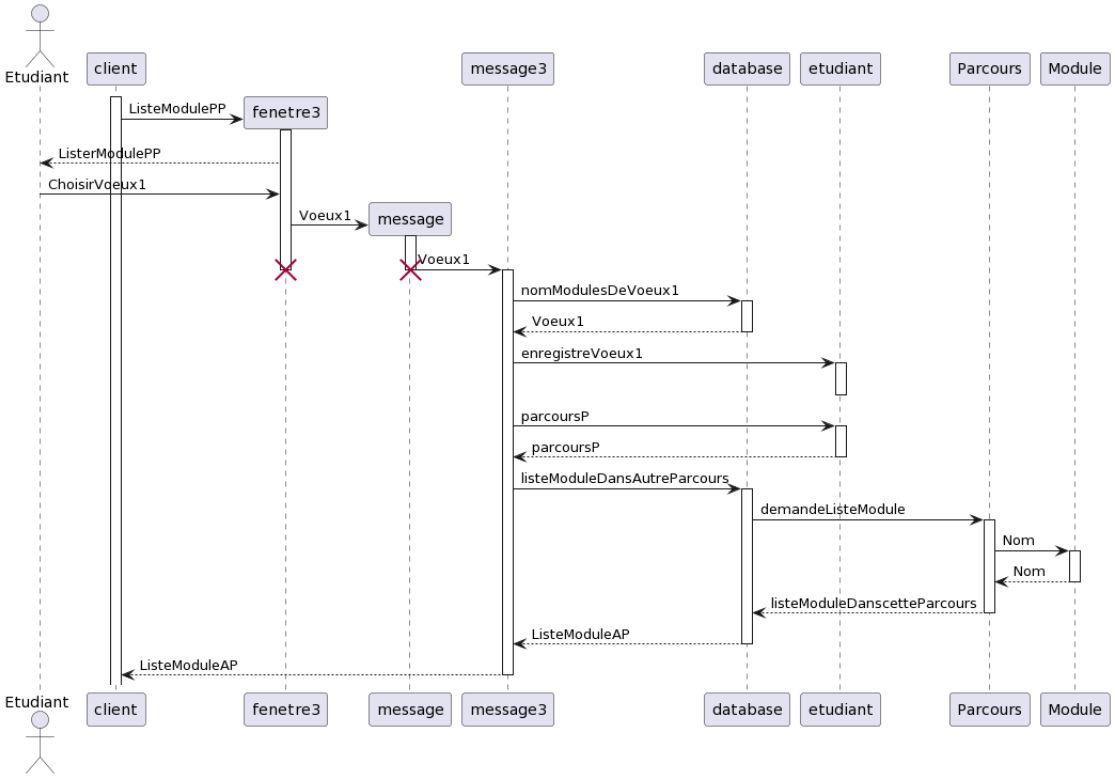


FIGURE 8 – Scénerio 3

Fenetre3 est cré par information Liste de module dans parcours préférentiel, et le forme huite combo boxes pour chosir le voeux. une fois choisi est fait , c'est le même scénario que avant, on créer message pour traiter le message string, et le envoyer par outstream de socket de fenetre3 vers coté Serveur. message3 est créer , et évoquer le premier méthode qui créer liste de module et les ajoute dans voeux1 de étuidant et deuxième méthode qui utiliser le parcours préférentiel de étudiant pour cherche les module dans l'autre parcours , et le retouner vers client.

2.6.4 Scénerio 4

On va aborder par rapport notre fenêtre4

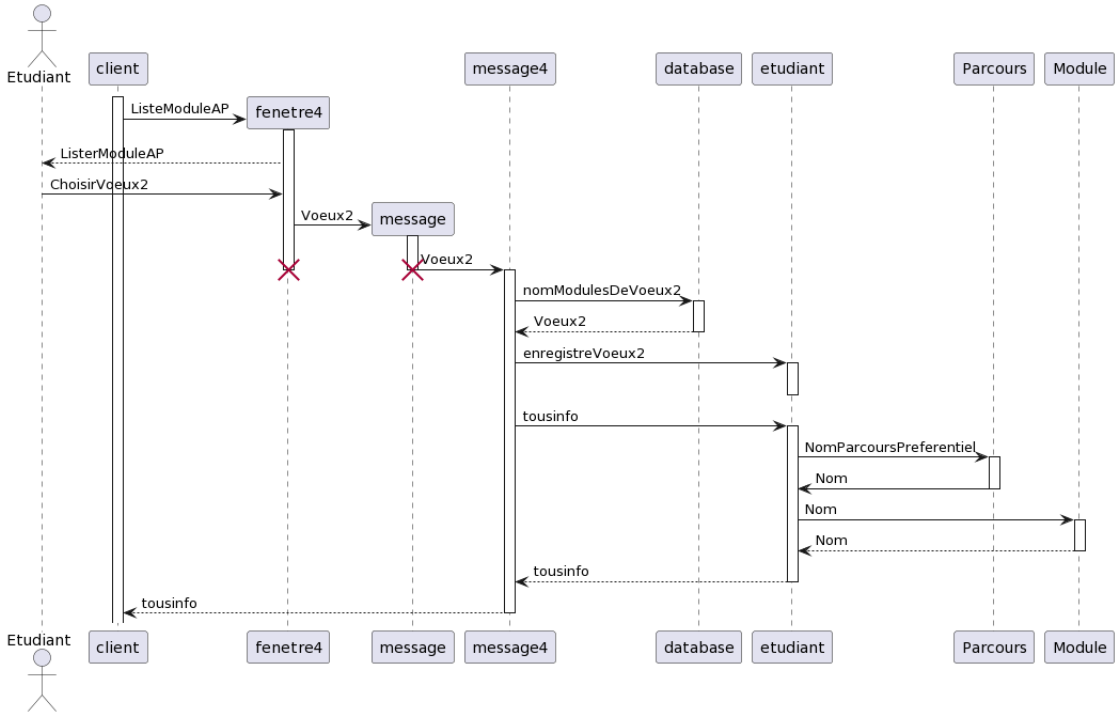


FIGURE 9 – Scénerio 4

Ce fenetre4 est fourni par Liste de module de l'autre parcours , après le Etudiant choisie le voeux , on le enregistre sous message d'abord et le envoyer vers coté Serveur à modifier le contenue message4. message4 évoque méthode mettre etudiant enregistrer le voeux2 et puis évoque deuxième pour cherche tous information sur etudiant c'est à dire parcours préférentiel+voeux parcours préférentiel et voeux l'autre parcours. envoyer tous info vers client permettre être lire par Etudiant.

2.6.5 Scénerio 5

On va aborder par rapport notre fenêtre5

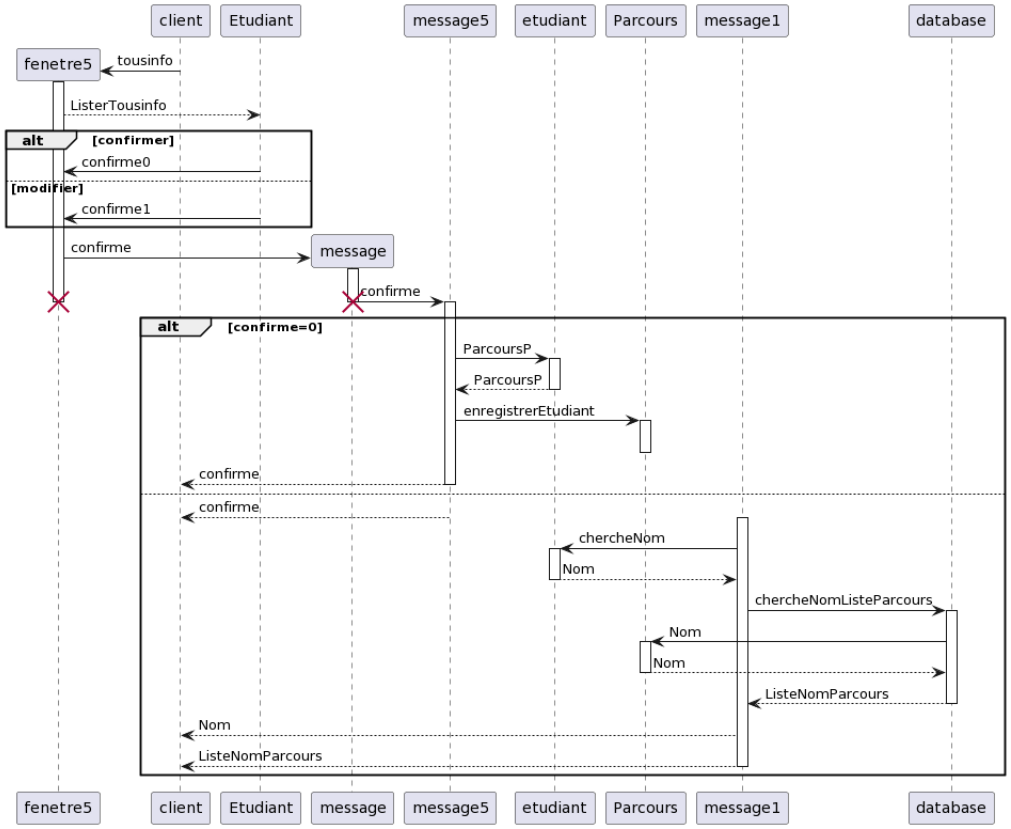


FIGURE 10 – Scénerio 5

Cet scenérios Etudiant il y a deux options ,soit confirmer info, soit modifier info si il confirme , le message5 va enregistre notre client dans database, et puis retourner confirme permettre sauter la loop (regarder diagramme générale) si il modifier , le serveur va créer message5 et message1 pour retouner confirme et retourner Nom de etudiant + Liste de Nom de parcours.

2.6.6 Scénerio 6

On va aborder par rapport notre fenêtre6

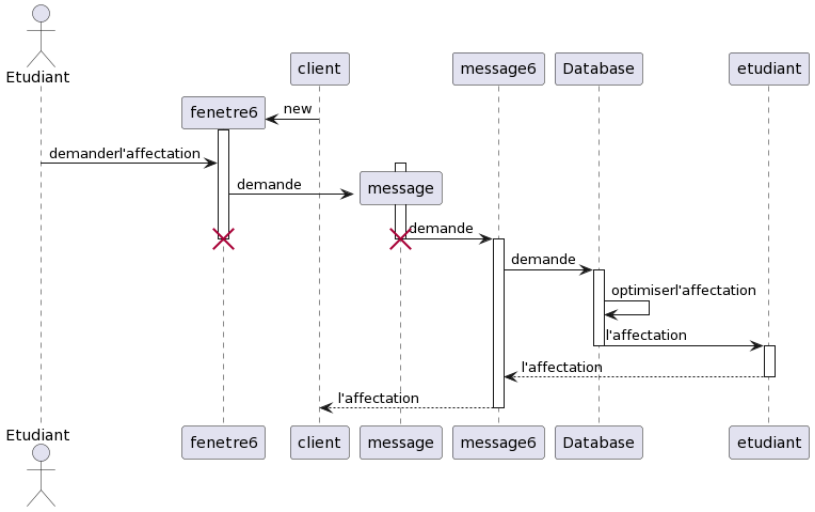


FIGURE 11 – Scénerio 6

Cette fois Etudiant peut demander l'affectation, donc message6 va demander Database de optimiser l'affectation et retourner la liste Affectation et puis le ajouter dans chaque étudiant son propos Affectation et puis le revoie ver client.

2.6.7 Scénerio 7

On va aborder par rapport notre fenêtre7(vraiment la même).

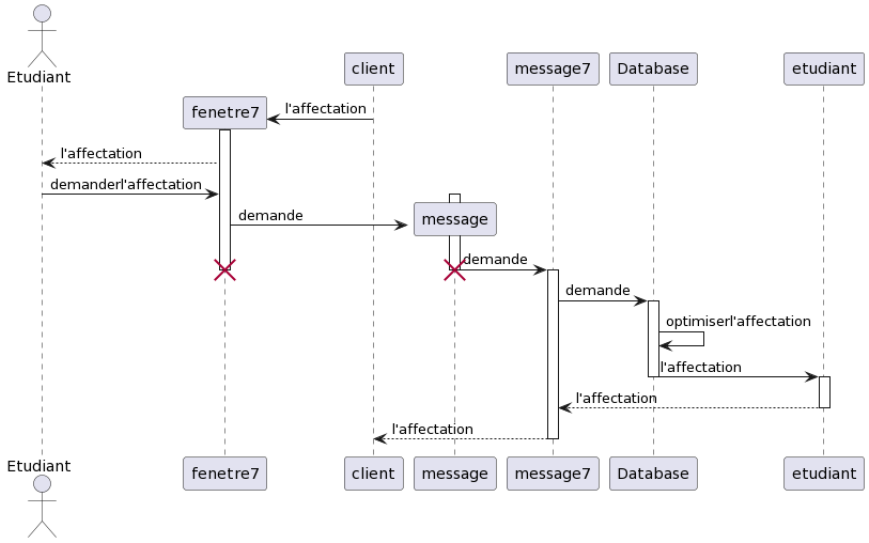


FIGURE 12 – Scénerio 7

2.7 Diagramme des classes participantes

Ici, nous analysons nos classes spécifiquement à travers le modèle de conception MVC.

2.7.1 Les classes de dialogues

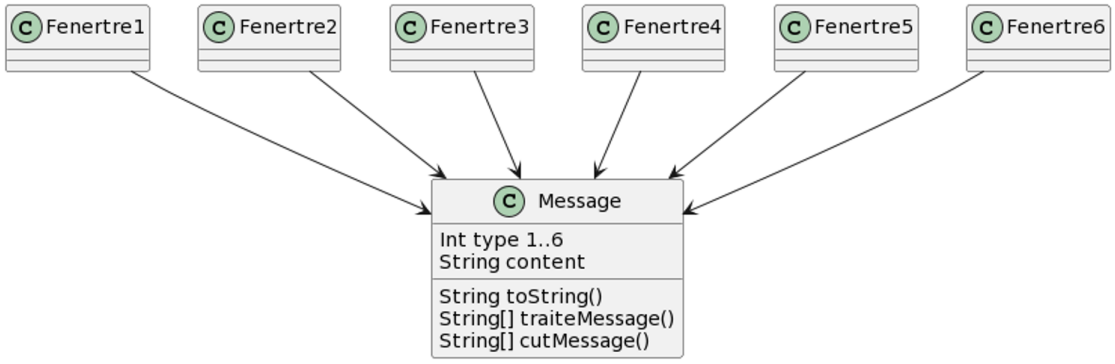
Les classes qui permettent les interactions entre l'IHM et les utilisateurs sont qualifiées de dialogues. Pour notre cas Fenetre1 à Fentre6 sont les classes de dialogue. il affiche les information vers client et puis enregistre les information dans le objet réalisée par Message. Une fois le Message récupère les information, il va envoyer vers le côte Serveur.

2.7.2 Les classes de contrôles

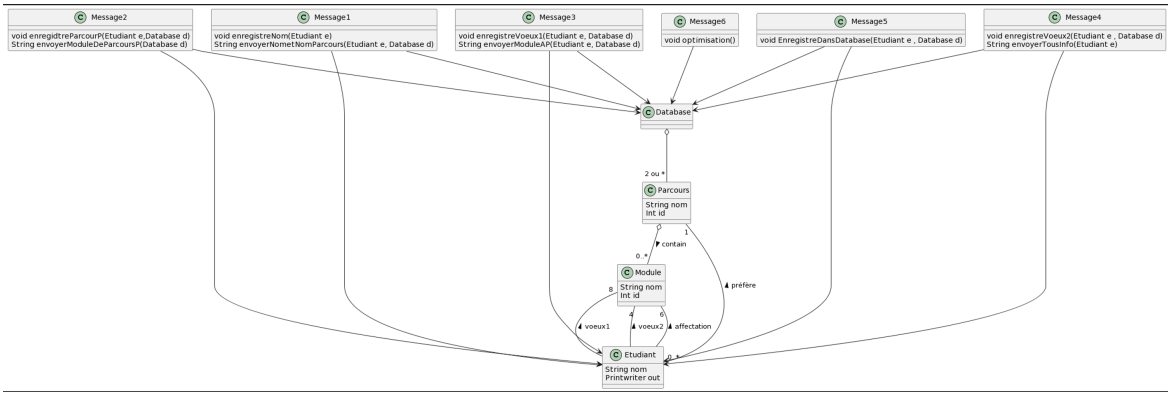
Les classes qui modélisent la cinématique de l'application sont appelées contrôles. Elles font la jonction entre les dialogues et les classes métier en permettant aux différentes vues de l'application de manipuler des informations détenues par un ou plusieurs objets métier. Elles contiennent les règles applicatives et les isolent à la fois des dialogues et des entités. Pour notre cas, de Message1 à Message6 sont notre classes des contrôle. ils disposent certaine méthode pour réagir à le Message arrivant et manipuler notre objets de Database et Etudiant. Message1 est chargée de enregistre le nom de étudiant dans objet réalisé par classe Etudiant. et puis envoyer le nom de étudiant et le liste de nom de Parcours vers coté client. Message2 est chargée de enregistre le parcours préférentiel dans objet etudiant, et renvoyer le liste de Modules dans son Parcours préférentiel vers coté client Message3 est chargée de enregistre les voeux (quantité 8) sur Parcours préférentiel et renvoyer le liste de Modules dans autres Parcours vers coté client. Message4 est chargées de enregistre lles voeux (quantité 4) sur autre Parcours et renvoyer tous les information de cette étudiant vers coté client. Message5 est chargées de soit enregistre le étudiant dans un parmi de liste de parcours de Database et renvoyer confirm(0) pour cas confirm, soit renvoyer modifier(1) et invoker le méthode de Message1 pour envoyer le nom de étudiant et le liste de nom de Parcours vers coté client. Message6 est chargées de faire le optimisation et envoyer les affectations ver chaque un.

2.7.3 Les classes entités

Les classes métier, qui proviennent directement du modèle du domaine, sont qualifiées d'entités. donc vue précédant.



(a)



(b)

FIGURE 13 – Diagramme des classes de conception préliminaire

2.8 Diagramme de composants

Nous analysons ici les structures physiques nécessaires. Pour le diagramme des composants, dans notre cas nous n'avons que le code source côté client et le code source côté serveur.

Dans le diagramme, nous en avons détaillé certains en fonction de la situation réelle. Plus précisément, les étudiants appellent le code source du client sur leurs machines, puis commencent à sélectionner des cours, et le code source du client envoie les messages saisis par les étudiants au serveur via Socket. Le côté serveur utilise le code de Accepter_Client pour créer un thread, reçoit les messages de l'étudiant, puis traite les informations via d'autres codes source du côté serveur, et fait les réponses. Normalement, ces informations doivent être enregistrées dans la base de données, mais dans notre cas, nous ne stockons pas les informations localement ou dans la base de données, uniquement dans l'attribut du code. En d'autres termes, notre situation ne permet que la sélection de cours en une seule exécution de code côté serveur.

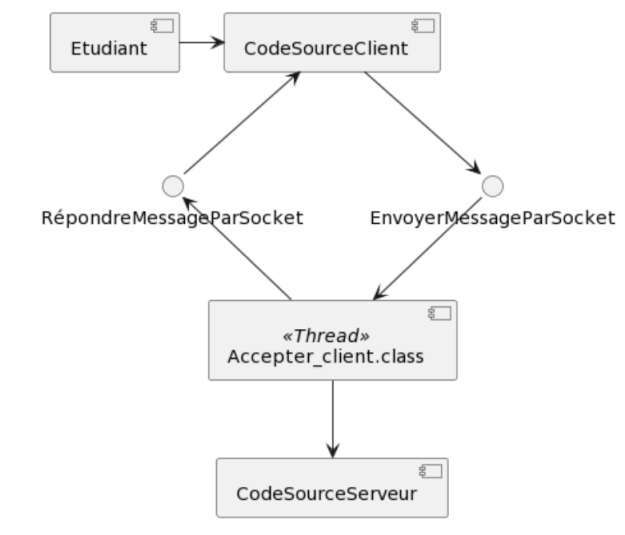


FIGURE 14 – Diagramme de composants

Par conséquent, nous pouvons également obtenir le diagramme de déploiement suivant.

2.9 Diagramme de déploiement

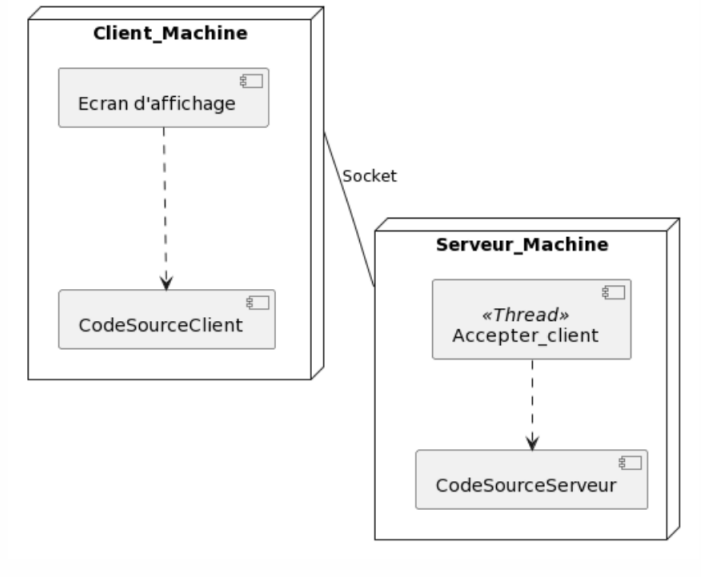


FIGURE 15 – Diagramme de déploiement

3 Conception détaillée

3.1 Diagramme de paquetages

Après la partie précédente de l'analyse, nous savons que nous devons d'abord avoir deux paquets du côté client et du côté serveur, le code source pour le client et le code source pour le serveur, et qu'ils se connectent directement par Socket.

Les deux côtés ont des paquets de Controle, qui sont utilisés pour transmettre et recevoir des messages. Comme le côté Serveur doit effectuer une opération différente à chaque fois qu'il reçoit un message, nous le divisons en 6 opérations différentes. Ensuite, du côté du client, nous avons le paquet Client, qui est utilisé pour les tests. Ensuite, le paquet View, qui est utilisé pour créer des fenêtres. Du côté du Serveur, nous avons le package Serveur, l'Accepter_Client pour créer le thread qui reçoit les informations de l'étudiant et échange ensuite les informations avec le client, et le Serveurtest pour les tests. Le package Model contient nos classes de base, qui ont été expliquées auparavant.

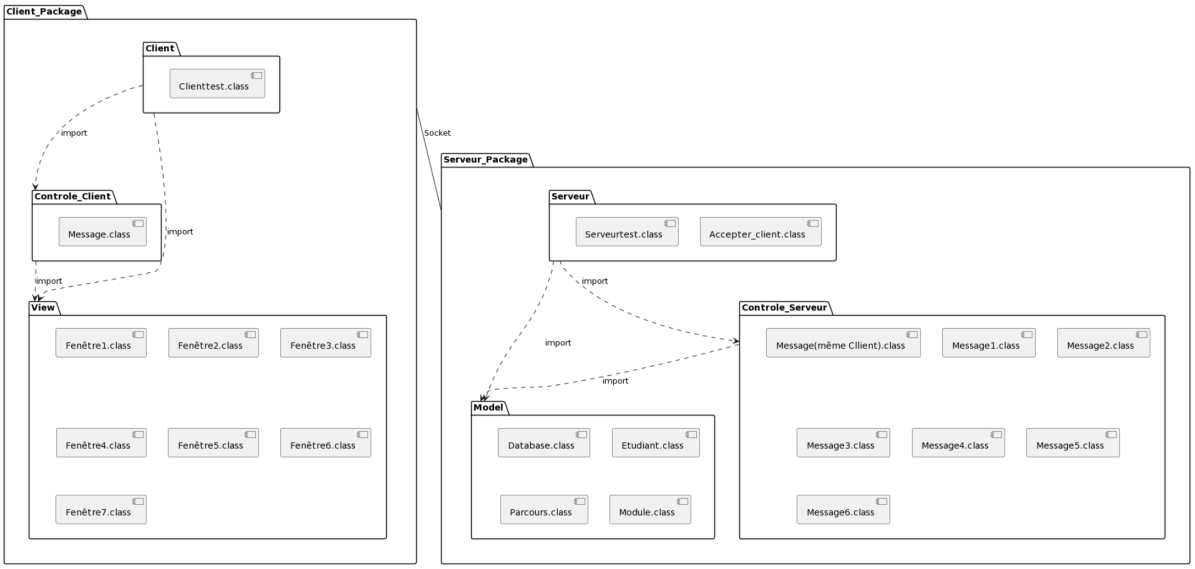
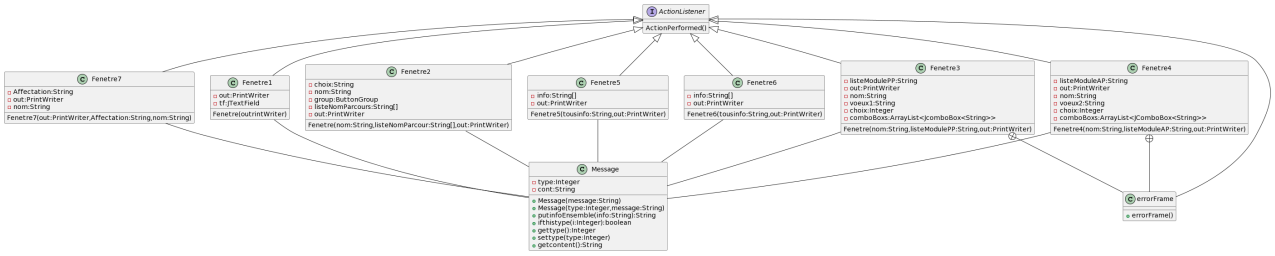
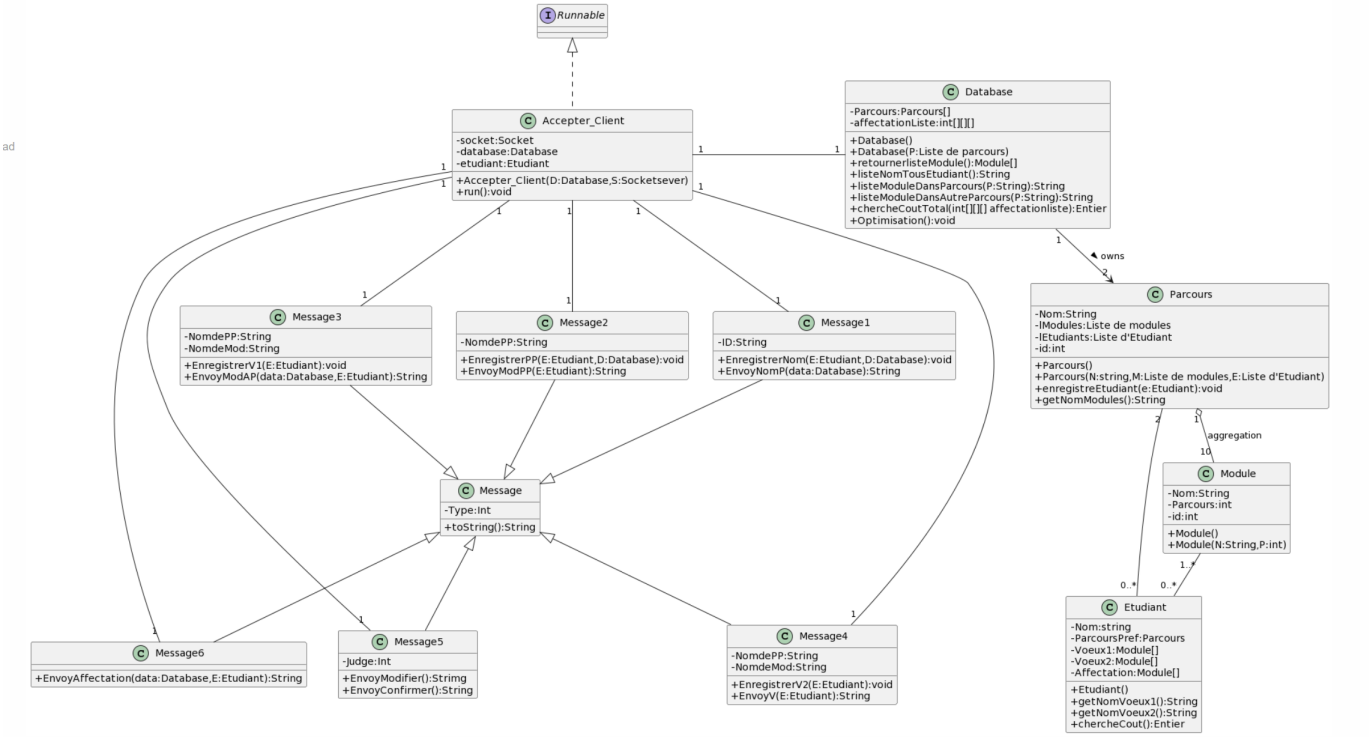


FIGURE 16 – Diagramme de paquetages

3.2 Diagramme de classes



(a) Client



(b) Serveur

FIGURE 17 – Diagramme de classes

Tout d'abord, pour client, on a la Classe Message.

Message est la partie contrôleur de notre modèle MVC, qui est responsable de la transmission des demandes et de leur traitement.

Le message contient deux attribus. L'un d'entre eux est appelé type, c'est un type Integer qui indique de quel type de message il s'agit. L'autre est appelé cont, qui est un type de String et représente le message que le Message délivré. Puisque les messages que nous lisons dans Fenetre sont des chaînes de caractères et ne font pas de distinction entre les différents composants du message (i.e. voeux 1, voeux 2, etc.), la classe Message, en plus de transmettre les messages, a pour fonction de traiter les chaînes de caractères acceptées et de les distinguer selon leurs différents composants.

Pour ce faire, nous avons deux fonctions. L'une est appelée traiteMessage(String m). m est le message que nous avons lu et cette fonction nous permet de diviser une chaîne de caractères en un

tableau de chaînes de caractères en utilisant "/" comme point de séparation. Une autre fonction est appelée cutMessage(String m). m est le message que nous avons lu, cette fonction nous permet de diviser une chaîne en un tableau de chaînes en utilisant "," comme point de séparation.

En suite, on a les classes Fenetre.

Avec ces deux fonctions, nous pouvons transformer une chaîne reçue de fenetre sans séparateur en une chaîne avec séparateur. message(String message) est une fonction qui nous permet de générer une nouvelle chaîne en séparant la chaîne reçue de fenetre selon ses différentes composantes en utilisant "/" comme point de séparation.

La fonction putinfoEnsemble(String[] info), par contre, distingue le tableau de chaînes obtenu de fenetre selon les différentes classes de cours en utilisant "," comme séparateur pour générer de nouvelles chaînes.

Nous avons sept interfaces au total, donc nous avons un total de sept classes de fenetre (de Fenetre1 à Fenetre7).

Ces classes de fenetre ont toutes besoin d'interagir avec Message, donc la relation entre Message et fenetre est une relation d'association.

Pour savoir quelle action a été effectuée dans la fenetre, nous devons utiliser l'ActionPerformed de l'ActionListener, donc les fenetres 1 à 7 implémentent toutes l'interface ActionListener.

Fenetre1 :

Fenetre1 a deux attribus dans cette classe, l'un est tf, dont le type est JTextField, et à travers ce paramètre nous obtenons le contenu du ID d'étudiant. Il y a aussi un attribu appelé out, qui est de type PrintWriter, et il a une fonction qui est une construction de cette Fenetre, appelée Fenetre(outprintWriter). Avec cette fonction, nous construisons une fenetre avec un champ de texte pour lire l'ID de connexion et un bouton "Entrer".Nous ajoutons un actionlistener au bouton "Entrer", et lorsque nous appuyons sur "Entrer", nous créons un nouveau Message : Messge(1, tf.getText()). Avec ce nouveau message, nous passons l'ID de l'utilisateur.

Fenetre2 :

Le premier attribut de la classe Fenetre2 est le choix, de type String, qui sert à indiquer le Parcours préférentiel que nous avons sélectionné dans la fenetre. Le deuxième attribut est le nom, de type String, par lequel nous obtenons le nom de l'utilisateur. Le troisième attribut est appelé out, qui est de type PrintWriter. Le quatrième attribut est appelé group, qui est de type ButtonGroup et est utilisé pour former le radiobouton. Le cinquième attribut est appelé listeNomParcours, qui est de type String[]. Il enregistre les noms de tous les parcours. Il dispose d'une fonction pour la construction de cette Fenetre, appelée Fenetre(nom :String,listeNomParcour :String[],outprintWriter). Avec cette fonction, nous construisons une interface avec l'ID de l'utilisateur dans le coin supérieur gauche, les noms de tous les parcours en dessous et un bouton suivant dans le coin inférieur droit.Nous ajoutons un actionlistener au bouton "suivant", et lorsque nous appuyons sur "suivant" nous créons un nouveau Message : Messge (2, choix). Avec ce nouveau message, nous transmettons le choix de l'utilisateur pour le parcours préférenciel.

Fenetre3 :

Le premier paramètre de Fenetre3 est listeModulePP, de type String, qui enregistre tous les modules du parcours préférentiel sélectionnés par l'utilisateur. Le deuxième paramètre est out, de type PrintWriter. Le troisième paramètre est nom, de type Le quatrième paramètre est voeux1, de type String, qui enregistre les modules du parcours préférentiel sélectionnés par l'utilisateur dans l'ordre. Le cinquième paramètre est choix, de type Integer, qui enregistre les modules sélectionnés par l'utilisateur dans la comboBox. Il est chargé d'enregistrer l'index du module sélectionné par l'utilisateur dans la comboBox.Le sixième attribut est constitué par les comboboxes, qui sont de type ArrayList<jcomboBox<String>>. Nous construisons l'interface en utilisant Fenetre(nom :String,listeModulePP :String,outprintWriter), avec l'ID de l'utilisateur dans le coin supérieur gauche, huit comboBoxes en bas pour que l'utilisateur puisse sélectionner son cours, et un bouton "suivant" en bas à droite. Nous ajoutons un actionlistener au bouton "suivant" de sorte que lorsque nous appuyons sur le bouton "suivant", s'il n'y a pas d'options en double, nous retournons

le message message(3,listeNomModule). Nous avons mis en place une classe interne pour répondre aux utilisateurs qui ont sélectionné la même option. Cette classe interne est appelée errorFrame, et nous utilisons errorFrame() pour construire une interface qui indique que l'utilisateur a sélectionné la même option, et un bouton "D'accord" pour laisser l'utilisateur confirmer qu'il sait qu'il a une option en double. Nous ajoutons un actionlistener au bouton "D'accord", de sorte que lorsque nous appuyons sur le bouton "D'accord", l'utilisateur est renvoyé à l'écran de fenetre3 et invité à faire une nouvelle sélection.

Fenetre4 :

Le premier paramètre de Fenetre4 est la listeModuleAP qui est de type String. Elle enregistre les 4 modules d'un autre parcours sélectionné par l'utilisateur. L'attribut nom est de type String, qui enregistre l'identifiant de l'utilisateur, et l'attribut voeux2 est de type String, qui enregistre le choix de l'utilisateur de modules dans un autre parcours. Comme pour Fenetre3, nous utilisons le choix pour enregistrer les indexs sélectionnés par l'utilisateur dans chaque comboBox. comboBoxs est un arrayList de comboBoxes.

Fenetre4 est construit en utilisant la fonction Fenetre4(nom :String,listeModuleAP :String,outprintWriter). Dans cette fenetre, l'ID de l'utilisateur est affiché dans le coin supérieur gauche et en dessous il y a quatre comboBoxes pour permettre à l'utilisateur de sélectionner les modules souhaités. Dans le coin inférieur droit se trouve un bouton "suivant", auquel nous ajoutons un actionListener et lorsque nous appuyons sur ce bouton, s'il n'y a pas d'option équivalente, nous retournons Message(4, voeux2). Comme pour Fenetre3, nous créons un errorFrame pour indiquer que l'utilisateur a sélectionné les mêmes modules.

Fenetre5 :

Fenetre5 a deux attributs, le premier est appelé info, qui est de type String[], un String[] obtenu en séparant la chaîne tousinfo du délimiteur "/". tousinfo contient des informations sur l'ID utilisateur, voeux1 et voeux2.

Nous construisons Fenetre5 en utilisant Fenetre5(tousinfo :String,outprintWriter). Dans le coin supérieur gauche de Fenetre5 se trouve l'ID de l'utilisateur et en dessous, un titre : Confirmation. Après cela, il y a les voeux1 et les voeux2 dans l'ordre sélectionné par l'utilisateur. En dessous, il y a deux boutons, l'un appelé "Modifier" et l'autre "Confirmer", à chacun desquels nous ajoutons un actionListener.Pour le bouton "Modifier", lorsque nous cliquons dessus, nous retournons le message Message(5," 0"). Pour le bouton "Confirmer", nous retournons Message(5, "1").

Fenetre6 :

Les attributs de la Fenetre6 sont les mêmes que ceux de la Fenetre5.Lla Fenetre6 est construite avec l'ID de l'utilisateur dans le coin supérieur gauche et ensuite le titre en dessous : Votre Voeux. En dessous se trouvent les voeux1 et voeux2 choisis par l'utilisateur (c'est-à-dire 8 pour les modules dans le parcours préférentiel et quatre pour l'autre parcour) En bas, il y a un bouton "Demander l'affectation".On l'ajoute un actionListener pour cette bouton. Lorsque nous cliquons sur ce bouton, nous renvoyons un message, Message(6," ").

Fenetre7 :

Fenetre7 a trois coefficients, le premier est appelé out, qui est de type PrintWriter. le deuxième est appelé Affectation, qui est de type String et enregistre tous les modules sélectionnés par l'utilisateur. le troisième paramètre est nom, qui est de type String et enregistre l'ID de l'utilisateur. nous utilisons Fenetre7(Dans cet écran, nous avons l'ID de l'utilisateur dans le coin supérieur gauche, le titre : Votre affectation, les six modules sélectionnés par l'utilisateur, et le bouton "demander l'affectation" en bas. En bas se trouve le bouton "demander l'affectation". Nous ajoutons un actionListener à ce bouton et lorsque nous appuyons dessus, nous renvoyons le message Message(6," "). Nous faisons en sorte que Fenetre7 hérite de JFrame, de sorte que lorsque nous effectuons des mises à jour en parallèle, nous ne mettons pas à jour plus d'une fenêtre à la fois.

Pour serveur, on a les messages d'abord. Cette partie est la partie contrôleur de notre modèle MVC, qui est responsable de la transmission des demandes et de leur traitement. Message est une classe parente. Il contient un attribut, appelé Type, qui est un type Entier qui indique de quel type

de message il s'agit (nous avons six classes Message de 1 à 6 qui héritent de Message). En effet, les messages que nous transmettons entre le client et le serveur sont tous des chaînes de caractères. Ainsi, chaque fois que nous transmettons un message, notre chaîne de caractères commence par cette variable comme identifiant afin que le client ou le serveur puisse identifier le type de message transmis. La première chose dont nous devons être conscients dans Message est la méthode toString(). Dans chaque transmission de message suivante, nous utilisons cette méthode pour former la chaîne dont nous avons besoin. Par conséquent, dans la classe parentale, nous la formatons de manière cohérente. Plus précisément, le premier caractère est l'identifiant, puis chaque élément est séparé par un espace.

Dans la sous-classe Message1, nous avons un attribut appelé ID, qui stocke le nom de la personne qui s'est connectée, le type est donc String. Tout d'abord, du côté client, lorsque nous créons une instance de cette classe Message1, nous mettons le nom entré par l'étudiant dans l'ID. Ensuite, nous utilisons la méthode toString() pour produire la chaîne de caractères du nom. pour une transmission ultérieure. Pour le côté Serveur, nous avons deux méthodes principales. L'un est EnregistrerNom(e :Etudiant,D :Database), qui stocke l'Id transmis par l'utilisateur dans Etudiant dans Accepter_Client, et vérifie s'il existe déjà quelqu'un avec le même nom dans le database, et si c'est le cas, ajoute un suffixe au nom lors de son stockage. L'autre est EnvoyNomP(D :Database), où nous lisons la base de données, qui contient les noms des parcours, et produisons une chaîne de caractères composée des noms des deux parcours, séparés par des espaces.

Dans la sous-classe Message2, nous avons un attribut appelé NomdePP, qui signifie nom de parcours préférentiel, et dont le type de données est String. Du côté du client, nous devons encore stocker le nom du Parcours Préférentiel choisi par l'étudiant lorsque nous créons l'instance Message2. Ensuite, nous utilisons la méthode toString() pour produire la chaîne de nom de parcours préférentiel. Du côté du serveur, il y a deux méthodes principales dans Message2. La première est EnregistrerPP(E :Etudiant). Dans cette méthode, nous stockons le parcours préférentiel sélectionné par l'étudiant reçu du client dans Etudiant dans Accepter_Client. La deuxième méthode est appelée EnvoyModPP(D :Database, E :Etudiant). Nous lisons le Parcours préférentiel d'attibut Etudiant dans l'Accepter_Client, puis nous lisons le nom des modules qu'il possède dans les Parcours correspondants de la base de données. Produire une chaîne de noms de ces modules, en séparant chaque cours par un espace.

Dans la sous-classe Message3, nous avons deux paramètres, l'un est NomdePP, qui est le même que NomdePP dans Message2. L'autre paramètre est NomdeMod, dont le type de données est String. Du côté client, l'étudiant trie les cours qu'il a sélectionnés par désir, les place dans NomdeMod lors de la création d'une instance de la sous-classe Message3, puis utilise la méthode toString() pour sortir la chaîne du cours sélectionné. Le parcours à cet endroit est le parcours sélectionné dans le parcours préférentiel. Du côté du serveur, il y a deux méthodes principales dans Message3, l'une étant EnregistrerV1(E :Etudiant). Le but est de stocker les cours dans Parcours préférentiel choisis par les étudiants reçus du client dans la liste Voeux1 dans Etudiant dans Accepter_Client. L'autre méthode est EnvoyModAP(D :Database, E :Etudiant), où nous lisons tous les modules d'un autre parcours dans la base de données et sortons une chaîne de ces noms, en séparant chaque parcours par un espace.

Dans la sous-classe Message4, nous avons deux paramètres, et ils sont les mêmes que dans Message3. Du côté client, nous devons stocker le cours sélectionné par l'étudiant dans NomdeMod lorsque nous créons une instance de la sous-classe Message4, puis utiliser la méthode toString() pour sortir la chaîne du cours sélectionné. Le parcours à cet endroit est le parcours d'un autre Parcours. Du côté du serveur, nous avons deux autres méthodes principales, correspondant aux deux du Message3. L'une est EnregistrerV2(E :Etudiant), qui enregistre le cours sélectionné par l'étudiant dans un autre Parcours vers l'Etudiant dans l'Accepter_Client du Message3. Liste de Voeux2. L'autre est EnvoyV (Etudiant e). Nous lisons la liste de V1 et V2 depuis Etudiant et trions les noms des cours sélectionnés par l'étudiant dans une sortie String par ordre de désir et par ordre de Parcours (Parcours préférentiel en premier). afin que les élèves puissent vérifier leurs choix.

Pour la classe Message5, dans notre conception, l'étudiant a la possibilité de modifier ou de

confirmer sa sélection à ce stade, nous avons donc besoin d'un attribut pour savoir si l'étudiant a choisi de modifier ou de confirmer, appelé un Juge. Par exemple, nous supposons le nombre 0 pour la modification et le nombre 1 pour la confirmation. Du côté client, l'élève choisira de confirmer ou de modifier. Lorsque nous créons une instance de Message5, nous stockons le choix de l'élève dans le Juge et l'éditons à l'aide de la méthode toString(). Lorsque nous créons Message5 du côté du serveur, nous stockons également ce numéro dans cet attribut. Nous savons que, du côté client, le processus de sélection tournera en boucle et ne s'arrêtera pas tant que nous n'aurons pas reçu un message du côté serveur, que nous représenterons ici sous la forme d'un nombre. Par conséquent, nous avons besoin de EnvoyConfirmer() pour donner l'information correspondant à la confirmation, et EnvoyModifier(), pour donner l'information correspondant à la modification.

Pour Message6, du côté client, si l'utilisateur choisit de visualiser les informations relatives à l'attribution des cours, nous créerons une instance de Message6 et enverrons ensuite son identifiant à l'aide de toString. Du côté serveur, nous n'avons qu'une seule méthode principale, appelée EnvoyAffectation(data :Database,E :Etudiant). Après avoir reçu un message d'un client demandant à voir les résultats de son affectation de cours, la méthode EnvoyAffectation(data :Database,E :Etudiant) est utilisée pour récupérer l'Affectation de l'étudiant dans la Base de données, formant une Chaîne de caractères dans l'ordre des voeux et des Parcours.

Pour la classe Parcours, la principale méthode dont nous avons besoin est la méthode enregistreEtudiant, nous devons enregistrer les étudiants qui ont choisi ce Parcours comme un Parcours préférentiel dans la liste d'étudiant afin de connaître le nombre d'étudiants et les étudiants qui choisissent ce parcours comme le parcours préférentiel lors de l'optimisation.

Pour la classe de module, nous n'avons pas de méthode particulière qui nécessite une attention particulière. Il convient de mentionner que nous avons conçu chaque module pour qu'il soit numéroté automatiquement et séquentiellement en fonction du Parcours dans lequel il se trouve lorsqu'il est créé. Les classes du premier Parcours sont numérotées de 0 à 10 et les classes du second Parcours sont numérotées de 11 à 20.

Pour la classe Étudiant, nos principales méthodes sont getNomVoeux1 et getNomVoeux2, qui récupèrent respectivement la chaîne des modules sélectionnées dans le Parcours Préférentiel de l'étudiant et la chaîne des modules sélectionnées dans l'autre Parcours. En outre, nous disposons d'une méthode, chercheCout, qui est utilisée pour calculer le coût du cours auquel l'étudiant est affecté.

Pour la classe Database, nous disposons de deux méthodes principales, listeModuleDansParcours et listeModuleDansAutreParcours, qui sont utilisées pour obtenir la chaîne du cours dans le Parcours lorsqu'on lui donne la chaîne Parcours. Enfin, la méthode d'optimisation, la plus importante, use une méthode d'optimisation pour attribuer le cours à chaque étudiant au coût total minimum, tout en stockant le cours reçu par chaque étudiant dans l'attribut Affectation pour chaque étudiant de la liste des étudiants de étudiants. Il existe également la méthode chercheCoutTotal qui sera appelée dans la méthode d'optimisation pour calculer le coût total.

3.3 Les méthodes non triviales

Notre main() dans la Classe Clienttest. Nous commençons par initialiser nos paramètres de sorte que confirm = 0, de sorte que lorsque nous ne cliquons pas sur le bouton de confirmation, notre confirm sera toujours égal à 0. Cela permet à l'utilisateur d'éditer ou de modifier à plusieurs reprises les parcours et les modules qu'il souhaite sélectionner. dans la boucle while, nous lisons d'abord nomEtlisteNomparcours. nomEtlisteNomparcours est sous la forme "nom /listeNomparcours",l'attribut pre est la chaîne[] qui le sépare du séparateur "/". Nous obtenons notre nom et notre listeNomparcours par traiteMessage la fonction. les apporter pour créer une nouvelle Fenetre2. Par Fenetre2, On lit le choix des modules du parcours préférentiel et après on crée le fenetre3. Par Fenetre3, On lit le choix des modules d'autre parcours et après on crée le fenetre4. By Fenetre4,On lit le voeux1 et voeux2 et après on crée le fenetre5. Par Fenetre5,On lit le choix si on choisit "confirmer", confirm=1,on sort le cycle, sinon on reste dans le cycle. Lorsque nous sortons de ce cycle, nous

créons fenetre6, un écran d'attente qui affiche les choix de l'utilisateur et lui offre la possibilité de visualiser son affectation. Lorsque l'utilisateur clique pour demander son effectuation, nous créons fenetre7, et afin de permettre à l'utilisateur de cliquer à plusieurs reprises pour visualiser la nouvelle effectuation, nous laissons la création de fenetre7 dans une boucle while jusqu'à ce que l'utilisateur clique pour fermer fenetre7 et que nous sortions de la boucle.

Algorithme 1 Main() pour Clienttest.java

```
main():void
Début:
        Socket socketclient;
        BufferedReader in;
        PrintWriter out;
    Essayer
        socketclient = new Socket(InetAddress.getLocalHost(),2009);
        write("demander de connexion");

        in = new BufferedReader(new
        ↪  InputStreamReader(socketclient.getInputStream()));
        out = new PrintWriter(socketclient.getOutputStream());
       String  info = read(in);
       print(info);
       Int confirm = 0;
       String tousinfo = " ";
       String nom = " ";

       Fenetre1 f1= new Fenetre1(out);
       Tant que(confirm == 0)
 String nomEtlisteNomparcours= read(in);

                String[] pre = Message.traiteMessage(nomEtlisteNomparcours);
                nom = pre[0];
                String listeNomparcours= pre[1];
                Fenetre2 f2 =  new Fenetre2(nom,listeNomparcours,out);
                String listeModulePP = lire(in);
                Fenetre3 f3 = new Fenetre3(nom,listeModulePP,out);
                String listeModuleAp = lire(in);
                Fenetre4 f4 = new Fenetre4(nom,listeModuleAp , out);
                tousinfo = lire(in);
                Fenetre5 f5 =new Fenetre5(tousinfo , out);
                String confirmstring = lire(in);
                confirm =
                ↪  Integer.parseInt(Message.traiteMessage(confirmstring)[0]) ;
       Fin Tantque
       new Fenetre6(tousinfo,out);
       Tantque(true)
                String Affectation =lire(in);
                new Fenetre7(out , Affectation,nom);
       FinTantque
   FinEssayer
        catch(IOException e)

            e.printStackTrace();
            System.out.println(" il y a eu déja 20 étudiant sur
            ↪  connection,byebye");
        Fincatch;
Fin
```

Notre fonction run() dans la Classe Accepter_client

Algorithme 2 Run() pour Accepter_Client.java

```
ALGO Accepter_client.run()
début
        Etudiant e = new etudiant
        e.getsocketout(socket.out)
        Tantque (socket.close) faire
                message = socket.in
                Message m = new Message(message)
                switch(m.gettype)
                case 1 :
                        Message1 m1 =new Message1(m)
                        m1.enregistreNomDeEtudiant(e)
                        socket.out =
                        ↪ m1.envoyerNomDeEtudiantEtListeNomdeParcours(e,database)//database:stocke
                        ↪ les infos de parcours et modules et etudiant qui
                        ↪ remplir tous info.
                case 2 :
                        Message2 m2 =new Message2(m)
                        m2.enregistreParcoursPreferentiel(e,database)
                        socket.out = envoyerListeModuleDeParcoursPreferentiel
                case 3 :
                        Message3 m3 =new Message3(m)
                        m3.enregistreVoeux1(e,database);//Voeux1 comme précédant
                        socket.out = envoyerListeModulesDeAutreParcours
                case 4 :
                        Message4 m4 =new Message4(m)
                        m4.enregistreVoeux2(e,database);//Voeux2 comme précédant
                        socket.out =
                        ↪ EnvoyerTousInfo(e)//tousinfo:nom+ParcoursPréférentiel+voeux1+voeux2
                case 5 :
                        Message5 m5 =new Message5(m)
                        socket.out=m5.confirm()
                        Si(m5.getconfirm) alors
                                m5.enregeistreDansDatabase(e,database)//enregistrer
                                ↪ étudiant dans database.
                        else
                                socket.out=
                                ↪ m1.envoyerNomDeEtudiantEtListeNomdeParcours(e,database)
                        finsi
                case 6 :
                        database.optimisation()
        fintantque
fin
```

Pour l'optimisation, nous utilisons l'algorithme génétique. Dans l'algorithme, on remplace tous les cours par son numéro (id), donc le type est un entier. Chaque individu de l'algorithme génétique est une affectation aux cours de tous les étudiants à la fois, que nous stockons dans un tableau

tridimensionnel. Ce tableau tridimensionnel est divisé en deux couches ; la première couche est l'attribution des cours aux étudiants qui ont choisi le premier Parcours comme leur Parcours Préférentiel. Le deuxième niveau est la répartition des cours pour les étudiants qui ont choisi le deuxième Parcours comme Parcours Préférentiel. Dans notre classe Database, nous avons l'attribut listeParcours, un tableau avec nos deux Parcours, et nous avons stocké les étudiants avec les informations dans le LEtudiants du Parcours Préférentiel qu'ils ont sélectionné. Ainsi, nous utilisons un tel tableau tridimensionnel pour faciliter les opérations d'optimisation et nous évitons un grand nombre de situations coûteuses en initialisant directement en fonction du Parcours Préférentiel de l'élève. Cela permet d'obtenir de meilleurs résultats d'optimisation. Nous avons choisi d'effectuer 100 itérations génétiques.

Algorithme 3 Optimisation() pour Database.java

```
Optimisation() :void
Var final nbSelect100, i:int, init:int[][][][], select:int[][][][]
Début
        i=0
        initinitalilize()
        Tantque (i<nbSeclect)
                selectselection(ini)
                initgivebirth(select)
                i+=i+1
        Fintantque
        this.affectationListeinit[0]
        retournerListeAffectation()
Fin
```

Nous avons utilisé la méthode selection pour sélectionner les 10 situations d'attribution de cours les moins coûteuses en tant que parents.

Algorithme 4 Selection pour Database.java

```
Selection(ensembleaffectationListe :int[][][][]) :int[][][][]
Var final selectNumber10, select:int[][][][]
Début
        Arrays.sort(ensembleaffectationListe,(first,second)->chercheCoutTotal(first)-chercheCoutTota
        ↪           selectArrays.copyOf(ensembleaffectationListe,selectNumber)
        ↪
        return(select)
Fin
```

Dans chercheCoutTotal, nous calculons le coût total de tous les étudiants affectés en une seule fois. Le coût d'un étudiant est calculé en appelant chercheCout dans Etudiant, puis additionné par une boucle. coutNombrelimite est utilisé pour calculer le coût d'une allocation due au fait qu'une session n'atteint pas le nombre minimum d'étudiants (5) ou dépasse le nombre maximum d'étudiants (10). Pour éviter cela, nous avons fixé le coût pour ce cas à un niveau très élevé, fixé à 1000.

Algorithme 5 chercheCoutTotal pour Database.java

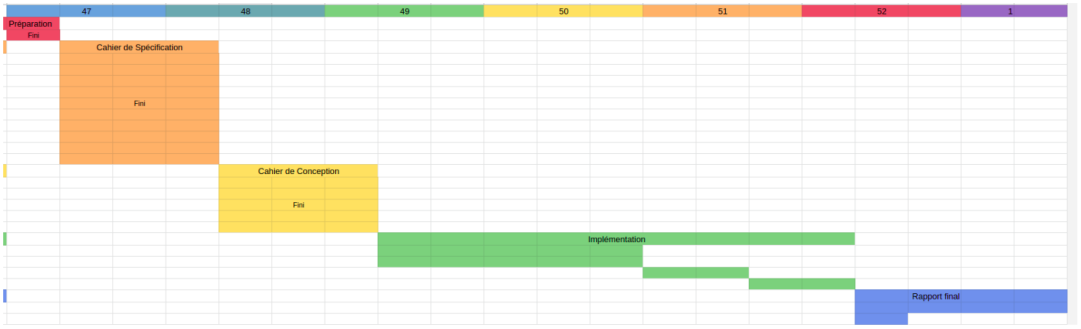
```
chercheCoutTotal(affectationliste :int[][][]) :int
var cout :int, afP :int[][], el :Etudiant[], i :int, j :int
Début
        cout=0
        Pour i0 à 2 Inc +1 faire
                afPaffectationliste[i]
                elthis.listeParcours[i].getEtudiant()
                Pour j0 à el.length Inc +1 faire
                        Cout +=el[j].chercheCout(afp[j])
                FinPour
        FinPour
        cout += coutNombrelimite(affectationliste)
        return(cout)
Fin
```

Nous utilisons la méthode givebirth pour échanger et faire muter des parents sélectionnés dans une certaine mesure pour produire les enfants.

Algorithme 6 givebirth pour Database.java

```
givebirth(select :int[][][][]) :int[][][][]
Var final Numbrechild10, final tauxCrossover0.4, tauxMutation0.005,
↪  child:int[][][][], k:int, i:int, j:int
Début
        k=0
        Pour i0 à select.length Inc +1 faire
                child[k]select[i]
                k+=1
        FinPour
        Pour i0 à select.length-1 Inc +1 faire
                Pour j0 à Numbrechild Inc +1 faire
                        child[k]=echangeligne(select[i],select[i+1],tauxCrossover,tauxMutation)
                        ↪
                        k+=1
                FinPour
        FinPour
        Return(child)
Fin
```

4 Planning



Activity	Personne en charge	Status	Début	Fin	Jours
Préparation		Fini	21/11/22	23/11/22	2
Lire et comprendre du projet	Les 3	Fini	21/11/22	23/11/22	2
Cahier de spécification		Fini	23/11/22	30/11/22	7
Comprendre le cas pour utiliser ce projet	Les 3	Fini	23/11/22	30/11/22	7
Trouver l'objectif du projet	Les 3	Fini	23/11/22	30/11/22	7
Trouver les règle de gestion et des contraintes d'intégrité	Les 3	Fini	23/11/22	30/11/22	7
Trouver les besions des fonction pricipale	Les 3	Fini	23/11/22	30/11/22	7
Touver le diagramme de cas d'utilisation	Yuchen MO	Fini	23/11/22	30/11/22	7
Trouver le diagramme de séquence	Hengshuo LI	Fini	23/11/22	30/11/22	7
Spécifier fonction détaille	Yuchen MO	Fini	23/11/22	30/11/22	7
Faire une visualisation et Maquette	Peiyao LI	Fini	23/11/22	30/11/22	7
Trouver un diagramme de navigation	Peiyao LI	Fini	23/11/22	30/11/22	7
Trouver un diagramme de modèle du domaine	Hengshuo LI	Fini	23/11/22	30/11/22	7
Cahier de conception		Fini	30/11/22	07/12/22	7
Trouver un diagramme d'interaction	Hengshuo LI	Fini	30/11/22	07/12/22	7
Trouver un diagramme des classes de conception préliminaire	Peiyao LI	Fini	30/11/22	07/12/22	7
Trouver un découpage de composants avec les interfaces requises et fournies	Yuchen MO	Fini	30/11/22	07/12/22	7
Trouver un diagramme de déploiement	Yuchen MO	Fini	30/11/22	07/12/22	7
Faire une explication des diagrammes	Les 3	Fini	30/11/22	07/12/22	7
Implémentation			07/12/22	27/12/22	20
Coder les classe dans le Client(GUI principale)	Peiyao LI		07/12/22	17/12/22	10
Coder les classe dans le Serveur	Yuchen MO		07/12/22	17/12/22	10
Coder les connextion entre Client et Serveur	Hengshuo LI		07/12/22	17/12/22	10
Debug de code	Les 3		17/12/22	22/12/22	5
Tester le code et analyser les résultats	Les 3		22/12/22	27/12/22	5
Rapport final			27/12/22	04/01/23	7
Rédaction d'un rapport final	Les 3		27/12/22	04/01/22	7
Faire le vidéo	Les 3		27/12/22	29/12/22	2

FIGURE 18 – Planning

5 Choix techniques justifiés

Pour le choix de la technologie, nous discutons principalement de deux aspects, l'un est le choix de la méthode de connexion, et l'autre est le choix de l'algorithme d'optimisation.

5.1 Méthode de connexion

Pour la connexion entre le client et le serveur, nous choisissons la méthode TCP. Parce que nous ne considérons que la situation où 20 personnes sont connectées, le nombre de personnes connectées est faible et l'utilisation de TCP ne sera pas lente et n'entraînera pas trop d'occupation des ressources. Et TCP est une connexion plus fiable et plus stable. Dans la sélection des cours, nous devons nous assurer qu'il n'y a pas de perte dans la transmission des fichiers, sinon cela entraînera des erreurs dans les résultats. Par conséquent, nous choisissons d'utiliser la méthode TCP pour nous connecter.

5.2 Algorithme d'optimisation

5.2.1 Modélisation mathématique du problème d'optimisation

$$min_{\forall i \in D, 5 \leqslant \sum_{x \in E} g_i(x) \leqslant 10} \sum_{x \in E} c(x)$$

x : Affectation de chaque étudiant.
c : Coût total de cette affectation.
E : Les ensembles des affectation de chaque étudiants.
D : le nombre module totale.
g_i : pour i-mème module si le affectation contient cette module, 1 :existé ; 0 : non existé.

Si on écrite le terme coût sous forme linéaire $c(x)$, alors naturellement, il nous faut écrire x comme $(0, 0, ..., 0, 1, 0, ...0, 1, 0, ...0, 0)^t$, pour la taille D le nombre module totale, et notre c on peut désigne comme un matrice C. Chaque ligne est représente un élève , est différent en raison de voeux ils prennent la taille $card(E) \times D$ et on peut former X colonne par colonne par tous les $x \in E$ de taille $D \times card(E)$ et on regarde que le valeur dans la diagonale. et pour notre contrainte g_i, effectivement , il nous faut assurer de chaque composante de $5 \leqslant X \cdot 1^t \leqslant 10$.

On ne peut pas utiliser les méthodes comme moindre carrées , car on ne peut pas assurer C est de rang D . Aussi, c'est très compliquées de résoudre de ce problème par méthode de directe , ou d'autre méthode sur optimisation linéaire , car on ne peut assurer le système vérifier le condition nécessaire de chaque méthode et unicité de solution. Donc on va choisir d'autre moyen pour résoudre cette problème. Donc, on veut choisir les méthodes comme algorithme génétique.

5.2.2 Algorithme génétique

Les algorithmes génétiques simulent les processus évolutionnaires darwiniens et génétiques tels qu'ils s'appliquent aux chromosomes.

Ils transforment un ensemble d'objets mathématiques, une population d'individus souvent représentés par des chaînes de caractères pour imiter des chaînes de DNA, chacune avec une valeur d'adaptation, en une nouvelle population.

— Une solution = Un individu
— Un ensemble de solutions = une population (taille p)
— Chaque nouvelle population est appelée une Génération
— Le codage d'une solution = chromosome (longueur n)
— Un caractère de la solution = un gène
— Des solutions sont recherchées dans le quartier par les opérateurs génétiques

Et on a quatre opérateurs génétiques de base

— évaluation du niveau d'adaptation d'un individu.
— la sélection : c'est le choix des individus selon leur niveau d'adaptation.
— croisement ou recombinaison : c'est le brassage du bagage génétique.
— mutation : le bagage génétique est brusquement modifié, éviter de rester coincé dans un optimum local

Pour évaluer la pertinence d'une solution par rapport à une autre, on introduit ce que l'on appelle une fonction d'adaptation, qui correspond à l'utilité de la solution par rapport au problème.

Grâce au processus de sélection naturelle simulée mentionné ci-dessus, nous pouvons enfin obtenir la solution optimale.

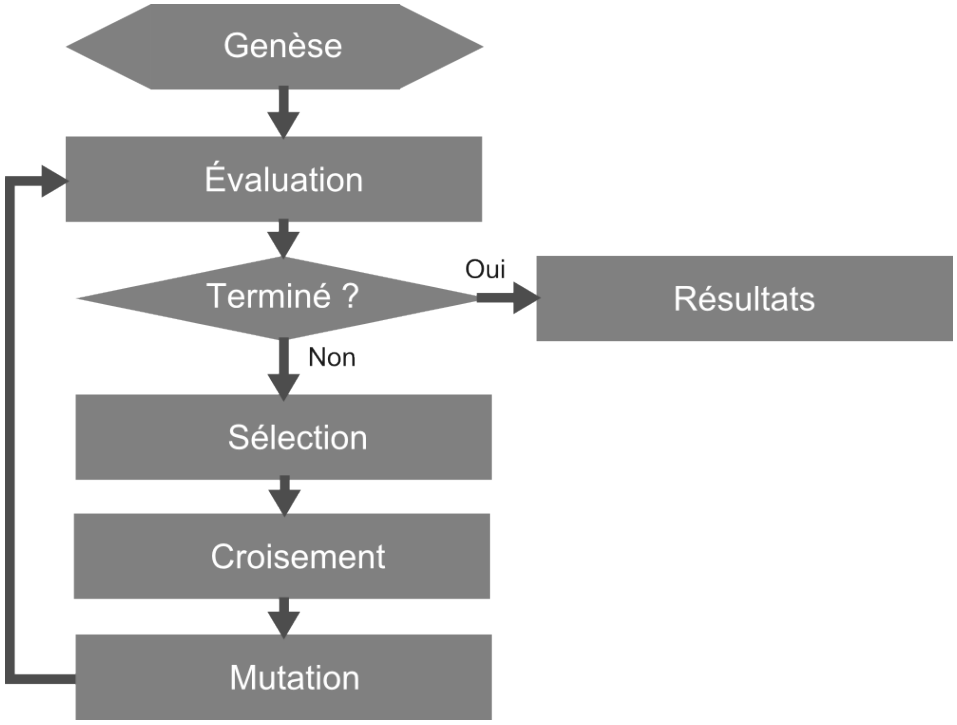


FIGURE 19 – Algorithme génétique

5.2.3 Algorithme génétique dans notre cas

Dans notre cas, on peut traiter la table d'affectation de toutes les personnes comme un individu génétique, donc une population est l'ensemble d'affectation, on mets le cardinal de l'ensemble de 100 et un gène est un cours. Pour évaluer, nous créons une fonction de coût comme niveau d'adaptation.

Dans notre cas, on veut privilégier une affectation où un étudiant ne sera pas trop privilégié. Autrement dit, on préfère que deux étudiants obtiennent leur second choix plutôt que l'un des étudiants obtienne sont premier choix et le second son troisième choix. On décide donc que le coût d'affectation d'un étudiant à un module d'un parcours sera :

— $(i-1)^2$ si l'étudiant obtient son i-ème choix
— $10 \times Nc^2$ si l'étudiant est affecté à un module qu'il n'avait pas choisi
— $Nc = 8$ pour le module préférentiel, et $Nc = 4$ pour l'autre module
— Le coût total d'affectation d'un étudiant sera la somme de tous les coûts des 6 affectations calculées
— Si un cours est choisi par moins de 5 ou plus de 10 personnes, nous attribuons un coût important à cette situation, on mets 1000 dans notre cas.

Pour nous, nous avons les quatre opérations correspondantes ci-dessus.

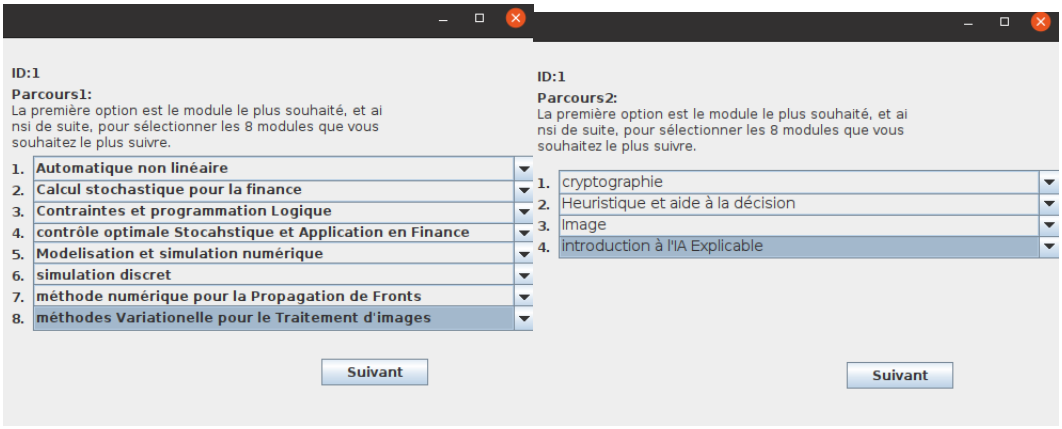
- évaluation du niveau d'adaptation d'un individu : calculer coût pour chaque l'affectation(individu)
- la sélection : sélection des meilleurs n individus qui ont des coûts minimes
- croisement ou recombinaison : échanger certains cours de deux affectations aléatoirement dans une certaine mesure, pour nous on choisi taux de croisement $= 0.4$
- mutation : changer aléatoirement certaines cours dans une certaine mesure, pour nous on choisi taux de mutation $= 0.1$

Nous choisissons d'itérer ce processus 100 fois, c'est-à-dire que nous avons 100 générations. Ce nombre d'itérations peut nous fournir un meilleur résultat, mais bien sûr nous ne pouvons pas prouver qu'il s'agit de la solution optimale globale.

6 Résultat

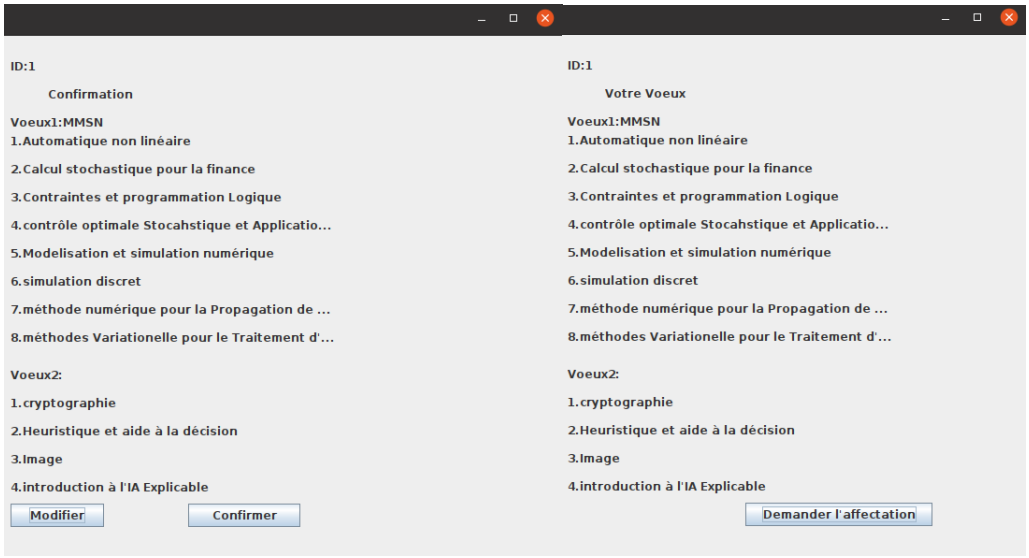
Tout d'abord, montrons l'interface vue par le client.

(a) Fenêtre 1 (b) Fenêtre 2



(c) Fenêtre 3 (d) Fenêtre 4

FIGURE 20 – Interface vue par client



(a) Fenêtre 5 (b) Fenêtre 6

(c) Fenêtre 7

FIGURE 21 – Interface vue par client

Ces sept fenêtres correspondent à notre Diagramme de maquette. En particulier, lorsque le bouton Modifier est cliqué dans la cinquième image, il reviendra à la deuxième fenêtre et l'utilisateur pourra resélectionner. On voit que lorsqu'il n'y a qu'une seule personne, les cours qui lui sont assignés sont exactement conformes à ses souhaits.

De plus, lors de la sélection de deux cours identiques, le système donnera également une invite.

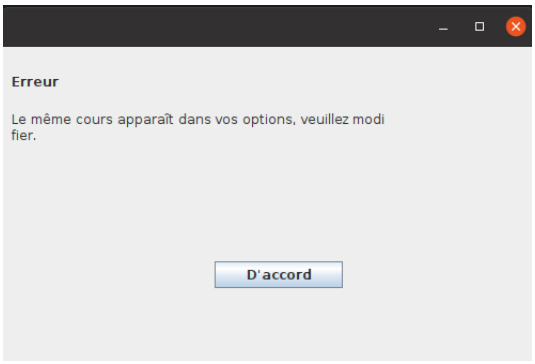


FIGURE 22 – Deux cours identiques

Dans le test, nous avons conçu deux parcours, pour MMSN et INFO. Il y a 10 cours dans chaque parcours, comme indiqué dans le tableau ci-dessous.

MMSN	INFO
Automatique non linéaire	cryptographie
Calcul stochastique pour la finance	Heuristique et aide à la décision
Contraintes et programmation Logique	Image
contrôle optimale Stocahstique et Application en Finance	introduction à l'IA Explicable
Modelisation et simulation numérique	approximation de données
simulation discret	Introduction to Machine Learning
méthode numérique pour la Propagation de Fronts	introduction aux Métaheuritisque
méthodes Variationelle pour le Traitement d'images	Système Multi-Agents
Modelistation appliquée :perturbations et problèmes inverses	calcul parallèle
Equations de hamilton-Jacobi et application	Programmation Orientée-Objet Avancée

TABLE 1 – Modules pour tester

Nous savons que nous limitons le nombre de personnes par cours à 5 à 10. Par conséquent, lors de nos tests,on choisit 20 personnes, les 11 premières personnes ont choisi les mêmes cours, mais l'ordre de leurs souhaits était différent, il était donc garanti que plus de 10 personnes avaient choisi ces cours. Parmi les 9 dernières personnes, il y a plusieurs parcours qui ont été sélectionnés moins de 4 fois.

Tout d'abord, nous testons que si la 21e personne se connecte, elle ne pourra pas se connecter avec succès, toujours en attente.

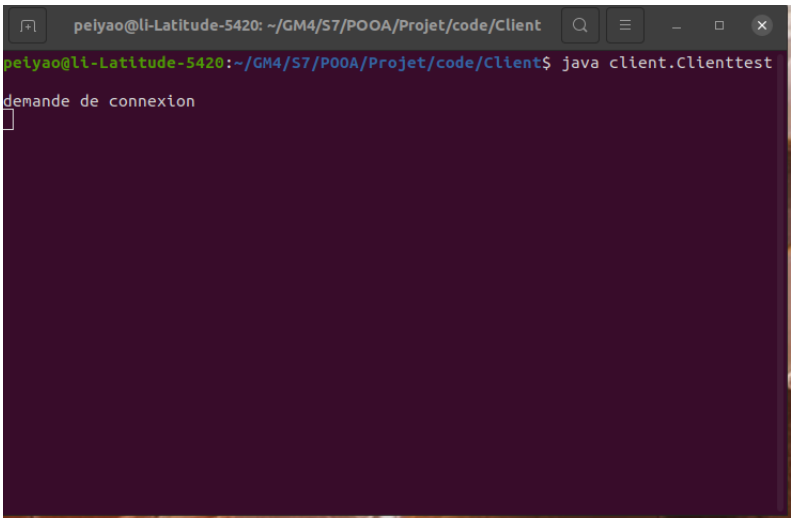


FIGURE 23 – 21e demande de connextion

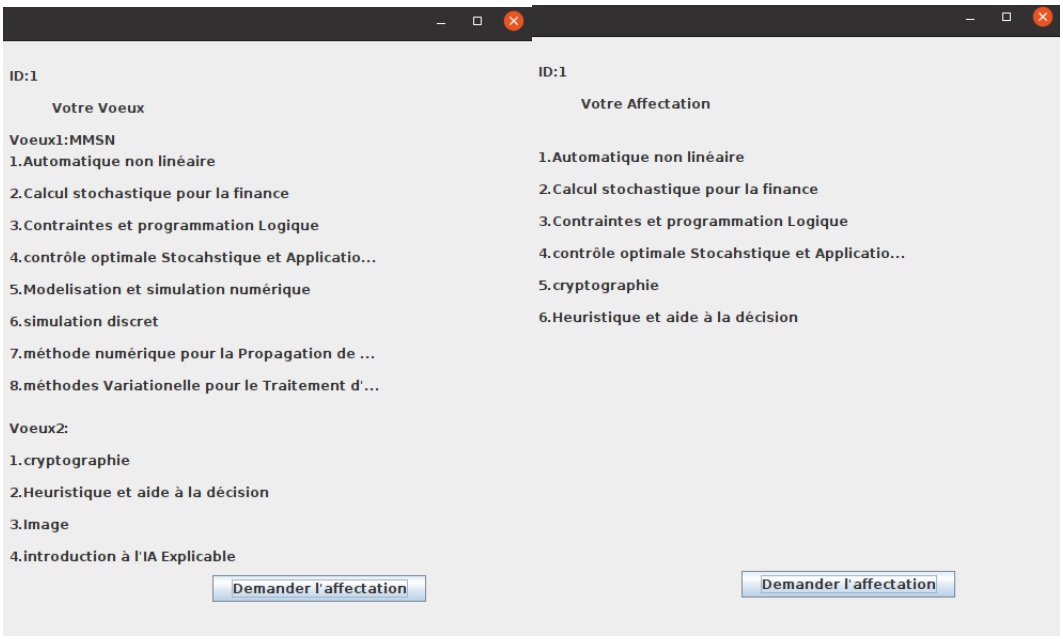
Le résultat de la répartition de 20 personnes, le nombre de personnes pour chaque cours est le suivant.

MMSN	Nb	INFO	Nb
Automatique non linéaire	9	cryptographie	6
Calcul stochastique pour la finance	7	Heuristique et aide à la décision	6
Contraintes et programmation Logique	10	Image	5
contrôle optimale Stocahstique et Application en Finance	8	introduction à l'IA Explicable	5
Modelisation et simulation numérique	5	approximation de données	5
simulation discret	5	Introduction to Machine Learning	5
méthode numérique pour la Propagation de Fronts	7	introduction aux Métaheuritisque	5
méthodes Variationelle pour le Traitement d'images	7	Système Multi-Agents	5
Modelistation appliquée :perturbations et problèmes inverses	5	calcul parallèle	5
Equations de hamilton-Jacobi et application	5	Programmation Orientée-Objet Avancée	5

TABLE 2 – Nombres de personnes dans l'affectation

Nous pouvons voir que tous les cours sont attribués à 5-10 personnes. Notamment, notre algorithme tient compte du fait que s'il y a moins de 5 personnes sélectionnées pour un cours, toutes les personnes qui choisissent ce cours seront ajustées à d'autres cours, et le cours sera annulé. Mais dans le cas de notre test, chaque cours a plusieurs personnes sélectionnées, même s'il n'atteint pas 5 personnes. Par conséquent, le résultat optimal est qu'aucun cours n'est annulé, et ceux qui choisissent d'autres cours sont ajustés au cours qui a été sélectionné moins de 5 fois, de sorte que le nombre de personnes dans ce cours atteigne 5, et le cours est conservé. C'est raisonnable.

Pour tester si les affectations sont raisonnables, nous sélectionnons au hasard quelques étudiants et examinons leurs souhaits et les résultats des affectations du cours final. Tout d'abord, nous observons l'étudiant 1, ses souhaits et les résultats d'affectation lorsqu'il est seul au début sont les suivants.



(a) Voeux Etudiant 1 (b) Affectation lorsqu'il est seul

FIGURE 24 – Etudiant 1 lorsqu'il est seul

Au final, ses résultats d'affectation sont les suivants.

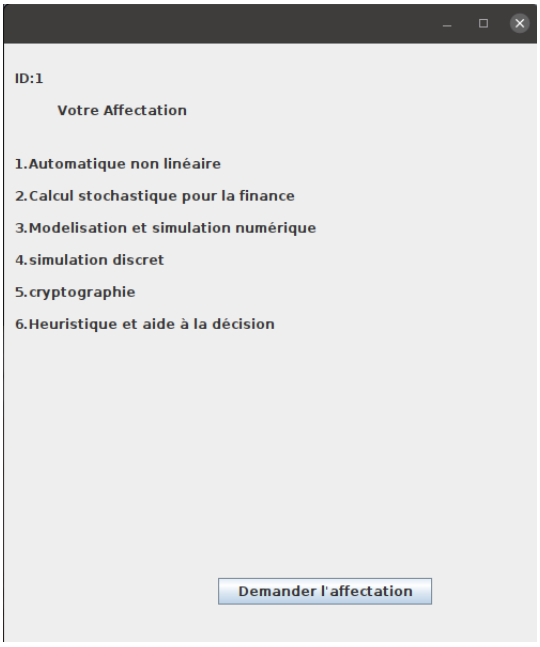
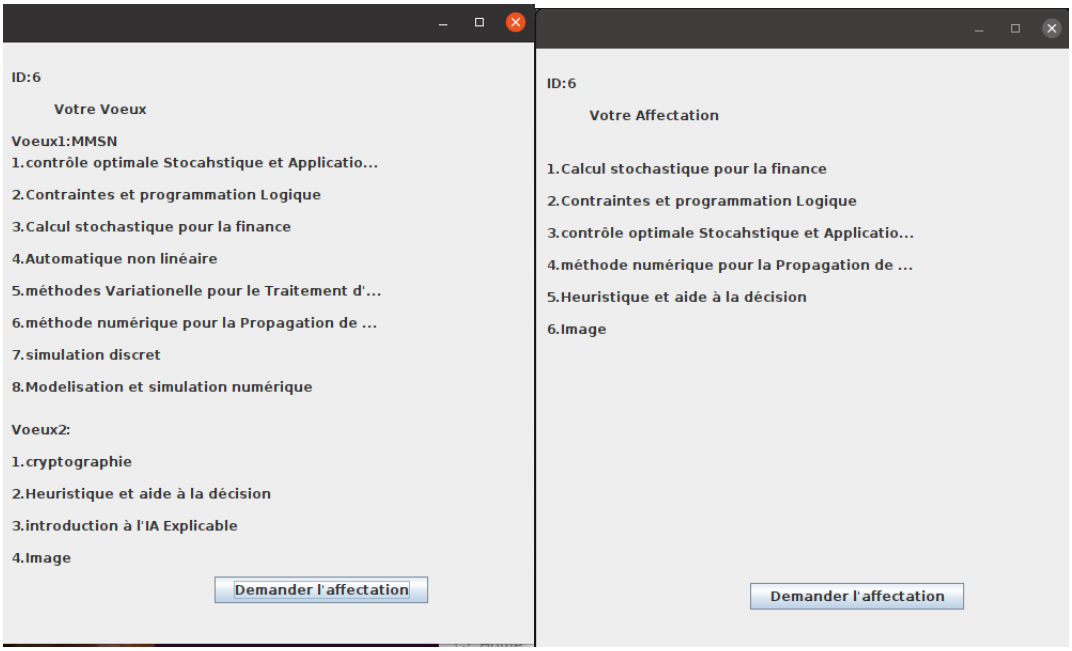


FIGURE 25 – Etudiant 1 Affectation final

On peut voir que les deux premiers souhaits de ses deux Parcours sont conservés, et son 5ème et 6ème voeux de parcours préférentiel sont choisit, car le cours est sélectionné plus de 10 fois. Ca a

du sens.

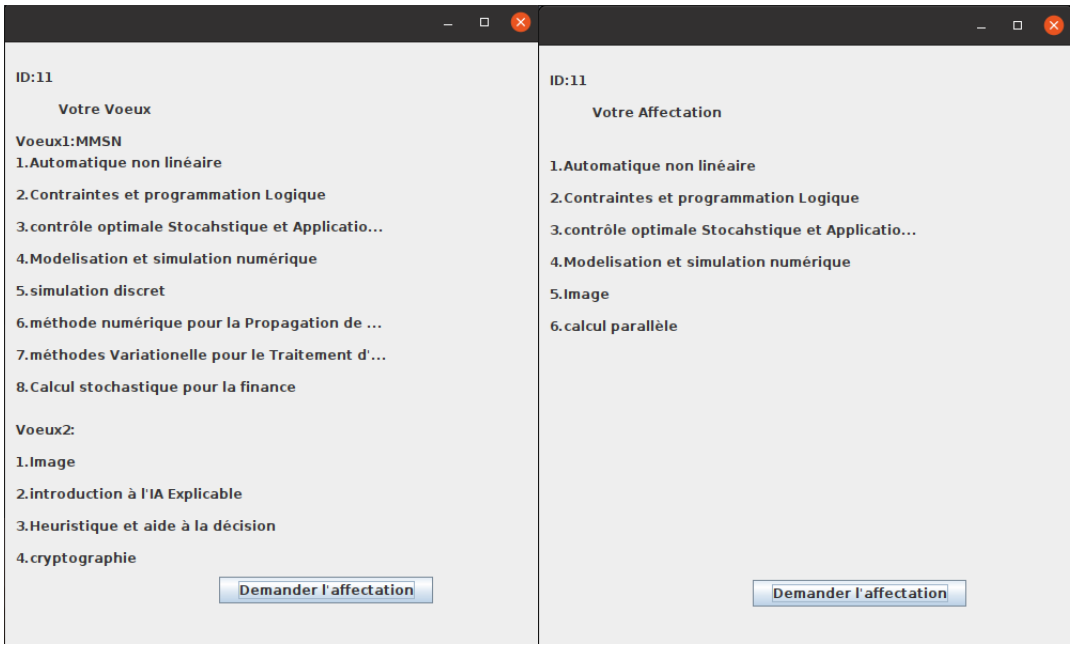
Faisons encore quelques comparaisons.



(a) Voeux Etudiant 6 (b) Affectation Etudiant 6

FIGURE 26 – Etudiant 6

Pour l'élève 6, les 1er, 2e, 3e, 6e souhaits de son parcours préférentiel ont été attribués, et les 2e, 4e souhaits d'un autre parcours ont été attribués, pas mal.



(a) Voeux Etudiant 11 (b) Affectation Etudiant 11

FIGURE 27 – Etudiant 11

Pour l'élève 11,les 1er, 2e, 3e, 4e souhaits de son parcours préférentiel ont été attribués, et le 1er souhaits d'un autre parcours ont été attribués, pas mal.

De manière générale, on constate que la plupart des devoirs correspondent fondamentalement aux choix des élèves. Étant donné que certains cours ont été délibérément sélectionnés au-delà de la limite de 5 à 10 lors de nos tests, il est raisonnable que certains étudiants n'aient pas été affectés aux cours sélectionnés. Nous avons également effectué plusieurs tests, et les résultats étaient tous bons.

7 Améliorations possibles

7.1 Ajouter plus de parcours et de modules

Lors de nos tests, nous avons choisi d'utiliser deux Parcours de 10 Modules dans chaque Parcours. En effet, côté Serveur, on peut choisir n'importe quel nombre de Parcours et créer n'importe quel nombre de Modules à distribuer, il suffit de changer quelques paramètres. Notre code est déjà réalisé cette situation, il suffit de créer d'autres parcours et modules directement pendant le test, tous les tableaux sur parcours s'agrandira automatiquement. Côté client, il ajoutera automatiquement le bouton de Parcours3.

7.2 Laissez plus de clients se connecter

Lors de nos tests, nous avons limité le nombre de connexions simultanées à 20 utilisateurs pour des affectations de cours de 20 utilisateurs. Bien sûr, nous pouvons augmenter le nombre de personnes, ou même ne pas limiter le nombre de personnes. Nous avons deux méthodes. Tout d'abord, nous pouvons augmenter le nombre de connexions simultanées. Dans notre cas nous avons créé 20 threads, chaque thread gère la connexion d'un utilisateur. Si nous augmentons le nombre de personnes, nous pouvons créer plus de threads.

Ensuite, on sait que la création de threads conduira à l'occupation de ressources. En fait, nous pouvons conserver la condition de restreindre 20 utilisateurs à se connecter en même temps, c'est-à-dire continuer à utiliser 20 threads, mais réaliser l'allocation de cours de plus d'utilisateurs. Plus précisément, nous pouvons créer un ThreadGroup avec 20 threads. Il surveillera automatiquement si le thread est en cours d'exécution. Si le thread ne s'exécute pas, c'est-à-dire qu'un utilisateur est déconnecté, nous laissons le thread s'exécuter à nouveau. Après le redémarrage, il peut enregistrer une nouvelle socket pour connecter un nouvel utilisateur.. Bien sûr, cette situation nous oblige à stocker les affectations de cours localement ou dans une base de données afin que lorsqu'un utilisateur déconnecté se reconnecte, ses affectations de cours puissent lui être envoyés.

7.3 Améliorer l'algorithme d'optimisation

Actuellement, lorsque nous utilisons l'algorithme génétique, laissons-le effectuer 100 itérations, soit 100 générations. Nous pouvons augmenter le nombre d'itérations pour améliorer l'optimisation. Mais son problème est qu'on ne peut jamais prouver que le résultat obtenu a atteint la solution optimale globale.

De plus, nous avons implémenté une amélioration de l'algorithme d'optimisation. Normalement, les mutations dans un algorithme génétique sont complètement aléatoires. Mais nous ajoutons des contraintes à la mutation. Nous restreignons qu'une mutation d'une cours doit passer à une autre cours au sein du même parcours que lui. Car nous savons que si le cours d'un élève d'un parcours sont affectés par le cours dans un autre parcours, le coût sera très élevé. Nous excluons donc manuellement certaines très mauvaises situations, puis calculons l'algorithme, afin que les résultats de l'algorithme puissent être plus précis.

7.4 Interface de suivi des résultats d'affectation

Côté Serveur, on peut ajouter une interface. Il est pratique pour les administrateurs de suivre en temps réel l'attribution des cours, les informations sur les étudiants, les informations sur les cours et les informations sur les parcours.Il peut également afficher le coût après chaque attribution, afin de voir si les résultats de l'affectation sont toujours bons.

8 Conclusion

Nous avons beaucoup gagné grâce à ce projet.

Dans un premier temps, nous avons collaboré à la mise en place d'une application complète. Nous avons d'abord analysé les exigences de ce projet. Ensuite, nous avons appris le framework MVC, qui est le framework de base pour implémenter une application. Sur la base de ce cadre et combiné aux exigences de notre projet, nous avons analysé les classes requises, les méthodes requises dans chaque classe, les connexions entre classes, les connexions entre packages et les connexions entre appareils. Puis, grâce aux connaissances acquises en classe, l'implémentation du code est complétée. Surtout la communication entre deux appareils, la communication utilisant l'architecture Client-Serveur est quelque chose que nous n'avons pas essayé. Nous avons mis en pratique ce que nous avons appris et ce fut une expérience pratique très enrichissante. Cela constituera une bonne base pour nos études et nos travaux futurs.

Nous avons aussi beaucoup gagné en travail de groupe. Nous avons expérimenté comment répartir le travail plus efficacement, résumer le travail et terminer le travail en groupe. Dans le travail réel de programmation, il y aura de nombreuses formes de coopération de groupe. Par conséquent, il sera très utile d'avoir une telle expérience de travail avec un groupe pour mettre en œuvre une application.

Enfin, nous avons également découvert un nouvel algorithme d'optimisation, l'algorithme génétique. Les algorithmes heuristiques sont quelque chose que nous n'avons pas touché auparavant. Nous voyons une nouvelle façon de mettre en œuvre des optimisations, ce qui est très intéressant. Dans l'utilisation de l'algorithme génétique, nous avons également apporté quelques améliorations en fonction de la situation réelle. Dans un algorithme génétique normal, le processus de mutation est complètement aléatoire. Mais dans notre cas, nous avons deux parcours avec différentes cours. Il est très mauvais d'attribuer cours dans l'autre parcours au Parcours sélectionné pour les étudiants. Par conséquent, lorsque nous mutons, nous limitons la mutation aux cours dans le même parcours que le cours, et éliminons manuellement de nombreuses très mauvaises situations, améliorant la précision et l'efficacité de la mise en œuvre de l'algorithme.

9 Bibliographie

Les PDFs de cours POOA
https ://docs.oracle.com/en/

10 Remerciement

Nous remercions pour les connaissances acquises dans le cours de POOA et pour l'ordinateur fourni par l'école.

Nous tenons particulièrement à remercier Mme.Zanni-Merk pour ses conseils sur notre direction de recherche et son aide technique.