IUT de Lille - Département informatique - N2
Université de Lille
2022-2023

SAÉ 2.01 & 2.02

Organisation de séjours linguistiques

L'objectif général de cette SAÉ est de développer un outil d'aide à la décision pour résoudre des problèmes trop complexes pour être résolus manuellement. Cet outil s'appuiera sur des algorithmes d'optimisation et une interface graphique utilisée pour visualiser les solutions proposées par l'algorithme d'optimisation. L'interface permettra également de modifier les paramètres de l'algorithme pour guider la solution.

Le besoin client

Des collèges organisent tous les ans des séjours linguistiques entre établissements scolaires de France, d'Italie, d'Espagne et d'Allemagne. Il s'agit d'envoyer ou de recevoir un groupe d'adolescents en partance ou en provenance d'un des autres pays. Chaque adolescent visiteur est logé dans la famille d'un adolescent hôte. La constitution de paires hôte–visiteur doit prendre en compte plusieurs critères détaillés dans la suite. On considérera des séjours et non des échanges : l'accueil d'un adolescent italien chez un adolescent français par exemple, n'implique pas nécessairement l'accueil réciproque. Si un tel séjour est organisé, il donnera lieu à une nouvelle affectation.

Certains des critères sont des contraintes rédhibitoires (par exemple, problèmes d'allergies) et interdisent la formation de certaines paires. D'autres critères expriment des préférences (par exemple, les intérêts de l'adolescent) et sont utilisés pour associer des adolescents ayant plus d'affinités. Plus l'affinité entre deux adolescents est grande, moins il y a de risque de rupture de paire hôte–visiteur durant un séjour linguistique. Vous devez développer un outil qui permet aux enseignants de constituer les paires hôte–visiteur de manière à satisfaire au mieux les différents critères.

Voici le fonctionnement préconisé. Tous les participants à un séjour linguistique (hôtes et visiteurs) remplissent un questionnaire en ligne pour renseigner l'ensemble des informations utiles. Les données ainsi collectées sont disponibles sous forme de fichiers CSV, sur lesquels votre application est basée.

L'outil doit fournir ces fonctionnalités :

— *Charger un fichier CSV* suivant un format bien précis et qui contient les données d'entrée (adolescents et leurs contraintes).

— *Filtrer les données* pour éliminer les éventuelles incohérences.

— *Calculer un appariement* qui satisfait aux mieux les critères exprimés.

— *Afficher l'appariement* trouvé.

— *Afficher les contraintes rédhibitoires non satisfaites.* S'il est impossible de satisfaire toutes les contraintes, l'outil doit permettre d'afficher les adolescents pour qui on n'a pas trouvé de correspondant.

— *Prendre en compte un historique.* Plusieurs séjours peuvent avoir lieu entre les mêmes pays de provenance et destination différentes années. Il arrive qu'un séjour ne se passe pas bien pour un participant, dans ce cas on veut éviter de l'apparier avec le même adolescent lors d'un prochain séjour. L'outil doit permettre de charger les précédents appariements afin de les prendre en compte. Même s'il faut conserver l'historique des appariements, les séjours sont gérés par année : on ne planifie pas les affectations sur plusieurs années.

Règles d'appariement

Les adolescents peuvent exprimer des contraintes, des préférences, ou bien des indications sur leur capacité à satisfaire les contraintes d'un correspondant potentiel. Dans la suite, nous appelons toutes ces informations des *critères*. Voici la liste exhaustive de critères qui seront considérés :

— `GUEST_ANIMAL_ALLERGY` décrit le fait d'être ou non allergique à un animal. Les valeurs possibles sont *yes* et *no* ;

Version du sujet : 7 avril 2023

— `HOST_HAS_ANIMAL` informe sur la présence ou non d'un animal allergène dans son domicile. Les valeurs possibles sont *yes* et *no* ;

— `GUEST_FOOD` énumère les spécificités du régime alimentaire (comme `vegetarian` or `nonuts` par exemple). L'absence de valeur signifie qu'aucune contrainte n'existe ;

— `HOST_FOOD` liste les régimes que l'on accepte de servir. L'absence de valeur signifie qu'on n'accepte aucune contrainte ;

— `HOBBIES` représente la liste des passe-temps les plus appréciés ;

— `GENDER` décrit le genre de l'adolescent (*male*, *female* ou *other* seront les seules valeurs considérées) ;

— `PAIR_GENDER` exprime une préférence quant au genre de l'éventuel correspondant (sur les mêmes valeurs). L'absence de valeur signifie qu'il n'y a aucune préférence ;

— `HISTORY` décrit la manière de prendre en compte l'historique. Une absence de valeur représente l'indifférence au passé, *same* représente la préférence de conservation du même correspondant que la dernière fois et *other* représente l'obligation de changer de correspondant.

Vous trouverez en annexe, Section A, des exemples de critères exprimés et leur représentation dans un fichier CSV. **Notez que l'ordre des colonnes n'est pas fixé dans le fichier CSV ; c'est la première ligne qui détermine de quel critère il s'agit.**

L'appariement d'adolescents sera établi en prenant en compte des contraintes (comme les allergies) et des affinités (déterminées par les préférences exprimées, comme les passe-temps). On dira qu'un hôte et un visiteur sont *compatibles* s'ils respectent les contraintes et l'appariement ne doit pas associer des adolescents non compatibles. Voici les règles qui déterminent la compatibilité et l'affinité entre un hôte et un visiteur :

— **Allergie aux animaux** (contrainte).
Les deux adolescents sont compatibles lorsque :
  — le visiteur n'a pas d'allergie aux animaux (valeur *no* pour le critère `GUEST_ANIMAL_ALLERGY`), ou bien
  — le visiteur a une allergie aux animaux (valeur *yes* pour le critère `GUEST_ANIMAL_ALLERGY`) et l'hôte n'a pas d'animal (valeur *no* pour le critère `HOST_HAS_ANIMAL`).

— **Régime alimentaire** (contrainte).
Les deux adolescents sont compatibles seulement si l'ensemble des régimes requis par le visiteur est inclus dans l'ensemble des régimes acceptés par l'hôte.

— **Historique** (contrainte ou préférence).
Lorsque les deux adolescents étaient correspondants l'année précédente :
  — si au moins un des deux a exprimé la contrainte *other*, les deux sont considérés comme incompatibles (c'est une contrainte rédhibitoire),
  — si les deux ont exprimé la contrainte *same*, alors ils **doivent** à nouveau être correspondants (c'est une contrainte forte qui doit être respectée),
  — si l'un des deux a exprimé la contrainte *same* et l'autre n'a exprimé aucune contrainte, alors on applique un bonus d'affinité.

— **Passe-temps** (préférence).
Plus les adolescents ont de passe-temps en commun, plus leur affinité est importante.

— **Genre** (préférence).
On considère qu'il y a affinité pour chaque préférence de genre satisfaite (c'est-à-dire, il y a un bonus d'affinité important si les préférences de genre des deux adolescents sont satisfaites. Ce bonus d'affinité est moindre si une seule des préférences de genre est satisfaite). Notons qu'une préférence non exprimée (une valeur vide pour `PAIR_GENDER`) est considérée comme étant toujours satisfaite.

— **Différence d'âge** (préférence).
Une différence d'âge inférieure à 1 an et demi apporte un bonus d'affinité.

Un appariement ne doit pas associer des adolescents incompatibles, sauf s'il est impossible de satisfaire toutes les contraintes exprimées. Lorsqu'il est possible de satisfaire toutes les contraintes, un appariement est meilleur qu'un autre s'il associe plus d'adolescents ayant plus d'affinités.

Détails des consignes

[POO] Programmation orientée objet (R2.01/R2.03)

Les adolescents sont identifiés par un nom, un prénom, un genre, une date de naissance et un pays d'origine. Les critères décrits ci-dessus seront stockés via une table associative (de type `Map`).

Chacun des critères présentés dans la section précédente est caractérisé par un nom et un type (représenté par un caractère : `'B'` pour booléen, `'T'` pour du texte, `'N'` pour une valeur numérique, `'D'` pour une date). Un nom de critère sera toujours associé au même type de valeur. Les types des valeurs sont les suivants : `GUEST_ANIMAL_ALLERGY` et `HOST_HAS_ANIMAL` sont de type booléen, les autres sont de type textuel. Pour des raisons de simplicité de modélisation, les valeurs des critères seront toujours stockées sous forme de chaîne de caractères. Ainsi, un critère est donc donné par un nom (provenant de la liste prédéfinie) et une valeur (de type chaîne de caractères), qu'il faudra éventuellement transformer ultérieurement.

La modélisation de base doit **impérativement** respecter la conception décrite Figure 1. Bien entendu, il est nécessaire de compléter et d'étendre cette base de conception pour y intégrer intelligemment les fonctionnalités plus avancées.

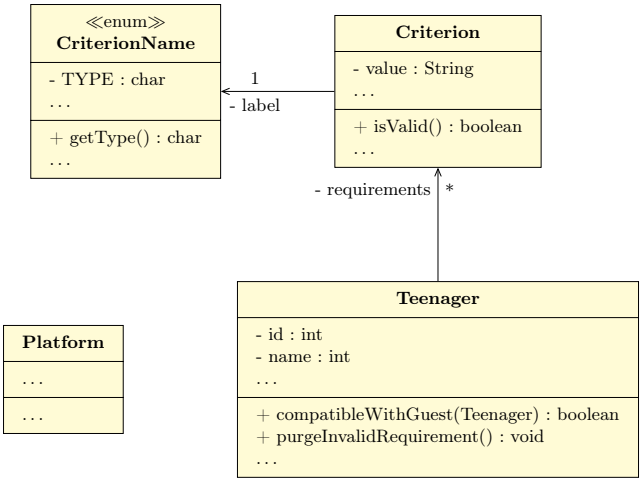

FIGURE 1 – Conception partielle de base à respecter

La plateforme d'affectation regroupe un ensemble d'adolescents des différents pays. Le but de cette plateforme est de produire des affectations entre tous les adolescents de deux pays donnés. Un certain nombre de fonctionnalités seront nécessaires afin que la plateforme puisse faire face aux différentes situations pouvant survenir. La liste de ces fonctionnalités est reprise dans le planning proposé Section 1 pour respecter la progression pédagogique des ressources et faciliter leur implémentation.

— **La vérification de la validité des critères exprimés**.
La valeur de tous les critères est stockée sous forme de chaîne de caractères. Or, ils correspondent tantôt à des booléens, tantôt à des valeurs numériques (selon le type de critère pris en compte). Il faut être en mesure de vérifier que les critères, récupérés à partir d'un fichier CSV issu d'un formulaire, sont cohérents avec le type prévu. Les critères non valides devront être supprimés le cas échéant.

— **La sélection des adolescents à retirer**.
Il arrive que des formulaires soient mal remplis par exemple ou que le nombre de places disponibles soit limité (quel que soit le pays d'origine). Dans ce cas, on peut souhaiter enlever un nombre donné d'adolescents du programme de séjours linguistiques (d'un pays donné ou de manière générale). Par défaut, on choisit d'enlever en premier lieu les adolescents ayant le plus grand nombre de contraintes incohérentes. On se concentrera sur deux types d'incohérences suivants : un critère censé stocker un booléen mais de valeur effective d'un autre type est incohérent, ou un adolescent déclarant une allergie aux animaux (contrainte rédhibitoire) mais déclarant aussi posséder un animal à la maison.

— **L'implémentation de règles de compatibilité propres à certaines nationalités**.
Il est bien connu que les Français sont râleurs :-). On a observé lors des jumelages impliquant la

France un taux de rupture plus élevé que lors de jumelages entre autres pays. Pour éviter cela, on souhaite modifier la règle de compatibilité lors de jumelage impliquant des Français. Pour être compatible avec une Française ou un Français, il faut avoir au moins un passe-temps en commun, sinon la probabilité d'une rupture est trop grande.

— **La gestion des imports/exports**.
Dans un premier temps, les données vous seront fournies directement dans une structure statique, le temps de voir en cours la manipulation des fichiers. On souhaite cependant pouvoir importer des données au format CSV pour remplir ou compléter l'ensemble des adolescents répertoriés. De même, on souhaite pouvoir exporter au format CSV le résultat des affectations.

— **La gestion de l'historique des affectations**.
On souhaite pouvoir sauvegarder les affectations qui ont eu lieu selon les années et les pays impliqués. Les séjours entre deux mêmes pays sont annuels (et ne peuvent avoir lieu qu'une fois par an), mais un pays peut tout de même faire plusieurs séjours par an avec d'autres. Cet historique est à réaliser par sérialisation binaire.

— **La prise en compte de l'historique dans les affectations**.
Les adolescents peuvent demander à être avec le même correspondant que la dernière fois ou à l'inverse, ne pas être affectés avec la même personne. Dans un premier temps, on pourra ne prendre en compte que le dernier séjour linguistique en date.

— **Le traitement des données chargées**.
On veut pouvoir lancer la vérification de la validité des contraintes ainsi que la désinscription de certains adolescents, soit à la demande, soit de manière automatique dès lors qu'un fichier de configuration contenant les informations nécessaires existe à un emplacement prédéfini.

[GRAPHE] Graphe (R2.07)

L'appariement sera calculé grâce à une *modélisation par le problème d'affectation dans les graphes* de la manière suivante :

1. À partir d'un ensemble d'adolescents hôtes et un ensemble d'adolescents visiteurs, construire un graphe valué dont les sommets sont des adolescents et le poids de chaque arête traduit la compatibilité et l'affinité entre les deux adolescents à ses extrémités.
2. Exécuter l'algorithme d'affectation sur ce graphe.
3. Depuis le résultat de l'algorithme, déduire un appariement entre adolescents.

Nous mettons à votre disposition une librairie qui permet de :

— construire un graphe en donnant ses sommets et ses arêtes,

— exécuter l'algorithme d'affectation sur ce graphe et récupérer le résultat.

Ainsi, la partie la plus importante de votre travail (du point de vue de la ressource [GRAPHE]) sera la *modélisation*, et plus particulièrement *le choix des poids du graphe* pour représenter de manière adéquate la compatibilité et l'affinité entre adolescents. Des séances encadrées par l'équipe [GRAPHE] vous permettront de vous approprier la modélisation par le problème d'affectation et les librairies fournies. La modélisation proposée devra être décrite dans un **rapport**, **implémentée** et **testée**.

Concernant l'implémentation, vous devez créer une classe `AffectationUtil` qui permettra de calculer les poids des arêtes du graphe modèle.

```
public class AffectationUtil {
  /** Calcule le poids de l'arête entre host et visitor dans le graphe modèle.
   *  Doit faire appel à la méthode compatibleWithGuest(Teenager) de Teenager.
   *  Peut avoir d'autres paramètres si nécessaire.
   */
  public static double weight (Teenager host, Teenager visitor, ...) {
    ...
  }
  // ... ajouter toutes autres méthodes jugées nécessaires
}
```

Vous utiliserez la méthode `weight(...)` dans la partie de votre code qui s'occupe de créer le graphe. Les attentes précises pour le rapport et les classes de tests sont détaillées dans un document récupérable sur Moodle.

Vous allez procéder par étapes, en commençant par un modèle partiel et en ajoutant de plus en plus de critères pour arriver à un modèle complet.

Version 1 : allergie aux animaux et passe-temps Votre modélisation basée sur les graphes doit prendre en compte uniquement l'allergie aux animaux comme contrainte de compatibilité et les passe-temps (*hobbies*) comme critère d'affinité. En d'autres termes, la méthode `compatibleWithGuest(...)` de la classe `Teenager` sera basée sur l'allergie aux animaux. La méthode `weight(...)` de la classe `AffectationUtil` doit faire appel à `compatibleWithGuest(...)` et calculer un score basé sur l'(in)compatibilité et sur l'affinité due aux passe-temps.

Exemple 1 *Considérons en entrée les données décrites Figure 2 récoltées pour préparer un séjour français-italiens : certaines cases sont volontairement laissées vides car non pertinentes dans le cadre de cet exemple. D'après ces données, Bellatrix et Yak ne sont pas compatibles. On peut voir que l'affectation optimale est Bellatrix–Xolag, Adonia–Zander, et Callista–Yak.*

```
FORENAME;  NAME; COUNTRY; BIRTH_ GUEST_   HOST_   GUEST_      HOST_ HOBBIES;            GEN     PAIR_   HIS
                          DATE;  ANIMAL_  HAS_    FOOD_       FOOD;                    DER;    GEN     TO
                                 ALLERGY; ANIMAL; CONS                                      DER;    RY
                                                  TRAINT;
Adonia;    A;   FRANCE;  ;      no;      ;       ;           ;     sports,technology; ;       ;
Bellatrix; B;   FRANCE;  ;      yes;     ;       ;           ;     culture,science;   ;       ;
Callista;  C;   FRANCE;  ;      no;      ;       ;           ;     science,reading;   ;       ;
Xolag;     X;   ITALY;   ;      ;        no;     ;           ;     culture,technology; ;      ;
Yak;       Y;   ITALY;   ;      ;        yes;    ;           ;     science,reading;   ;       ;
Zander;    Z;   ITALY;   ;      ;        no;     ;           ;     technology;        ;       ;
```

FIGURE 2 – Exemple de données (partielles) au format CSV. Les données sont alignées et présentées avec des césures pour une meilleure lisibilité.

Version 2 : historique, degré d'incompatibilité, autres critères Le premier objectif de cette version est de prendre en compte les contraintes et préférences liées à l'historique. Pour cela, il vous faut *adapter la modélisation* proposée pour la Version 1, en particulier la construction du graphe et la méthode `weight(...)` de la classe `AffectationUtil`. Notez que pour la ressource [GRAPHE], il n'est pas nécessaire de charger l'historique depuis un fichier. Une représentation de l'historique suffit, par exemple sous la forme d'une collection de paires d'adolescents. Vous aurez sans doute besoin d'ajouter des paramètres à la méthode `weight(...)` permettant de prendre en compte l'historique.

Une fois que l'historique est bien géré, vous pouvez :

— modifier la fonction `weight(...)` pour ajouter la prise en compte des autres critères d'affinité : la préférence quant au genre du correspondant et la différence d'âge ;

— introduire un degré d'incompatibilité. Il s'agit de considérer que l'incompatibilité n'est pas binaire (incompatible vs compatible), mais que les contraintes non satisfaites apportent un malus qui peut être plus ou moins important et qu'il est possible de compenser grâce aux bonus d'affinité. On peut par exemple considérer que le non respect d'un seul régime alimentaire est moins grave que l'accumulation d'une allergie aux animaux et de plusieurs régimes alimentaires non satisfaits. Ou bien, une allergie aux animaux pourrait ne pas interdire l'appariement de deux adolescents qui ont par ailleurs une très bonne affinité. Ainsi vous devrez proposer une implémentation de la fonction `weight(...)` qui ne fait pas appel à la fonction `compatibleWithGuest(...)` de la classe `Teenager` mais traite les contraintes et les préférences de manière similaire.

[IHM] IHM (R2.02)

L'interface de l'application utilisera un cahier des charges plus restreint que celui présenté au début du sujet. Elle se concentrera sur les adolescents de deux pays et elle devra permettre de réaliser les opérations suivantes :

1. visualiser l'ensemble des appariements,
2. visualiser les détails sur un appariement (noms, prénoms des adolescents et informations associées),
3. visualiser les contraintes rédhibitoires qui ne sont pas respectées,
4. ajuster les pondérations des différents critères,
5. fixer au préalable des affectations,
6. éviter d'affecter des adolescents.

L'application sera développée en utilisant JavaFX. Il doit par exemple être possible de gérer la situation suivante : un adolescent s'est désisté et il faut trouver un nouveau correspondant qui réponde le mieux aux critères, en conservant les adolescents déjà affectés.

1 Organisation du travail

1.1 Gestion des équipes

Le travail est à réaliser en trinôme. Ceux-ci sont fixés par les étudiants eux-mêmes au début du projet de manière définitive, aucune modification ne sera alors possible, même en cas d'abandon d'un des membres. Les membres restants continueront le travail seuls, mais cela sera pris en compte lors de l'évaluation. Aucun trinôme intergroupe ne sera accepté, ni aucun groupe de plus de trois étudiants.

Des dépôts `Git`, sur le `https://gitlab.univ-lille.fr` de l'université ont été mis en place par les responsables. Vous devez être affectés à un de ces projets git par un des enseignants qui encadrent les SAÉs. La visibilité du dépôt (*repository*) doit rester privée pour éviter les problèmes de plagiat.

1.2 Évaluations et calendrier

Le travail est décomposé en parties distinctes selon les ressources impliquées. Des rendus sont attendus pour chacune des parties selon un calendrier précis. Tout manquement au respect du calendrier ou des consignes sera sanctionné lors de l'évaluation. Des *commits* étiquetés (munis de *tags* spécifiques) sont attendus.

La durée totale du projet est de 8 semaines. Pour vous aider dans la planification de votre travail, voici une liste des tâches à effectuer par période de deux semaines. Il est fortement recommandé de respecter ce planning qui prend en compte la progression pédagogique des ressources impliquées !

Semaines 1 (du 10/04) et 2 (du 1/05)

- [POO]classes de base (`CriterionName`, `Country`, `Criterion`, `Teenager`, `Platform`...)
- [POO]compatibilité entre adolescents (contraintes)
- [POO]développement des classes de tests correspondantes, illustrant le bon fonctionnement des principales méthodes de ces classes

⇒ commits étiquetés `POO-v1` à faire avant le 6/05/2023.

Semaines 3 (du 8/05) et 4 (du 15/05)

- [POO]gestion de la validité des critères par un mécanisme d'exception
- [POO]développement des règles spécifiques de compatibilité pour certains pays
- [GRAPHE]développement des règles de calcul d'affinité et d'affection pour la Version 1
- [GRAPHE]implémentation d'une fonction de test montrant le bon fonctionnement du système d'affectation

⇒ commits étiquetés `POO-v2 GRAPHE-v1` à faire avant le 20/05/2023

Semaines 5 (du 22/05) et 6 (du 29/05)

- [POO]import par fichier de format csv, répondant à la structure donnée
- [POO]export d'un résultat sous format csv
- [POO]gestion de l'historique par sérialisation binaire
- [GRAPHE]développement d'une nouvelle version des règles de calcul d'affinité pour la Version 2
- [GRAPHE][POO]implémentation d'un jeu de données montant le bon fonctionnement du nouveau système d'affectation
- [IHM]maquettage et prototype basse fidélité

⇒ commits étiquetés `POO-v3 GRAPHE-v2 IHM-v1`, à faire au plus tard une journée avant la séance suivante de SAÉ (la semaine du 29/05/2023).

Semaines 7 (du 5/06) et 8 (du 12/06)

- [POO]prétraitement des critères des adolescents sur demande
- [POO]prétraitement automatique des critères en cas d'existence d'un fichier de configuration à un emplacement prédéfini
- [IHM]prototype haute fidélité
- [IHM]développement en JavaFX

⇒ commits étiquetés `POO-v4 IHM-v2` à rendre la semaine du 12 juin, à une date qui sera précisée ultérieurement.

2 Rendus attendus

2.1 Partie I : modélisation UML et implémentation

De nombreux éléments et fonctionnalités ont été décrits en préambule. Il convient donc de réfléchir à la manière la plus efficace d'organiser le code, afin de limiter la redondance, de faciliter la maintenance ou l'évolution des fonctionnalités, etc. Pour cela, un diagramme UML prenant l'ensemble des éléments en compte doit être établi a priori, afin de ne pas devoir modifier les classes déjà écrites à chaque nouvelle fonctionnalité. Vous êtes libres de proposer de nouvelles fonctionnalités qui vous semblent pertinentes dès lors que celles requises sont implémentées.

Lors de l'évaluation de votre projet, différents aspects seront pris en compte comme la qualité du code produit, la couverture de votre code par les tests que vous aurez fournis, le respect du cahier des charges et des fonctionnalités demandées, la qualité des tests (les scénarios fournis, couvrent-ils bien l'ensemble des fonctionnalités demandées), la qualité du rapport et de la réflexion qui y est détaillée, l'utilisation des outils de gestion de projet (git via le nombre, la qualité et la régularité des *commit*s)...

Vous devez rendre sous Moodle un rapport au format `PDF` décrivant successivement :

— la manière de lancer et utiliser votre application (position des ressources nécessaires, commande de lancement et ses options éventuelles...)

— un diagramme UML de vos classes, accompagné d'une réflexion sur les mécanismes objets vus en cours que vous avez mis en œuvre et leurs intérêts le cas échéant

— une analyse technique et critique de l'implémentation des fonctionnalités demandées (cf Section)

— une analyse quantitative/qualitative des tests que vous avez réalisés, en rapport aux notions vues en cours

Une archive `JAR` de votre application fonctionnant **sans interface graphique** doit être disponible à la racine de votre dépôt Gitlab.

2.2 Partie II : modélisation par les graphes

L'évaluation va se baser sur un rapport écrit et certaines parties du code. Le rapport doit être au format pdf ou html et sera rendu sur Moodle. L'évaluation du code se limitera à la classe `AffectationUtil` décrite plus haut, et aux deux classes de test `TestAffectationVersion1` et `TestAffectationVersion2`.

Nous fournissons sur Moodle une trame qui contient le plan du rapport, en précisant les parties qu'il doit contenir et le contenu de chaque partie. Ce document décrit également ce qui est attendu pour les classes de test.

2.3 Partie III : IHM

Pour la partie [IHM], vous devrez rendre une archive **ZIP** contenant :

— un jar exécutable. Suivez les instructions disponibles dans le TP6 pour créer votre jar exécutable et utilisez JavaFX version 17.0.2 pour la compilation de votre code. Il n'est pas nécessaire d'ajouter les librairies propres au système à côté de votre jar exécutable.

— un export au format `HTML` ou PDF de vos *mockups* placés dans un répertoire `mockups`,

— un compte rendu **au format** `pdf` contenant :

  — vos noms et prénoms et numéro du groupe (par exemple B1),

  — lien vers le repo GitLab et référence du commit correspondant au rendu,

  — une capture d'écran de l'application finale,

  — une partie sur la justification de vos choix de conception au regard des critères ergonomiques, guide de conception...(cela servira notamment à évaluer la compétence 5 "identifier les besoins métiers des clients et des utilisateurs" de R2.02),

  — une partie qui détaille les contributions de chaque membre du groupe. Comment vous avez réussi à exploiter au mieux les compétences de chacun ? (cela servira notamment à évaluer la compétence 6 "identifier ses aptitudes pour travailler dans une équipe" de R2.02),

  — toute autre information utile permettant de mettre en valeur votre travail.

— Une vidéo de présentation de votre projet, de 2-3 minutes, conforme aux exigences disponibles ici. L'objectif est de présenter l'ensemble de votre réalisation, à destination de personnes qui n'ont aucune connaissance du projet (par exemple des recruteurs).

3 Matériel à disposition

Les ressources suivantes sont à votre disposition sur pour mener à bien la SAÉ :

— les bibliothèques nécessaires à la partie "*graphe*" sous la forme d'archives `jar`

Elles sont accessibles sur `https://gitlab.univ-lille.fr/sae2.01-2.02/common-tools`.

4 Portfolio

Ces documents vous seront utiles pour la constitution de votre portfolio, vous devez les conserver :

— le sujet de la SAÉ,

— les rapports rendus pour les parties I, II et III

— la vidéo de présentation du projet.

A Annexe : Représentation des données

Les données nécessaires à la formation des paires sont recueillies grâce au questionnaire de la Figure 3, rempli par tous les participants. Les réponses au questionnaire sont disponibles dans un fichier CSV.

Exemple 2 *La Figure 4 montre des données recueillies grâce au formulaire. Dans cet exemple, Adonia est allergique à un animal, Bellatrix ne l'est pas (colonne* ***`GUEST_ANIMAL_ALLERGY`****). Argob ne possède pas un animal et pourrait donc accueillir un visiteur allergique à animal, mais Bellatrix ne le pourrait pas car elle possède un animal (colonne* ***`HOST_HAS_ANIMAL`****). Khargol requiert un régime sans noix, alors qu'Adonia ne requiert aucun régime particulier (colonne* ***`GUEST_FOOD_CONSTRAINT`****). La famille de Khargol peut servic des repas sans noix et/ou végétariens, alors qu'Adonia ne fournira aucun régime alimentaire particulier (colonne* ***`HOST_FOOD`****). Bellatrix n'a aucune préférence quant au genre de son correspondant, Adonia préfère une correspondante femme (colonne* ***`PAIR_GENDER`****). Finalement, Bellatrix demande à changer de correspondant, Argob demande d'avoir le même correspondant pour le prochain séjour, alors que Khargol n'exprime aucune préférence sur ce sujet (colonne* ***`HISTORY`****).*

Formulaire d'inscription au programme de séjours linguistiques

Informations générales

Nom ..... Prénom ..... Date de naissance .....
Pays (choisir une seule réponse) ◯ ES ◯ FR ◯ IT ◯ GE

En tant que visiteur

1. Êtes-vous allergique à un animal ?
Choisir une seule réponse : ◯ Oui ◯ Non

2. Suivez-vous un régime alimentaire **strict** ?
Notez que donner une contrainte alimentaire pourrait nous empêcher de vous trouver un correspondant, donc ne l'utilisez que si c'est nécessaire, par exemple en cas d'allergie.
Zéro ou plusieurs réponses possibles : ☐ Sans noix ☐ Régime végétarien

En tant qu'hôte

3. Avez-vous un animal domestique allergisant à la maison (chat, chien, rongeur) ?
Choisir une seule réponse : ◯ Oui ◯ Non

4. Acceptez-vous d'adapter la cuisine familiale pour répondre aux besoins de votre visiteur, notamment en cas d'allergie ? Indiquer les régimes alimentaires que vous acceptez.
Zéro ou plusieurs réponses possibles : ☐ Sans noix ☐ Régime végétarien

Autres préférences ou contraintes

5. Quels sont vos passe-temps préférés ? .....

6. Quel est votre genre ?
Choisir une seule réponse : ◯ Femme ◯ Homme ◯ Autre

7. Avez vous une préférence pour le genre de votre invité·e ?
Choisir une seule réponse : ◯ Aucune préférence ◯ Femme ◯ Homme ◯ Autre

8. Pour le séjour à venir, voulez vous garder le même correspondant que la fois précédente, ou bien voulez-vous en changer ?
Choisir une seule réponse : ◯ Même correpondant ◯ Changer
◯ Aucune préférence ou ne s'applique pas

FIGURE 3 – Formulaire pour recueillir les données.

```
FORENAME;  NAME;    COUNTRY; BIRTH_      GUEST_   HOST_   GUEST_      HOST_              HOBBIES;                GENDER; PAIR_   HISTO
                            DATE;       ANIMAL_  HAS_    FOOD_       FOOD;                                              GENDER; RY
                                        ALLERGY; ANIMAL; CONSTRAINT;
Khargol;   Maalik;  GERMANY; 2008-01-22; no;      yes;    nonuts;     nonuts,vegetarian; culture,sports,reading; female; female;
Adonia;    Digby;   FRANCE;  2007-02-15; yes;     no;     ;           ;                  ;                       other;  female;
Bellatrix; Castel;  ITALY;   2007-11-02; no;      yes;    vegetarian; vegetarian;        science,reading;        female; ;       other
Argob;     Khijazy; SPAIN;   2007-07-07; no;      no;     ;           nonuts;            reading,technology;     male;   ;       same
```

FIGURE 4 – Exemple complet de données au format CSV. Les données sont alignées et présentées avec des césures pour plus de lisibilité.