documentation :

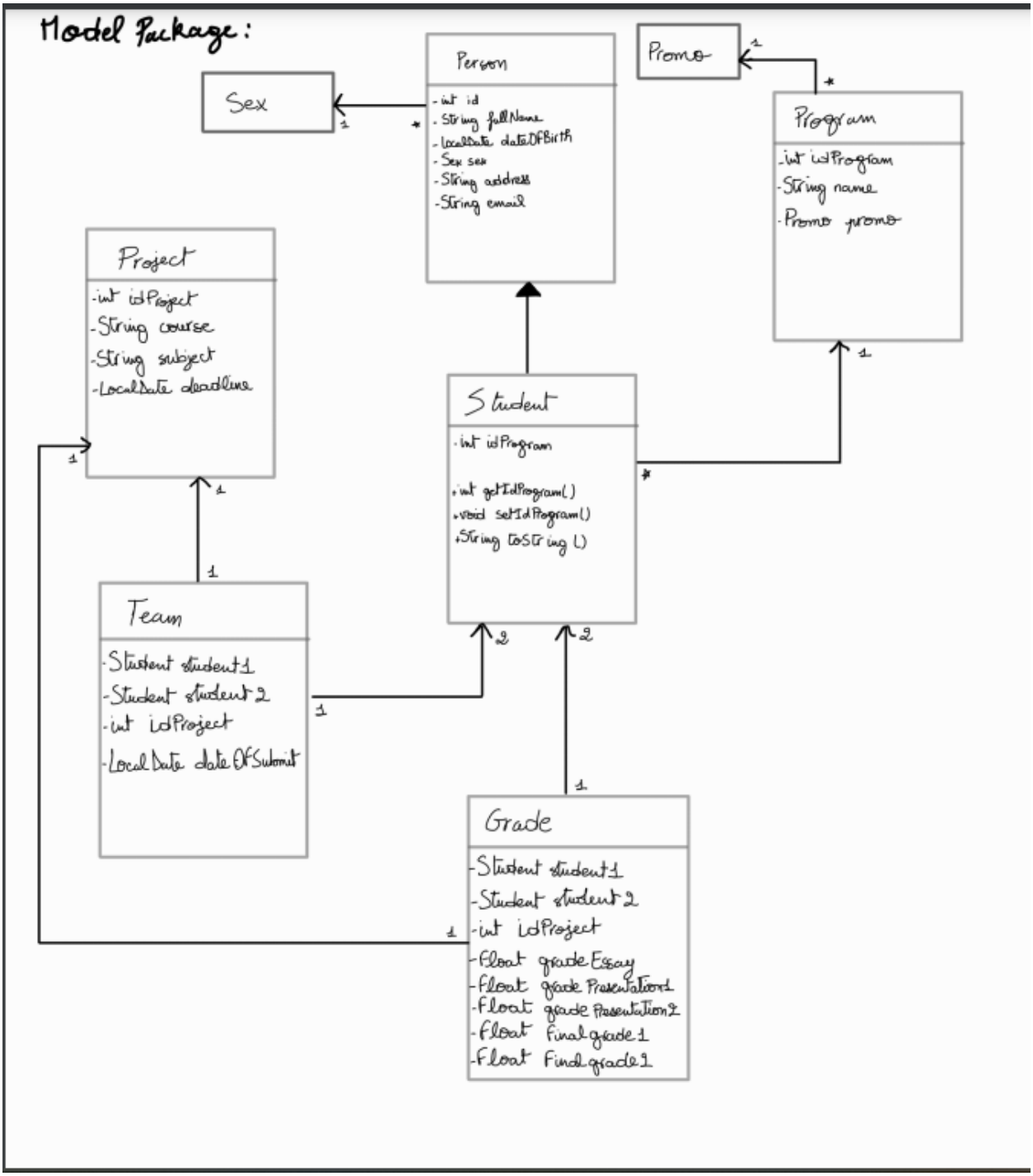
Architecture du projet :

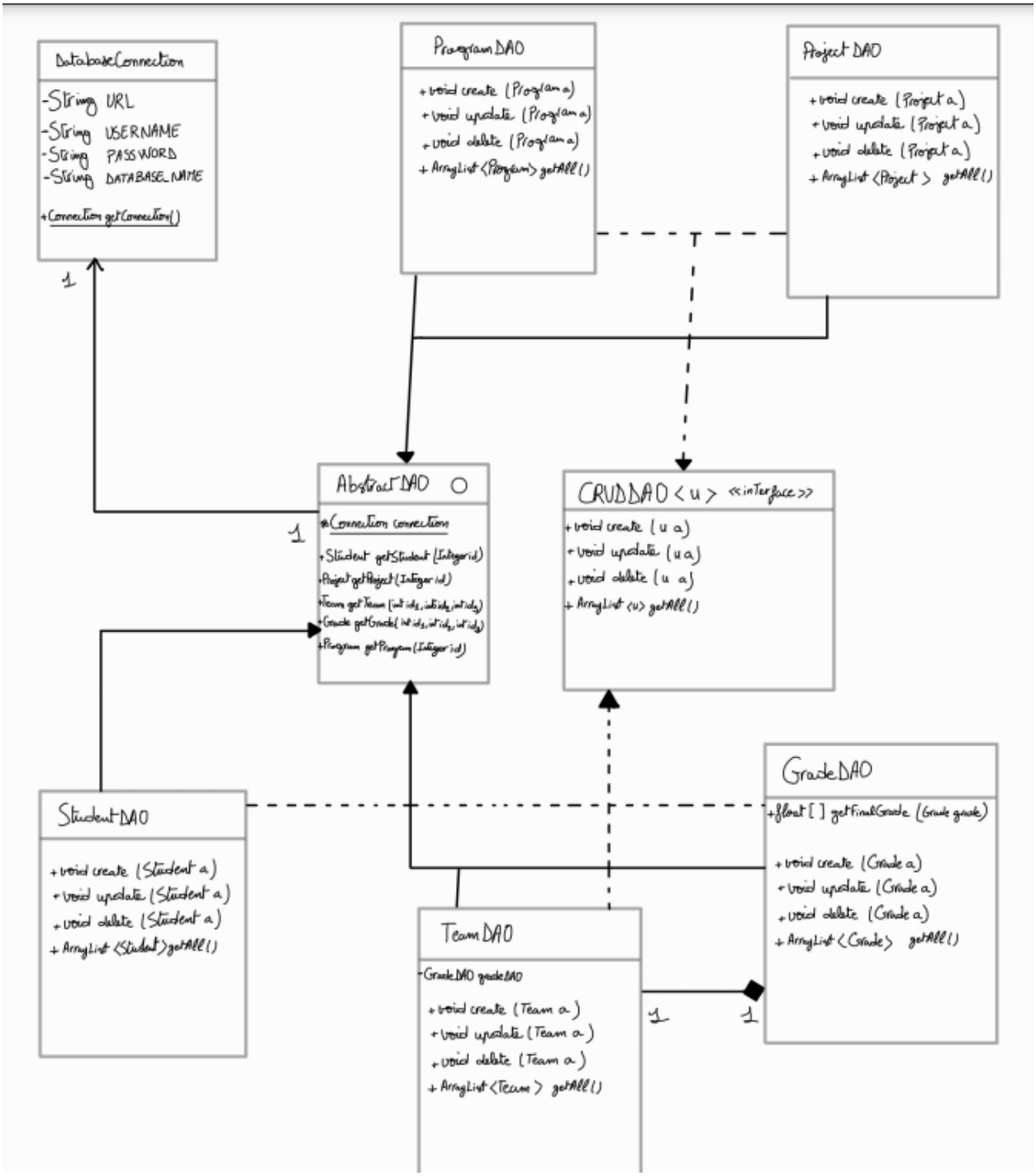
pour ce projet on s’est inspiré de l’architecture MVC (Modèle Vue Controlleur) et on l’a adapté aux besoins de notre projet, pour cela on a organiser notre code dans 5 packages :

- **Model :** ce package gère la **logique métier** de notre application. Il comprend notamment le code de bases de tous nos objets définissant ainsi un modèle pour notre projet. Son objectif est de faciliter le code pour la gestion des données et leurs affichage.
- **Application :** ce package contient la classe admin qui lance l’application.
- **Database :** ce package a pour objectif de fournir une abstraction propre et organisée pour gérer l'accès à la base de données, en séparant les détails de mise en réseau de la logique métier de l'application, afin de favoriser la maintenabilité, la lisibilité du code et la réutilisation des composants liés à la base de données. Il contient les classes DAO (data access object) qui sont utilisés pour encapsuler l'accès aux données spécifiques à une entité métier. Chaque DAO est responsable des opérations CRUD (Create, Read, Update, Delete) pour une entité particulière. On trouve aussi la classe DatabaseConnection qui est responsable à établir la connexion avec la base de données et gérer les paramètres de tels que l'URL de la base de données, le nom d'utilisateur, le mot de passe.
- **View :** ce package se concentre sur l'affichage. Il ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. cette partie gère aussi les **échanges** avec l'utilisateur. Elle va recevoir des requêtes de l'utilisateur. Pour chacune, il va demander à la database d'effectuer certaines actions (afficher les binômes, supprimer un étudiant …) et de lui renvoyer les résultats (la liste des binômes, si la suppression est réussie …). Puis il va adapter ce résultat et l’afficher à l’utilisateur.
- **Util :** Il regroupe les classes d'aide, qui, bien que cruciales pour le fonctionnement de l'application, ne trouvent pas leur place directe dans le modèle. Il héberge souvent des classes utilitaires, des enums, des classes abstraites, et

éventuellement un dossier "resources" contenant les images nécessaires à l'interface graphique de l'application.

Diagramme des classes :





Base de données :

dans notre projet, on utilise une base de données MySQL hébergée en ligne sur le site Freemysql. Cela permet d'avoir un accès centralisé à la base de données à partir de

n'importe quelle machine sans avoir à installer MySQL localement et ça automatise la création de la base de données, ce qui est une approche pratique.

Voici le schéma de nos Tables votre :

Student (ID_student, fullName, dateOfBirth, sex, address, email, #ID_program)

Program(ID_program,name, promotion)

Team (#ID_student1, #ID_student2, #ID_project, dateOfSubmit)

Project (ID_project, course, subject, deadline)

Grade (#ID_student1, #ID_student2, #ID_project, gradeEssay, gradePresentation1, gradePresentation2, finalGrade1, finalGrade2)

Choix d’héritages, Design Pattern et Interfaces :

Model et Util

- **héritage et polymorphisme de la classe `Person` et `Student` :** un étudiant est une sorte de personne, mais avec des caractéristiques ou des comportements plus spécifiques. La classe `Personne` contient des attributs et des méthodes générales. En héritant de cette classe, la classe `Student` peut réutiliser ces fonctionnalités sans avoir à les redéfinir, et peut ajouter ces attributs et méthodes supplémentaires sans avoir à réécrire les caractéristiques de base d'une personne. Cet héritage permet d'utiliser le polymorphisme. Cela est utile dans des situations où on traite avec une liste de personnes, mais certaines d'entre elles ne sont pas des étudiants.

Database

- on utiliser **Le modèle de conception Data Access Object (DAO)** pour abstraire l'accès aux données et encapsuler les opérations de base de données. Ce choix de design pattern nous a permet d’avoir une couche d'abstraction entre la logique métier de l'application et les détails spécifiques de la gestion des données dans une base de données. Cela a permet de changer plus facilement la source de données sans avoir à modifier le reste de l'application.
- En regroupant les opérations de base de données (CRUD - Create, Read, Update, Delete) dans des interfaces et des classes DAO, on favorise la réutilisabilité du

code. Chaque entité métier peut avoir son propre DAO, ce qui rend le code plus modulaire.

- Le DAO contribue à la séparation des préoccupations en isolant le code d'accès aux données du reste de l'application. Cela améliore la lisibilité du code, facilite la maintenance et permet à l'équipe de développement de se concentrer sur des aspects spécifiques du projet sans se soucier des détails de la persistance des données.
- Concernant l'implémentation d'une interface `CRUD` avec un type générique, on a gagné la **Flexibilité;** L'utilisation d'un type générique permet au DAO de manipuler différents types d'entités sans avoir besoin d'implémenter des méthodes spécifiques pour chaque type (les objets sont de types différents et l'interface `CRUD` nous permet d'utiliser les même méthodes pour tous les objets).
- **Réduction de la redondance de code :** En utilisant un type génériquedans l'interface `CRUD` et la classe abstraite `AbstractDAO`, on a évité la duplication de code pour des opérations similaires sur différentes entités. Les méthodes peuvent être écrites une fois dans la classe `AbstractDAO` et réutilisées avec différents types d'entités.

Rq : l'interface `CRUD` ne contient pas la méthode `read()` car la clé primaire de nos objets est différente, on a des objets qui ont pour ID juste un INT et d'autre qui ont pour ID un triplet de valeurs, et donc pour éviter la redondance du code, on a introduit toutes les méthodes `read()` dans la classe `AbstractDAO`, car on aura besoin dans toutes les classes DAO .

- **Consistance :** L'utilisation de types génériques et de la classe abstraite `AbstractDAO` garantit une certaine consistance dans la manière dont les opérations CRUD sont implémentées pour différentes entités. Cela peut rendre le code plus prévisible et faciliter la maintenance.

les difficultés rencontrées

Durant ce projet, on a été confronté à plusieurs difficultés qui ont impacté le processus de développement de manière significative. La première difficulté majeure était l'installation de l'environnement de développement. Avant même de commencer à coder, il a été nécessaire d'installer et de configurer le serveur MySQL, de mettre à jour

l'IDE Java et le JDK, ainsi que de gérer les build paths nécessaires. Cela a entraîné des retards importants, on a dû garantir que les membres partagent le même environnement de travail. Les problèmes d'installation ont consommé beaucoup de temps, car des ajustements fréquents étaient nécessaires pour résoudre les conflits entre les versions des logiciels.

La deuxième difficulté majeure a été l'adaptation à la librairie JDBC pour la manipulation de la base de données. Comprendre le fonctionnement de JDBC et mettre en œuvre le modèle de conception DAO ont représenté un défi, nécessitant une courbe d'apprentissage importante. on a dû investir du temps dans la documentation et les tutoriels afin de maîtriser efficacement ces concepts, ce qui a eu un impact sur la productivité initiale du projet.

La troisième difficulté a émergé lors du travail en binôme avec l'utilisation de Git pour la gestion de version. Les problèmes de synchronisation ont été fréquents, en particulier avec les pulls qui modifiaient le code local et généraient des erreurs, même si le code en remote et dans la branche du master était correct. Malgré plusieurs tentatives de résolution, y compris le clonage du projet, le changement de repositories, et la demande d'aide à des amis développeurs, aucune solution n'a été trouvée rapidement. En conséquence, on a décidé de suspendre temporairement l'utilisation de GitHub et de travailler localement pour éviter de perdre davantage de temps.

Ces difficultés soulignent l'importance d'une planification minutieuse de l'environnement de développement, de la compréhension préalable des technologies utilisées et de la mise en place d'une gestion de version efficace dès le début du projet. Ces expériences peuvent également être des leçons précieuses pour les projets futurs, incitant à accorder une attention particulière à la configuration de l'environnement et à la gestion collaborative du code source.

la répartition du travail

La répartition du travail entre les membres du binôme, on a suivi un processus collaboratif bien défini tout au long du projet. Au départ, Anfel a pris en charge la conception initiale du projet, y compris la conception de la base de données, et a élaboré la planification des différentes étapes du projet. De plus, Anfel a commencé à coder la base des classes nécessaires au projet (package Model).

Par la suite, la majorité du temps a été consacrée au pair programming, une approche où on a codé ensemble, partageant des idées et résolvant les problèmes en travaillant côte à côte. Une première étape de cette collaboration a été la création des classes essentielles, notamment la classe `DatabaseConnection` et la classe `StudentDAO`. En utilisant ces classes comme exemples, on a travaillé conjointement sur la création des autres classes DAO, chacune prenant en charge des classes spécifiques. Anfel a développé les classes `TeamDAO` et `GradeDAO`, mettant en œuvre des fonctionnalités telles que le calcul de la note finale en fonction des retards. Barbara, de son côté, s'est concentrée sur les classes `ProjectDAO` et `ProgramDAO`.

Lors de la conception de l'interface utilisateur, on a continuer à faire du pair programming, travaillant ensemble sur le design et les fonctionnalités de l'application. Les classes `LoginView` et `HomeView` ont été codées en tandem. Ensuite, la répartition s'est poursuivie avec Anfel travaillant sur les classes `GradeView`, `TeamView`, et `ProjectView`, tandis que Barbara s'est occupée des classes `StudentView` et `StudentInfoView`.

À la fin du processus de développement individuel, Anfel et Barbara ont réuni leur code, consolidant les différentes parties du projet en un ensemble cohérent. Anfel a ensuite dirigé les efforts pour résoudre les bugs, effectuer des tests, et ajouter des détails supplémentaires à l'application, tels que l'implémentation des boutons "back" et la mise en place de filtres pour l'affichage de la liste des étudiants en fonction de leurs programmes respectifs. et barbara s’est concentré sur l’implémentation du projet Maven et l’exportation du fichier JAR.

les fonctionnalités apportées

L'application que nous avons développées pour un utilisateur administrateur offre une gamme complète de fonctionnalités permettant une gestion efficace des étudiants, des projets, des binômes et des évaluations. Voici une explication détaillée des fonctionnalités apportées :

1. **Gestion des étudiants :**
    - **Visualisation de la liste des étudiants :** L'administrateur a la possibilité de consulter une liste complète de tous les étudiants inscrits. ou selon leurs formation en la sélectionnant.

- **Création d'étudiants :** L'administrateur peut ajouter de nouveaux étudiants à la base de données, facilitant ainsi la gestion des nouvelles inscriptions.
- **Suppression d'étudiants :** En cas de besoin, l'administrateur peut supprimer des étudiants de la base de données, gérant ainsi les désinscriptions ou les erreurs de saisie.
- **Visualisation des informations d'un étudiant :** L'accès aux détails individuels d'un étudiant permet à l'administrateur de consulter des informations spécifiques telles que les projets auxquels l'étudiant participe, les binômes formés, et les notes attribuées.

2. **Gestion des projets :**
    - **Visualisation de tous les projets :** L'administrateur peut parcourir la liste complète des projets disponibles, obtenant ainsi une vue d'ensemble des initiatives en cours.
    - **Ajout et Suppression de projets :** Lorsque cela est nécessaire, l'administrateur peut retirer des projets de la liste, entraînant automatiquement la suppression des binômes associés.
    - **Visualisation et gestion des binômes d'un projet :** En sélectionnant un projet spécifique, l'administrateur peut consulter la liste des binômes associés. La gestion inclut la suppression de binômes existants et l'ajout de nouveaux binômes.
    - **Ajout de dates de soumission pour les projets :** L'administrateur peut attribuer des dates de soumission aux projets, offrant ainsi une approche structurée pour gérer les délais. Cela est particulièrement utile pour les binômes qui n'ont pas encore soumis leurs projets.
3. **Gestion des binômes :**
    - **Visualisation des binômes d'un projet :** L'administrateur peut accéder à la liste des binômes travaillant sur un projet spécifique, facilitant ainsi le suivi des collaborations.
    - **Suppression et ajout de binômes :** La possibilité de retirer des binômes existants ou d'ajouter de nouveaux binômes donne à l'administrateur un contrôle complet sur la composition des équipes de projet (un étudiant ne peut pas être dans plus d’une équipe dans le même projet).

- **Notation des binômes :** L'administrateur peut attribuer des notes aux binômes pour le rapport et la présentation, offrant une évaluation détaillée du travail fourni.
- **Génération automatique de la note finale :** La note finale du projet est générée automatiquement en fonction des notes attribuées au rapport et à la présentation, ainsi que de la date de soumission du projet.

Problème Contourné :

Lorsque nous avons cru que notre projet était finalisé et que nous avons envoyé la dernière version, une réalisation perturbante est survenue : nous avions négligé la considération importante selon laquelle les deux membres du binôme devaient recevoir des notes distinctes pour la présentation. Cette omission a nécessité une révision complète de tout ce qui était lié à l'évaluation, particulièrement la table `Grade` dans la base de données, l'interface graphique dédiée à l'évaluation, ainsi que toutes les autres classes qui utilisaient la classe `Grade` ou `GradeDAO`.

La correction de ce problème a été un processus intensif. Nous avons dû restructurer la manière dont les données d'évaluation étaient stockées dans la base de données, en ajoutant une distinction claire entre les notes de présentation de chaque membre du binôme. Cela a également impliqué des ajustements dans l'interface utilisateur, les classes DAO, et d'autres parties du code qui interagissaient avec les données d'évaluation.

Ce défi inattendu a été une leçon précieuse, soulignant l'importance cruciale de bien comprendre et de respecter le cahier des charges dès les premières étapes du projet. Le fait de laisser cette vérification jusqu'à la fin du processus de développement a entraîné des retards significatifs et a nécessité un effort supplémentaire pour corriger les erreurs.