Rapport du projet d’application CRUD en JAVA FX

Nom complet : Malki Nawal

> Pour vous faciliter la lecture
>
> Nom complet : Malki Nawal
> Introduction
> - Étape 1 : Conception
> - Étape 2 : Conception du code et utilisation de l'architecture MVC
> - Etape 3 : Code
>
> Conclusion

Introduction

Dans ce rapport, je vais présenter le projet de développement d'une application de gestion des étudiants pour l'ENSAO. L'objectif de cette application est de simplifier l'inscription des étudiants, la gestion de leurs informations et la visualisation des données associées.

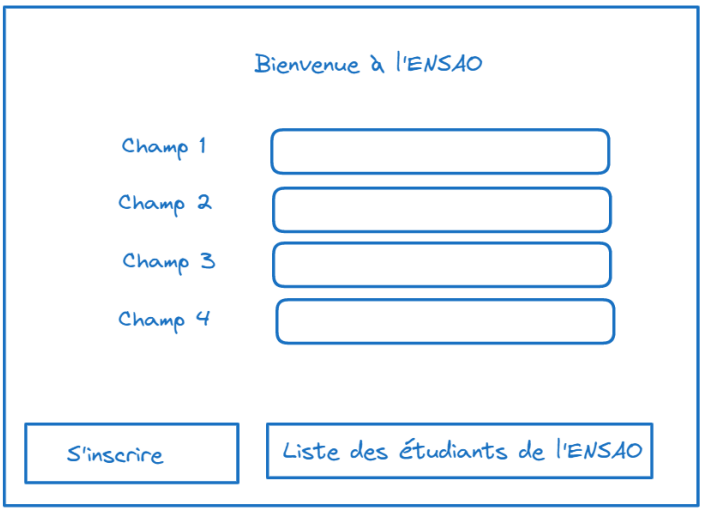
Je vais décrire les différentes étapes du projet, en commençant par la conception de l'interface utilisateur conviviale, jusqu'à l'implémentation du code en utilisant l'architecture Modèle-Vue-Contrôleur (MVC). Je vais également expliquer les choix de conception que j'ai faits, et les fonctionnalités clés de l'application .

Étape 1 : Conception

Après avoir examiné attentivement le sujet de notre projet, j'ai commencé par concevoir une interface utilisateur conviviale et intuitive pour faciliter l'utilisation de l'application.

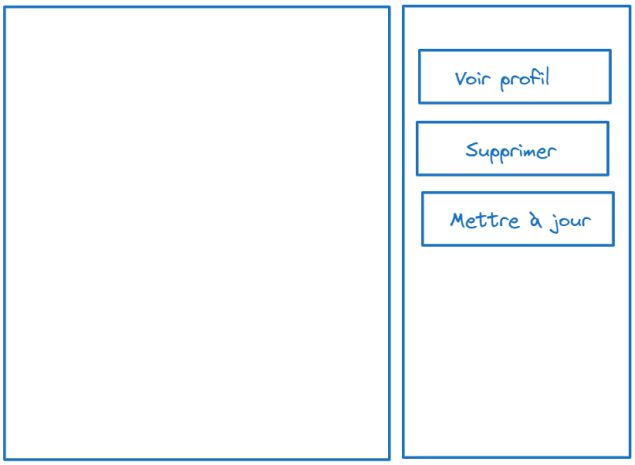
J'ai opté pour un design comprenant trois scènes principales. La première scène contient un formulaire d'inscription pour ajouter un nouvel étudiant à l'ENSAO. Cette scène comporte des champs tels que la note du bac, l'année d'obtention du bac, le nom, le prénom, le code Massar, l'année de naissance, etc. Un bouton "S’inscrire" permettra de soumettre le formulaire et un boutton “Liste des étudiants de l’ENSAO” permettera de passer à la scène 2.

Scène 1

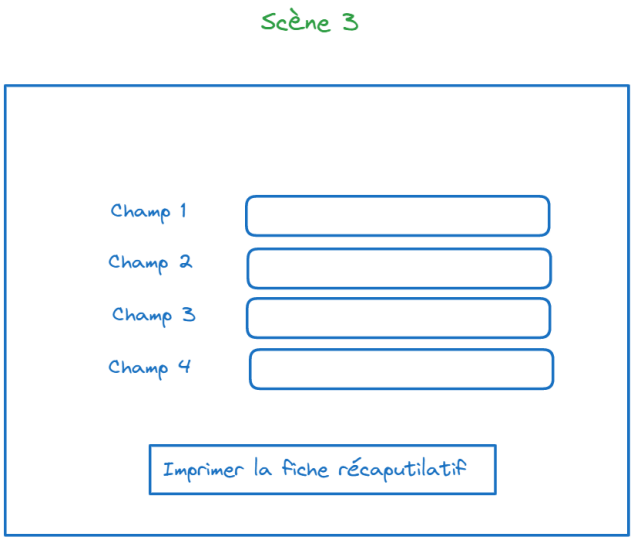


La deuxième scène affiche la liste des étudiants inscrits à l'ENSAO. J'ai utilisé une TableView pour afficher les différents champs d'inscription des étudiants. Les données de la TableView seront récupérées à partir d'une table "Etudiant" dans la base de données “ENSAO” . Dans cette scène, plusieurs boutons sont disponibles sur le côté droit pour gérer les événements liés à la liste des étudiants. Par exemple, lorsque l'utilisateur sélectionne un étudiant dans la TableView (la ligne est mise en surbrillance), en cliquant sur le bouton "Supprimer", l'étudiant sera supprimé à la fois de la base de données et de la TableView. De même, en cliquant sur le bouton "Mettre à jour", l'utilisateur pourra modifier les valeurs des colonnes de la ligne sélectionnée dans la TableView. Lorsque l'utilisateur clique sur le bouton "Voir profil", il est redirigé vers la troisième scène.

Scène 2



La troisième scène est dédiée à l'affichage du profil détaillé d'un étudiant spécifique. Elle contient tous les champs d'inscription remplis par l'étudiant lors de son inscription à l'ENSAO.



Étape 2 : Conception du code et utilisation de l'architecture MVC

Dans cette étape de conception, j'ai réfléchi à la manière dont le code de l'application serait structuré pour garantir sa maintenabilité et sa flexibilité. J'ai choisi d'utiliser l'architecture Modèle-Vue-Contrôleur (MVC) pour plusieurs avantages qu'elle offre.

L'architecture MVC permet de séparer les préoccupations liées à la logique métier, à la présentation des données et à la gestion des événements. Cela facilite la maintenance du code, car chaque composant a une responsabilité spécifique.

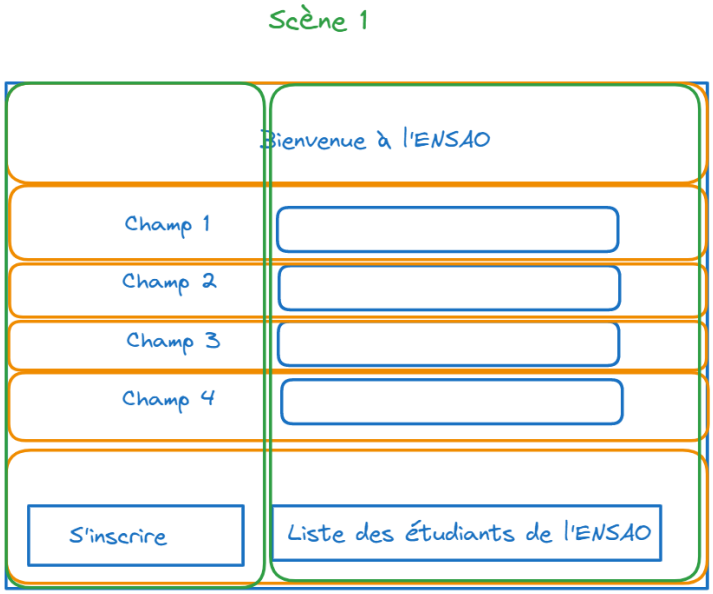
J'ai donc décomposé mon code en trois parties distinctes :

1. **Modèle** : Cette partie représente la logique métier de l'application. Elle comprend la classe qui définit l'entité qui s'appelle "**Etudiant**" et avec ses attributs et ses méthodes. J'ai également inclus une classe pour la gestion de la base de données qui s'appelle "**Connexion**", qui s'occupe de la configuration de la connexion à la base de données
2. **Vue** : Cette partie concerne l'interface utilisateur de l'application. J'ai codé chaque scène de façon programmatique en utilisant JavaFX, sans recourir à des fichiers de mise en page FXML. J'ai utilisé les composants JavaFX tels que `TextField`, `Button`, `TableView`, etc., pour créer les scènes et afficher les données à l'utilisateur.
3. **Contrôleur** : Cette partie agit comme un intermédiaire entre le modèle et la vue. Les contrôleurs sont responsables de la gestion des événements utilisateur, de la coordination

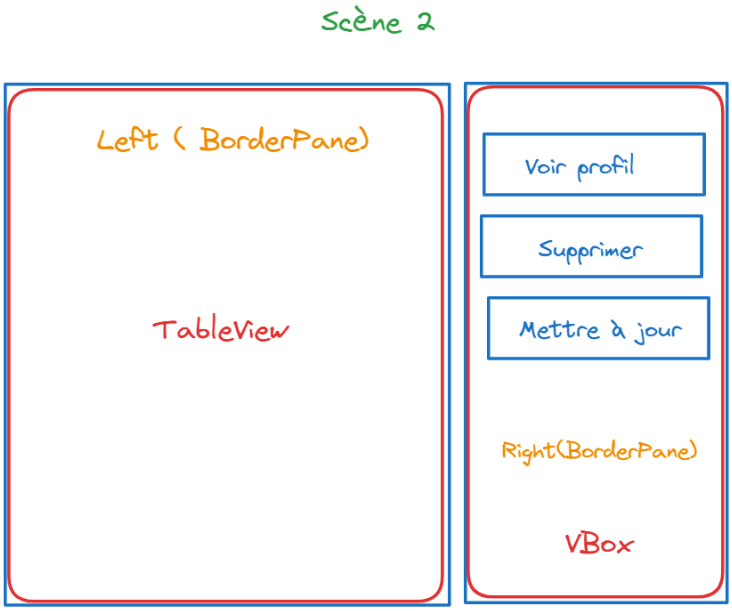
des actions entre la vue et le modèle, et de la mise à jour de l'interface utilisateur en fonction des modifications apportées aux données. J'ai utilisé une classe de contrôleur toutes les scènes , où j'ai défini des méthodes pour les actions des boutons, les événements de sélection de la TableView, etc.

En ce qui concerne les mises en page (layouts) utilisées dans chaque scène :

Pour la scène 1 , j’ai décidé d’utiliser un GridPane avec un nombre de lignes qui dépend des champs d’inscription à remplir et 2 colonnes .



Pour la scène 2 j’ai décidé d’utiliser un `BorderPane` parce qu’il me permettera le mieux la vue que je veux , j’ai mis à gauche la tableView et à droite un autre layout : VBox qui contiendra tous les bouttons de manipulation de la tableView



Enfin, pour la scène 3, j'ai également choisi d'utiliser un `VBox`.

Etape 3 : Code

J'ai implémenté le code en utilisant les concepts et les classes définis lors de la conception. J'ai pris en compte les interactions entre le modèle, la vue et le contrôleur pour assurer un fonctionnement cohérent de l'application.J'ai également géré les opérations de manipulation des données dans le modèle et la base de données en utilisant des requêtes SQL appropriées. Par exemple, pour l'ajout d'un nouvel étudiant, j'ai utilisé une requête INSERT INTO pour insérer les valeurs des champs dans la table "Etudiant".

De plus, j'ai utilisé des composants JavaFX tels que TextField, Button et TableView pour afficher les données à l'utilisateur et lui permettre d'interagir avec l'application. J'ai attaché des gestionnaires d'événements aux boutons pour effectuer des actions telles que la suppression d'un étudiant sélectionné ou la mise à jour des données d'un étudiant.

L'architecture MVC m'a aidé à organiser mon code de manière claire et modulaire. J'ai séparé les responsabilités en utilisant des classes distinctes pour le modèle, la vue et le contrôleur, ce qui facilite la maintenance et l'extension de l'application ,et j'ai aussi utilisé des mises en page appropriées, telles que GridPane, BorderPane et VBox, pour organiser les éléments de l'interface utilisateur de manière esthétique et fonctionnelle.

Enfin, j'ai veillé à prendre en compte les bonnes pratiques de programmation, telles que la gestion des exceptions, la validation des entrées utilisateur et l'utilisation de noms de variables et de méthodes significatifs.

Conclusion

En conclusion, je tiens à exprimer ma satisfaction concernant le cours de ce semestre et mon expérience avec JavaFX. Initialement, j'avais des réserves quant à l'intérêt de cette technologie, mais grâce à la préparation de l'examen et à la réalisation de ce projet, j'ai découvert tout son potentiel. Sa simplicité d'utilisation et sa flexibilité m'ont permis de créer une application de gestion des étudiants pour l'ENSAO de manière efficace.Ce projet m'a également permis de mettre en pratique les concepts d'architecture Modèle-Vue-Contrôleur (MVC) et d'apprécier les avantages qu'elle offre en termes de maintenabilité et d'évolutivité du code.

Je tiens à vous remercier pour votre accompagnement tout au long du semestre et pour m'avoir donné l'opportunité de découvrir et d'apprécier JavaFX.