Introduction à JavaFX

A travers la création d'une application « TodoList »

Sommaire

Gestion des TableView en JavaFX

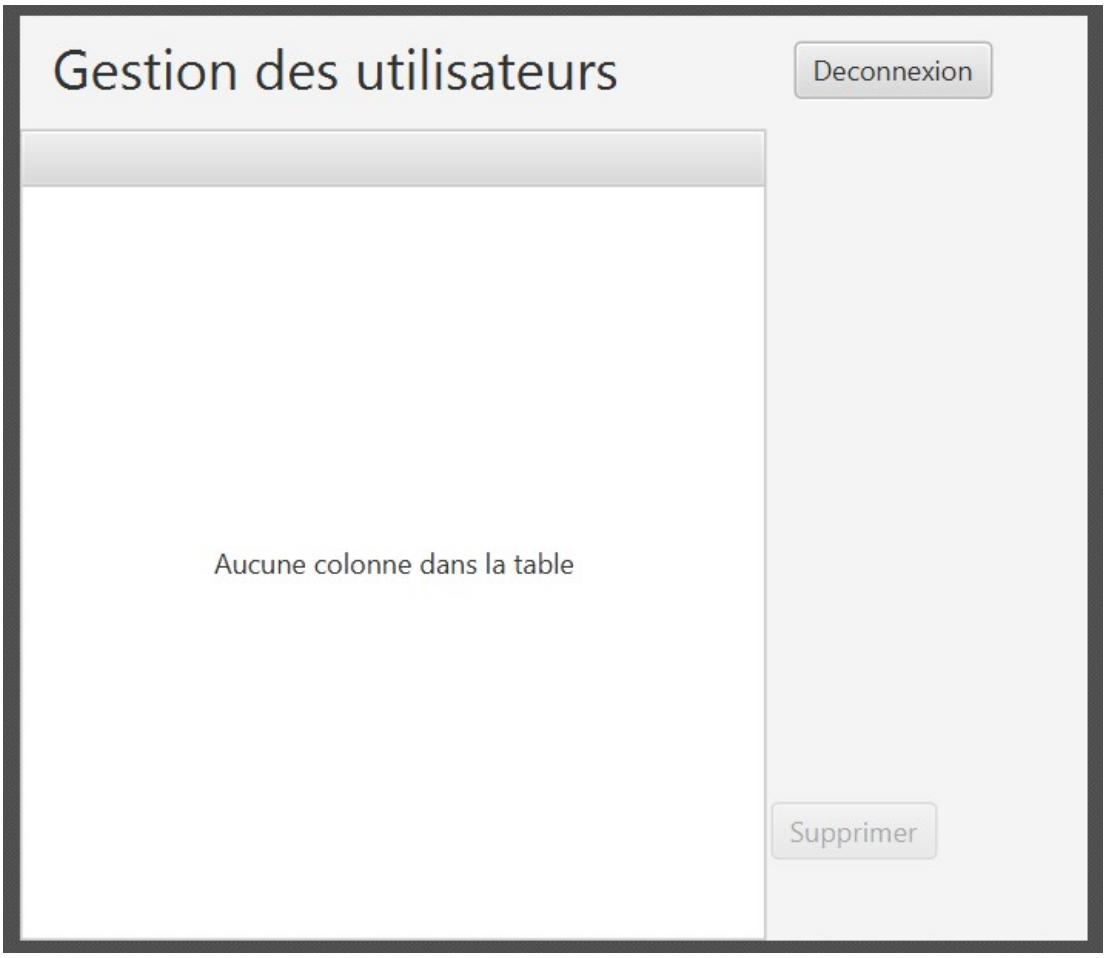
Nous allons mettre en place une TableView pour afficher des données dynamiques en JavaFX. Nous verrons comment créer un tableau, le lier à un modèle de données, ajouter des colonnes et gérer l'affichage des informations.

Objectif : Afficher une liste d'utilisateurs dans un TableView

Fichiers concernés :

- src/main/resources/views/appli/user/GestionUserView.fxml (Interface utilisateur)
- src/main/java/appli/user/GestionUserController.java (Logique du tableau)
- src/main/java/model/Utilisateur.java (Modèle de données)
- src/main/java/repository/UtilisateurRepository.java (Interface avec la base)

Un exemple d'interface ci-dessous



Création du TableView dans un fichier FXML

Étape 1 : Ajouter un TableView dans GestionUserView.fxml

- Ajouter à la page un TableView et retirer l'ensemble des colonnes présentes dans le tableau.
  - Nous allons les créer directement dans l'initialisation de notre tableau.
- Définissez le fx:id du tableau en « tableauUser »

Étape 2 : Déclarer les colonnes et les lier à notre modèle

L'interface Initializable

L'interface Initializable permet d'exécuter du code d'initialisation lorsque le contrôleur est chargé par JavaFX. Elle est utile pour **préparer des données avant l'affichage de la scène**.

Profitons-en pour typer notre tableauUser avec la classe Utilisateur

```
public class GestionUserController implements Initializable {
    @FXML
    private TableView<Utilisateur> tableauUser;
    [...]
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        // Code d'initialisation exécuté automatiquement au chargement de la vue
    }
}
```

Pourquoi l'utiliser ?

- Va nous permette **d'initialiser les colonnes** du tableau.
- Charge **des données dès l'ouverture de la scène** avant l'affichage.
- Évite d'avoir à appeler manuellement une méthode d'initialisation.

Création des colonnes

Nous allons créer dynamiquement les colonnes grâce à un tableau qui contiendra le nom des colonnes avec l'attribut liée dans utilisateur :

```
public class TableViewController implements Initializable {
    @FXML
    private TableView<Utilisateur> tableauUser;
    [...]
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        String [][] colonnes = {
            { "Id Utilisateur","idUser" },
            { "Nom","nom" },
            { "Prénom","prenom" },
            { "Email","mail" },
            { "Rôle","role" },
        }

        for ( int i = 0 ; i < colonnes.length ; i ++ ){
                //Création de la colonne avec le titre
                TableColumn<Utilisateur,String> maCol = new TableColumn<>(colonnes[i][0]);
                //Ligne permettant la liaison automatique de la cellule avec la propriété
                maCol.setCellValueFactory(
        new PropertyValueFactory<Utilisateur,String>(colonnes[i][1]));
                //Ajout de la colonne dans notre tableau
                tableauUser.getColumns().add(maCol);
        }
    }
}
```

Explication du code

Elément	Rôle
colonnes[][]	Tableau nous permettant de déclarer gérer les colonnes voulu dans notre tableView
TableColumn<Liste,String>	Création d'une colonne – Liste : un ensemble de valeur – String : Définit le type de la valeur affiché
maCol.setCellValueFactory	Lier la colonne a une propriété
New PropertyValueFactory	Créer une propriété avec l'attribut donné en paramètre
tableauUser.getColumns.add(maCol);	Ajoute la colonne au tableau

Afficher les utilisateurs

Il ne nous reste plus qu'à remplir notre tableau ! Pour se faire, il faut :

- UtilisateurRepository
- Récupérer l'ensemble des utilisateurs
- Ajouter les utilisateurs dans notre tableau
    - tableauUser.getItems().addAll(** lesUtilisateurs**)

Une tableView possède l'équivalent d'une ArrayList pour gérer les données qui est une ObservableList.

Pourquoi utiliser ObservableList ?

- Permet de **mettre à jour automatiquement l'interface** lorsque des données sont modifiées.
- Facilite l'ajout et la suppression d'éléments dans le tableau.

Activation du bouton de suppression

Etape 1 : Ajouter un bouton supprimer « disable » par défaut

- L'option se trouve dans la partie « properties » de votre bouton.
- Il devrait apparaitre en transparent sur l'interface
- Renseignez un Id au bouton ainsi qu'une méthode « onAction »
    - L'id pour nous permettre de l'activer / désactiver
    - Le « onAction » pour déclencher la suppression au clique

Etape 2 : Gérer la sélection d'un utilisateur

a) Sur SceneBuilder :

- Sélectionner votre TableView
- Dans la section « code », renseignez un nom de méthode correspondant à l'événement « On mouse Clicked »
- Récupérer la signature de la méthode et placez le dans le contrôleur de votre page
    - View > show Sample Controller Skeletton
    - Ex : « cliqueTableauEvent »

Pourquoi « On mouse Clicked » ?

- L'événement est déclenché à chaque clique fait dans le tableau
- Nous permettons de cloisonner l'événement du clique sur le tableau

b) Dans le corps de notre méthode

- Placez-y le code suivant

```
@FXML
void cliqueTableauEvent(MouseEvent event) {
    Utilisateur selection = tableauUser.getSelectionModel().getSelectedItem();
    if (selection != null) {
        btnSupprimer.setDisable(false);
    }else{
        btnSupprimer.setDisable(true);
    }
}
```

- **getSelectionModel().getSelectedItem()** : Permet de récupérer l'objet sélectionner dans le tableau
    - **Renvoie null si aucune ligne n'est sélectionnée**

En lançant le programme, maintenant, en sélectionant une ligne présent dans le tableau, le bouton supprimer devrait devenir cliquable !

Étape 3 : Suppression d'un utilisateur sélectionné

Complétez la méthode liée au bouton suppression en réalisant les étapes suivantes :

- Récupérer l'utilisateur sélectionnée
- Appeler votre méthode de suppression d'un utilisateur en base de données se trouvant dans le UtilisateurRepository
- En fonction de la réponse donnée par votre méthode (vrai si OK – false si erreur)
    - Supprimez l'utilisateur du TableView
        - tableauUser.getItems().remove(userSel)

Pourquoi cette approche ?

- Empêche la suppression si rien n'est sélectionné.
- Met automatiquement à jour l'affichage.

Modification d'un utilisateur via double-clic

Étape 1 : Détecter un double-clic sur une ligne

Préparation du travail

Complétez le code dans la méthode liée à l'événement du clic de votre tableView avec ce qui suit :

```
if (event.getClickCount() == 2) { // Vérifie si c'est un double-clic
        if (selection != null) {
            StartApplication.changeScene("user/modificationUser");
        }
    }
```

Nous allons devoir créer une nouvelle page se trouvant dans appli/user/ que l'on nommera :

- modificationUserView.fxml

Réaliser la page de modification d'un utilisateur avec les champs suivant :

- nom
- prenom
- email
- role

L'identifiant du compte et le mot de passe ne seront pas changé avec ce formulaire.

Envoi de l'utilisateur sélectionné à la nouvelle page

Dans notre nouveau controlleur nommé « modificationUserController » ajoutez-y la méthode suivante :

```
    public void initData(Utilisateur utilisateur) {
        this.utilisateurSel = utilisateur;
        nomField.setText(utilisateur.getNom());
        prenomField.setText(utilisateur.getPrenom());
        emailField.setText(utilisateur.getEmail());
        roleField.setText( utilisateur.getRole());
    }
```

Cette méthode a pour but de récupérer l'utilisateur sélectionné et d'y placer les informations directement dans le formulaire

Pour appeler cette méthode, il va falloir modifier notre StartApplication et lui ajouter une nouvelle méthode :

```
public static Object getControllerFromStage(){
    return mainStage.getScene().getUserData();
}
```

Que fait cette méthode ?

- A partir de notre Scene, elle nous renvoie le contrôleur lié à notre page.
- Ce qui va nous permettre d'appeler des méthodes provenant du Controller de notre page.

Modification supplémentaire dans StartApplication

Durant la première ligne de notre nouvelle méthode, nous récupérons le FXMLLoader venant de la page. Or, une fois chargé, nous ne sauvegardons pas cet élément, il nous nécessaire de le sauvegarder. Modifier la méthode changeScene comme suit :

```
public static void changeScene(String nomDuFichierFxml ) throws IOException {
    FXMLLoader fxmlLoader = new
FXMLLoader(StartApplication.class.getResource(nomDuFichierFxml+"View.fxml"));
    Scene scene = new Scene(fxmlLoader.load());
    scene.setUserData(fxmlLoader.getController());
    mainStage.setScene(scene);
}
```

Nous plaçons notre contrôleur dans un espace d'échange liée à notre Scene.

Appel de la méthode dans notre application

Nous allons compléter notre méthode pour appeler initData de notre controlleur au moment de notre changement de Scene :

```
if (event.getClickCount() == 2) { // Vérifie si c'est un double-clic
        if (selection != null) {
            StartApplication.changeScene("user/modificationUser");
            ModificationUserController controller = (ModificationUserController)
StartApplication. getControllerFromStage();
            controller.initData(utilisateur);
        }
    }
```

(ModificationUserController) StartApplication. getControllerFromStage();

- Permet de « Caster » / forcer le type d'objet que nous renvoie le getControllerFromStage() correspondant à la classe liée à notre page

Gérer la modification

Compléter la page « modificationUser » ainsi que son controller pour réaliser la modification de l'utilisateur et le renvoyer à la page de gestion des utilisateur une fois terminé.

Tâches à effectuer

- Activer le bouton de suppression lorsqu'un utilisateur est sélectionné.
- Gérer la suppression de la ligne sélectionnée.
- Détecter un double-clic pour ouvrir la fenêtre de modification.
- Créer modificationUser.fxml et son contrôleur.
- Tester la modification et la mise à jour des données.