Steeve XU
Nick WANG

M1 MIAGE
11/2023

Documentation du Projet

1. Introduction

1.1. Présentation du projet

Le projet vise à développer une application de gestion de données universitaire, mettant en œuvre une interface graphique pour la manipulation des informations liées aux formations, aux étudiants, aux projets, aux binômes (pairs), et aux évaluations (grades). Chaque formation est caractérisée par un numéro unique, un nom, et une promotion. Les étudiants sont enregistrés avec leurs informations personnelles et affiliés à une formation spécifique. Les projets sont également enregistrés avec des détails tels que la matière, le sujet et la date de remise. Les binômes d'étudiants travaillant sur des projets sont enregistrés, et des évaluations sont associées à chaque projet, comprenant des notes pour le rapport et les soutenances. L'application permet d'ajouter, mettre à jour, supprimer et visualiser ces données, offrant une gestion complète des informations universitaires.

1.2. Objectifs et portée

Ce projet est une application dédiée aux enseignants qui souhaitent évaluer et noter les étudiants en fonction de divers critères tels que la formation, les étudiants, les binômes, les projets et les notes associées. L'objectif principal est de fournir une plateforme centralisée pour permettre aux enseignants de gérer et d'enregistrer ces informations de manière efficace, facilitant ainsi le suivi des performances académiques des étudiants. La portée de l'application englobe la conservation et la manipulation de données essentielles liées aux activités universitaires, offrant ainsi une solution complète pour la gestion des évaluations.

2. Choix de Conception

2.1. Architecture de l'Application

Le projet qui nous a été confié est la création d'une interface graphique permettant à un enseignant de pouvoir gérer les projets des étudiants pour l'année scolaire courante.

Voici un diagramme UML nous présentant l'architecture que nous avons choisie d'utiliser pour la réalisation de notre application.

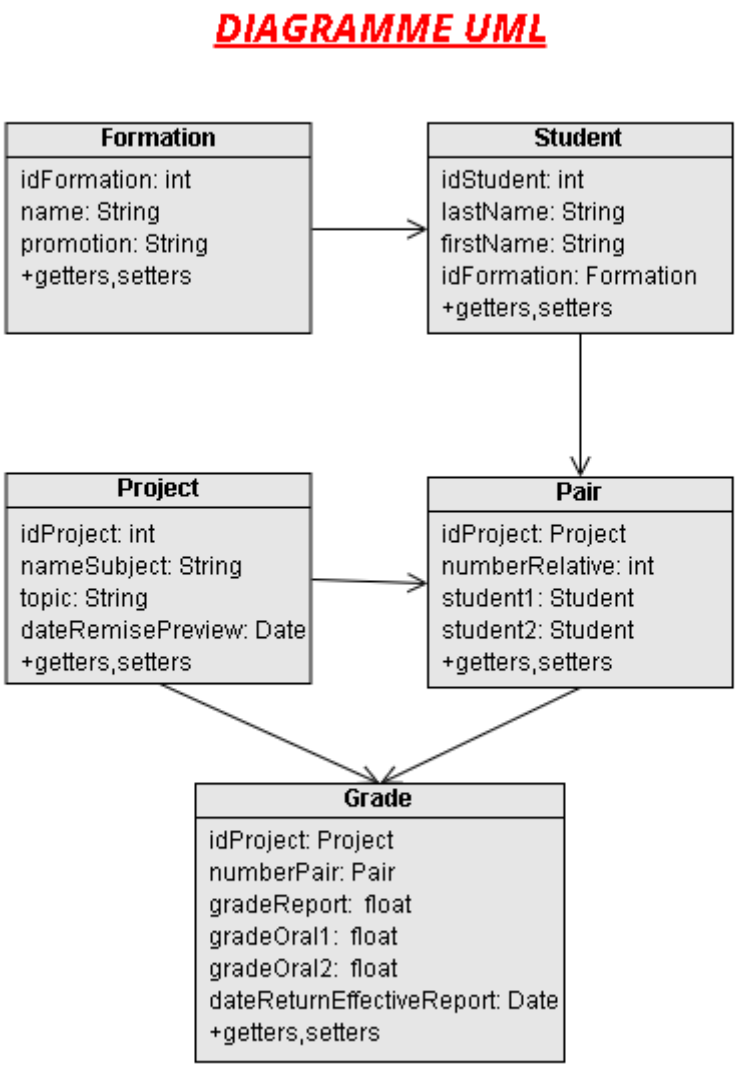


*Figure 1 : **Diagramme** UML*

Sur notre diagramme, L'entité principale est « *Formation* », caractérisée par un identifiant unique (*idFormation*), un nom de formation ne pouvant prendre que les valeurs « ID », « MIAGE-IF » et « MIAGE-SITN », ainsi qu'une promotion ne pouvant prendre que les valeurs « Initiale », « En Alternance », « Formation Continue ». Chaque « Formation » peut

avoir plusieurs « *Student* », identifiés par un identifiant unique (*idStudent*) et comprenant des détails tels que le nom de l'étudiant, son prénom ainsi que l'identifiant de la formation suivie (clé étrangère renvoyant à *idFormation* de la table Formation)

Les « *Student* » sont associés à des « *Project* » auxquels ils participent en tant que membres d'une « Pair ». Chaque « Project » est défini par un identifiant unique (*idProject*), un nom de matière (*nameSubject*), un sujet (*topic*), et une date de remise prévue. Les « Pair » sont composés de deux « *Student* » (clé étrangère renvoyant à *idStudent* de la table « *Student* ») et sont liés à un « Project » (clé étrangère renvoyant à *idProject* de la table « *Project* »). Enfin, les évaluations des projets sont stockées dans la classe « *Grade* », caractérisée par un identifiant de projet (*idProject*, clé étrangère renvoyant à la table « *Project* »), un identifiant de pair (*numberPair*, clé étrangère renvoyant à la table « *Pair* ») que nous avons décidé de nommer (*ID Pair*) dans notre interface graphique, et des notes pour le rapport, l'oral du premier étudiant (*gradeOral1*) et l'oral du deuxième étudiant (*gradeOral2*). La date de remise effective du rapport est également enregistrée.

En résumé, ce diagramme offre une vue structurée des entités clés, de leurs attributs et des relations entre elles au sein du système.

2.2. Justification des décisions prises

Nous avons délibérément opté pour ces tables et leurs relations afin de permettre la création de plusieurs onglets dans notre interface graphique. Cette structure a été choisie en raison de sa cohérence avec les objectifs de notre projet. Bien que d'autres approches aient pu être envisagées, comme l'insertion directe des notes dans la table « *pair* » qui partage les mêmes clés étrangères, notre choix s'est orienté vers cette configuration, car elle répondait mieux à nos besoins spécifiques.

3. La Répartition des taches

Pour accroître notre productivité, nous avons partitionné le projet en deux parties distinctes : une consacrée à l'interface (Nick) et une autre dédiée à la gestion des données et aux méthodes Java (Steeve). Cette séparation nous a permis d'organiser de manière plus claire les responsabilités, facilitant ainsi le développement, la maintenance et l'évolution de notre application. La partie interface se concentre sur l'aspect visuel et l'interaction utilisateur, tandis que la partie gestion des données met en œuvre les logiques métier, les connexions à la base de données et les opérations CRUD. Cette approche a contribué à une meilleure gestion du projet et à une collaboration plus fluide dans notre binôme.

4. Les Fonctionnalités

Nous avons envisagé d'implémenter la fonctionnalité permettant de sélectionner les lignes de données que nous ajoutons à notre base de données.

Voici un exemple du code pour sélectionner les lignes de notre table *Formation* :

```
@FXML
void getData(MouseEvent event) {
    Formation formation = table1.getSelectionModel().getSelectedItem();
    idFormation = formation.getIdFormation();
    tNameFormation1.setText(formation.getName());
    tPromotion.setText(formation.getPromotion());
    btnSave1.setDisable(true);
}
```

Figure 2 : Méthode getData

Ainsi, cela nous donne la possibilité de modifier (*Update*, Figure 4) et de supprimer (*Delete*, Figure 5) les lignes que nous sélectionnons d'un simple clic. Bien que cette fonctionnalité ait présenté quelques complexités lors de sa mise en place, à travers la consultation de différentes vidéos en ligne et des recherches sur Google, nous avons réussi à l'intégrer avec succès dans notre application.

```java
@FXML
void updateFormation(ActionEvent event) {

    String update = "update sql11666264.Formation set name = ?, promotion = ? where idFormation = ?";
    con = DBConnect.getCon();
    try {
        st = con.prepareStatement(update);
        st.setString( parameterIndex: 1,tNameFormation1.getText());
        st.setString( parameterIndex: 2,tPromotion.getText());
        st.setInt( parameterIndex: 3,idFormation);
        st.executeUpdate();
        showFormations();
        clear();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
```

Figure 3 : Méthode updateFormation

```java
@FXML
void deleteFormation(ActionEvent event) {
    String delete = "delete from sql11666264.Formation where idFormation = ?";
    con = DBConnect.getCon();
    try {
        st = con.prepareStatement(delete);
        st.setInt( parameterIndex: 1,idFormation);
        st.executeUpdate();
        showFormations();
        clear();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }

}
```

Figure 5 : Méthode deleteFormation

En ce qui concerne les méthodes, nous avons également mis en place deux autres fonctions pour ajouter des données (*Save*, Figure 6) dans notre table, ainsi que pour vider les zones de saisie (*Create*, Figure 7).

```java
@FXML
void creatFormation(ActionEvent event) {

    String insert = "insert into sql11666264.Formation(name,promotion) values(?,?)";
    con = DBConnect.getCon();
    try {
        st = con.prepareStatement(insert);
        st.setString( parameterIndex: 1,tNameFormation1.getText());
        st.setString( parameterIndex: 2,tPromotion.getText());
        st.executeUpdate();
        clear();
        showFormations();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }


}
```

Figure 6 : Méthode creatFormation (Save)

```java
@FXML
void clearField(ActionEvent event) {
    clear();
}
```

Figure 7 : Méthode clearField (Create)

Par ailleurs, l'amélioration de la qualité visuelle de notre interface graphique, construite avec *SceneBuilder*, a été réalisée en utilisant un fichier CSS ("style.css"). Ce dernier a permis de définir la mise en forme, la disposition et l'emplacement précis de nos divers boutons et tableaux, contribuant ainsi à l'esthétique générale de l'application.

5. Les Difficultés Rencontrées

Au début de notre projet, nous avons été confrontés à notre première expérience avec une interface graphique. Pour surmonter cette étape, nous avons entrepris des recherches approfondies et consulté diverses sources documentaires afin de nous familiariser avec la conception et la mise en œuvre d'une interface graphique pour notre application via JavaFx.

Nous avons également rencontré des problèmes aux niveau de notre base de données MySQL car cette dernière était une base de données « local » ce qui pose un problème lorsque nous souhaitons lancer notre projet sur une autre machine, par exemple ceux de l'Université Paris Dauphine. Afin de résoudre ce problème nous avons été amenés à héberger notre base de données local sur un serveur (Freesqldatabase).

La gestion de plusieurs classes au sein d'une unique interface a présenté des défis, d'où notre choix de concevoir une interface à onglets distincts. Cette approche nous offre l'avantage d'avoir un accès facile à toutes nos tables et de pouvoir manipuler efficacement l'ensemble de nos données. Chaque onglet correspond à une classe spécifique, simplifiant ainsi la navigation et l'interaction avec les différentes fonctionnalités de notre application.

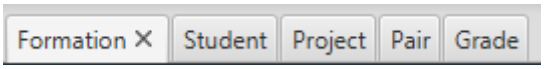
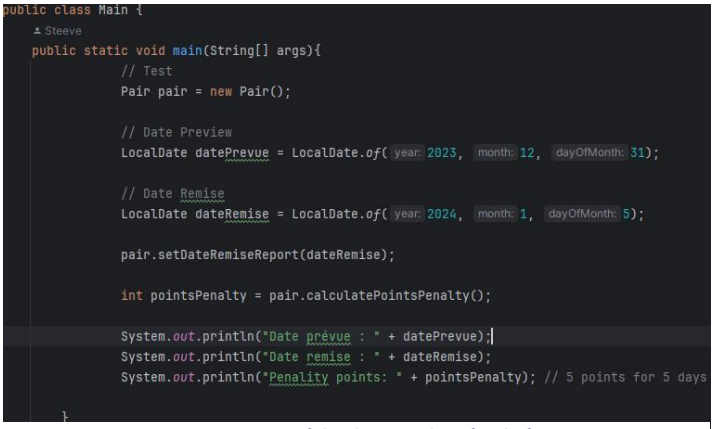


Figure 8 : Onglets de navigation

Par ailleurs, dans notre classe "*Pair*", nous sommes confrontés à l'incapacité d'utiliser les boutons que nous avons créés pour les opérations de mise à jour (*Update*) et de suppression (*Delete*). Actuellement, nous sommes limités à la création de binômes, et malgré nos efforts, la résolution de ce problème nous est impossible. Il convient de noter que les boutons dans nos autres classes fonctionnent parfaitement, rendant cette situation d'autant plus suspecte.

Un dernier problème que je souhaite souligner concernant la méthodologie de calcul des pénalités. Bien que nous ayons créé une méthode conforme aux exigences du cahier des charges, nous n'avons malheureusement pas réussi à l'intégrer pleinement dans notre interface graphique. Cette méthode peut uniquement être utilisée à partir du Main de notre projet, en spécifiant la date prévue de rendu (*datePreview*) ainsi que la date effective de remise du rapport (*dateRemise*). Notre méthode calcule le nombre de points de pénalité (1 point / jour de retard) en fonction du retard accumulé dans le rendu du projet.



```
public class Main {
    Steeve
    public static void main(String[] args){
            // Test
            Pair pair = new Pair();

            // Date Preview
            LocalDate datePrevue = LocalDate.of( year: 2023,  month: 12,  dayOfMonth: 31);

            // Date Remise
            LocalDate dateRemise = LocalDate.of( year: 2024,  month: 1,  dayOfMonth: 5);

            pair.setDateRemiseReport(dateRemise);

            int pointsPenalty = pair.calculatePointsPenalty();

            System.out.println("Date prévue : " + datePrevue);
            System.out.println("Date remise : " + dateRemise);
            System.out.println("Penality points: " + pointsPenalty); // 5 points for 5 days

    }
```

Figure 9 : Méthode pour la pénalité

6. Conclusion

En conclusion, notre application satisfait les exigences de l'énoncé. Malgré quelques problèmes dans l'onglet Pair et l'implémentation de la méthode *calculatePointsPenalty*, nous sommes en mesure de gérer et stocker efficacement les données dans une base de données grâce à notre application.

Ce projet a constitué une réelle opportunité pour nous d'apprendre à concevoir une interface graphique. Pendant notre L3 MIAGE à l'Université Paris Dauphine, nous n'avions ni l'occasion ni le temps d'aborder cette facette de l'interface graphique dans nos projets. Cet apprentissage marque le début d'une meilleure compréhension du développement d'applications en Java, ouvrant la voie à la création d'applications plus avancées à l'avenir.