**Documentation Technique – Projet de création de groupes d’étudiants**

1. **Présentation Générale**

**1.1. Objectif du projet**

Le projet consiste à développer une application web permettant à un professeur de créer automatiquement des groupes d’élèves en tenant compte des préférences exprimées par ceux-ci.

**1.2. Fonctionnalités principales**

* Authentification (étudiant / professeur)
* Gestion des élèves (ajout, suppression)
* Saisie des préférences d’équipe par les étudiants (pondération 1-100)
* Calcul automatique des groupes via un algorithme de satisfaction (CP-SAT)
* Affichage des groupes et du score global de satisfaction
* Interface de gestion pour le professeur (publication des résultats, ouverture/fermeture des choix)

1. **Architecture Système**

**box-cert-project/**

**│**

**│── data/**

**│── docs/**

**│── src/**

**│ │── auth/**

**│ │ └── app.py**

**│ │── static/**

**│ │ │── images/**

**│ │ │── auth.css**

**│ │ │── styles.css**

**│ │── templates/**

**│ │ │── auth/**

**│ │ │ │── signin.html**

**│ │ │ │── signup.html**

**│ │ │── index.html**

**│ │ │── student.html**

**│ │ │── teacher.html**

**│ │── algo.py**

**│── tests/**

**│ └── test\_algo.py**

**├── README.md**

**├── requirements.txt**

**└── to\_do\_list.xlsx**

**3. Choix Technologiques**

| **Composant** | **Technologie** | **Justification** |
| --- | --- | --- |
| Backend | **Flask** | Léger, Python natif |
| Templates | **Jinja2** | Intégration simple avec Flask, déjà expérimenté |
| Authentification | **SQLite** | Base légère, suffisante pour le projet |
| Stockage préf. | **JSON** | Simple à lire/modifier dynamiquement |
| Algorithme | **Google OR-Tools (CP-SAT)** | Résolution de contraintes efficace |
| Frontend | **HTML + CSS + JS** | Interface simple et dynamique |

**4. Interprétation du sujet**

* Le **professeur** est l’administrateur : il gère les utilisateurs, la saisie des préférences, le lancement de l’algorithme, et la publication des résultats.
* Les **étudiants** accèdent à l’application via login, saisissent leurs préférences (somme = 100), peuvent les modifier tant que la saisie est ouverte, et voient les résultats une fois publiés.

**5. Fonctionnement de l’algorithme**

**5.1. Objectif**

L’algorithme cherche à former des groupes d’étudiants de taille fixe (ici par défaut 4) en maximisant la somme des affinités mutuelles basées sur leurs préférences. En d’autres termes, il regroupe les étudiants de façon à maximiser la satisfaction globale des préférences exprimées entre eux.

**5.2. Données d’entrée**

**5.3. Variable**

* `x[i, g]` est une variable booléenne indiquant si l’étudiant `i` est dans le groupe `g`.

**5.4. Contraintes**

* Un élève ne peut être dans plusieurs groupes
* La somme des préférences saisies doit être égale à 100
* Les groupes doivent être équilibrés (taille définie par le prof

**5.5. Résolution**

L’algorithme est codé avec la bibliothèque **CP-SAT de Google OR-Tools** pour modéliser les contraintes et maximiser la satisfaction globale.

**5.6. Calcule du score de satisfaction**

* Chaque étudiant a des \*\*préférences pondérées\*\* (exemple : "Je préfère être avec X avec un poids 5").
* On crée une \*\*matrice de score `score[i][j]`\*\* qui donne l’affinité de l’étudiant `i` envers l’étudiant `j`.
* Le score mutuel entre deux étudiants `i` et `j` est `score[i][j] + score[j][i]`.

**5.6. Comment décide-t-il de mettre quelqu’un dans un groupe ?**

L’algorithme, via le solveur \*\*CP-SAT de Google OR-Tools

* Explore les affectations possibles des étudiants aux groupes sous les contraintes.
* Cherche l’affectation qui \*\*maximise la somme des scores d’affinités mutuelles.
* Il garantit qu’un étudiant n’est assigné qu’à un seul groupe.
* Il respecte la taille exacte de chaque groupe.

La décision est donc automatique et optimisée par le solveur, qui essaie toutes les combinaisons possibles sous contraintes pour maximiser le score.

**5.7. Gestion de l’équilibre des groupes (scores et tailles)**

* Taille équilibrée : les groupes ont strictement la même taille, ou un groupe de taille 1 supérieure seulement si le nombre d’étudiants n’est pas divisible par la taille du groupe. Donc, l’équilibre en termes d’effectif est assuré.
* Équilibre des scores : L’algorithme maximise le score total global des groupes, mais il ne garantit pas explicitement que tous les groupes auront des scores proches les uns des autres. Par exemple, il peut arriver qu’un groupe ait un score élevé et un autre un score plus faible, si cela maximise la somme globale.
* Pas de contrainte de minimisation des écarts de score entre groupe: L’algorithme ne contient pas de contraintes spécifiques pour équilibrer les scores (pas d’objectif ou contrainte visant à réduire la variance des scores entre groupes).

**6. Instructions d'installation et de mise en route**

**6.1. Prérequis**

**6.2. Installation**

- git clone https://github.com/Fjsksksk/box-cert-project

- cd box-cert-project

- pip install -r requirements.txt

- pour la bdd, un script sql MySQL se trouve dans le git, il faut le télécharger et l’importer dans un logiciel MySQL (Wamp, Laragon, etc..). Il faut modifier le mot de passe dans App.py afin d’établir les condition.

**6.3. Lancement de l’application**

cd src

python app.py

L'application sera disponible à l'adresse : [http://localhost:5000](http://localhost:5000/)

**7. Points à développer dans les prochaines versions**

* Interface responsive

**8. Crédits**

Candice : Développeuse Front-End — responsable de l’interface utilisateur et de l’architecture générale du projet.

Joe : Développeur Back-End — en charge de la logique serveur, de l’authentification et des interactions avec la base de données.

Maxence : Ingénieur Data — responsable de l’implémentation et de l’optimisation de l’algorithme de regroupement, avec un soutien ponctuel en développement lorsque nécessaire.

## 4. \*\*Fonction objectif (ce que l’algorithme maximise)\*\*

\* On veut \*\*maximiser la somme des affinités mutuelles au sein des groupes\*\*.

\* Pour chaque groupe `g` et chaque paire d’étudiants `(i, j)` dans ce groupe, on ajoute à la fonction objectif la valeur du score mutuel multiplié par une variable booléenne qui vaut 1 si `i` et `j` sont dans le même groupe, 0 sinon.

Cela encourage l’algorithme à placer ensemble les étudiants qui ont des affinités élevées.

---

## 5. \*\*Comment décide-t-il de mettre quelqu’un dans un groupe ?\*\*

L’algorithme, via le solveur \*\*CP-SAT de Google OR-Tools\*\*:

\* Explore les affectations possibles des étudiants aux groupes sous les contraintes.

\* Cherche l’affectation qui \*\*maximise la somme des scores d’affinités mutuelles\*\*.

\* Il garantit qu’un étudiant n’est assigné qu’à un seul groupe.

\* Il respecte la taille exacte de chaque groupe.

La décision est donc automatique et optimisée par le solveur, qui essaie toutes les combinaisons possibles sous contraintes pour maximiser le score.

---

## 6. \*\*Gestion de l’équilibre des groupes (scores et tailles)\*\*

\* \*\*Taille équilibrée\*\* : les groupes ont strictement la même taille, ou un groupe de taille 1 supérieure seulement si le nombre d’étudiants n’est pas divisible par la taille du groupe. Donc, l’équilibre en termes d’effectif est assuré.

\* \*\*Équilibre des scores\*\* :

L’algorithme maximise le score total global des groupes, mais \*\*il ne garantit pas explicitement que tous les groupes auront des scores proches les uns des autres\*\*. Par exemple, il peut arriver qu’un groupe ait un score élevé et un autre un score plus faible, si cela maximise la somme globale.

\* \*\*Pas de contrainte de minimisation des écarts de score entre groupes\*\* :

L’algorithme ne contient pas de contraintes spécifiques pour équilibrer les scores (pas d’objectif ou contrainte visant à réduire la variance des scores entre groupes).

---

## 7. \*\*Résumé\*\*

| Aspect | Comment c’est fait |

| ------------------------------- | ----------------------------------------- |

| Affectation étudiants → groupes | Variables booléennes et contraintes |

| Taille des groupes | Contrôle strict (groupes presque égaux) |

| Score par groupe | Somme des affinités mutuelles dans groupe |

| Décision placement | Optimisation globale par solveur CP-SAT |

| Équilibre des tailles | Oui, groupes équilibrés en nombre |

| Équilibre des scores | Non explicitement, maximise somme totale |

---

Si tu souhaites \*\*équilibrer aussi les scores des groupes\*\* (par ex. éviter qu’un groupe ait un score très haut et un autre très bas), il faudrait ajouter des contraintes ou un terme dans la fonction objectif qui limite l’écart des scores entre groupes (par exemple, minimiser la variance des scores).

---

Si tu veux, je peux t’aider à modifier l’algorithme pour ajouter ce type d’équilibre ou pour clarifier d’autres points !

message.txt5 Ko