



APPLICATION ADMINISTRATIVE

DOCUMENTATION TECHNIQUE

Projet E4

Java / Java FX

Cheik-Siramakan Keita

cheiksiramakankeita@gmail.com



Table des matières

Cahier des charges	2
Contexte	2
Prestation attendu.....	2
Outils utilisé	3
Base de données	4
Liste des tables	4
MLD	4
Connexion à la base de données.....	5
Arborescence du projet.....	6
Les classes métier.....	7
Data Acces Object	8
Vues.....	8
Créer une vue.....	9
Controllers.....	10
Redirection vers une autre page	10
Attribuer une action a un bouton	10
Affichage d'un tableau contenant des données	10
Recherche.....	12
Inscription	13
Modification	14
Sélectionner une ligne dans un tableau.....	14
Suppression.....	16
Déplacer la fenêtre.....	17
Statistiques.....	18
Erreurs.....	19

CAHIER DES CHARGES

CONTEXTE

Le Lycée du Parc de Vilegénis a fait appel à la société ZenMedia, une SSII afin de lui développer une application lourde de gestion des données d'une application web, ces applications marcheront en conjonction. Cette application devra permettre de gérer différents utilisateurs ainsi que les offres, il sera aussi possible de voir des données statistiques et seuls les administrateurs auront accès à cette application.

PRESTATION ATTENDUE

Dans un premier temps l'administrateur arrivera sur la page de connexion, sur celle-ci il entrera son email et un mot de passe pour pouvoir accéder à la page suivante. En plus du formulaire de connexion sur cette page se trouve un bouton pour pouvoir accéder à la page de configuration de la connexion à une base de données et d'activation de l'envoi d'emails.

Premier module : Configuration

Cette page permettra de modifier les informations de connexion à la base de données, l'administrateur réseau informera les utilisateurs de ce qu'il faut entrer sur cette page.

Deuxième module : Menu principal

Dans cette page l'administrateur connecté aura plusieurs options, il pourra gérer soit les jeunes, partenaires, les offres et voir des statistiques d'utilisation.

Troisième module : Gestion des jeunes

Dans ce menu l'administrateur trouvera une liste de tous les jeunes utilisateurs et aura plusieurs options à sa disposition tel que la recherche plus spécifique d'un utilisateur grâce à la barre de recherche. Les autres options incluent l'inscription, la modification et la suppression.

Quatrième module : Gestion des partenaires

Comme pour les jeunes dans ce menu l'administrateur trouvera une liste de tous les partenaires utilisateurs et aura plusieurs options à sa disposition tel que la recherche plus spécifique d'un utilisateur grâce à la barre de recherche. Les autres options incluent l'inscription, la modification et la suppression.

Cinquième module : Gestion des offres

Dans ce menu l'administrateur trouvera une liste de toutes les offres, il aura plusieurs options comme pour les deux précédents modules à part qu'il ne pourra pas créer d'offre ou modifier les informations de celles-ci. Il pourra par contre lire la description d'une offre qu'il pourra sélectionner dans la liste et il pourra décider de supprimer cette offre s'il y a un défaut.

Sixième module : Statistiques

Dans ce menu l'administrateur pourra voir des statistiques sur les offres postées sur le site web telle que le pourcentage de formation posté et le pourcentage des offres posté par les entreprises.

Septième module : Gestion des administrateurs

Ce module est un peu spécial il ne sera accessible que par un certain type d'administrateurs appelé « Super Administrateur ». Cet administrateur aura le droit de modifier les informations de n'importe quel administrateur et pourra même modifier leur statut d'administrateur basique en super administrateur.

OUTILS UTILISE

Pour ce projet j'ai utilisé Eclipse comme IDE, MySQL pour la base de données, SceneBuilder pour créer l'interface graphique et Git comme gestionnaire de version.

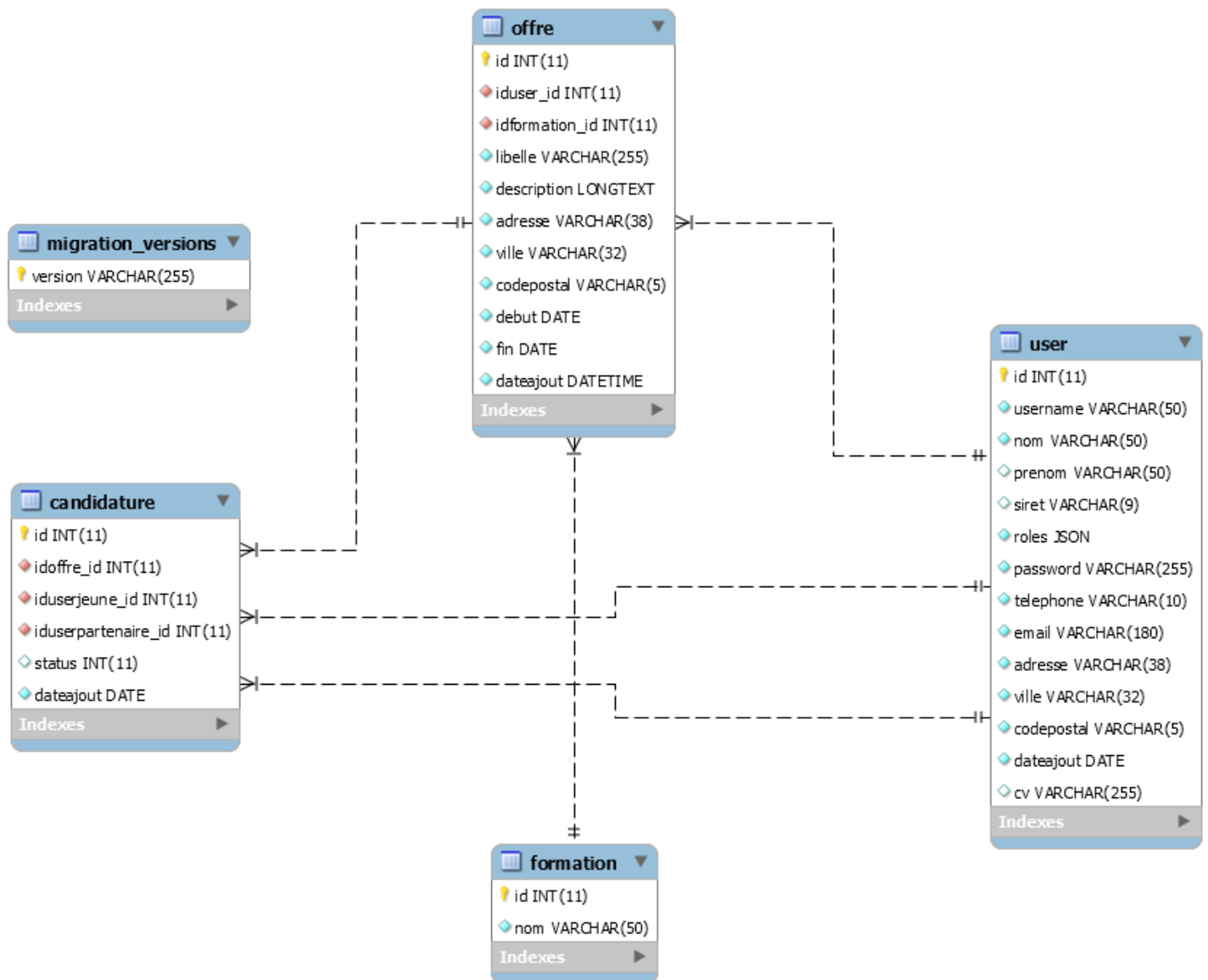


BASE DE DONNEES

LISTE DES TABLES

Table	Action	Rows	Type	Collation	Size	Overhead
candidature		5	InnoDB	utf8mb4_unicode_ci	64 KiB	-
formation		3	InnoDB	utf8mb4_unicode_ci	16 KiB	-
migration_versions		0	InnoDB	utf8_unicode_ci	16 KiB	-
offre		75	InnoDB	utf8mb4_unicode_ci	96 KiB	-
user		85	InnoDB	utf8mb4_unicode_ci	64 KiB	-
5 table(s)	Sum	168	InnoDB	latin1_swedish_ci	256 KiB	0 B

MLD



ARBORESCENCE DU PROJET

Le package « application » contient la classe principale qui lance l'application, le package « controllers » contrôle les actions liées aux vues, le package « models.base » contient les classes métier. Le package « models.dao » contient les classes utilisées pour manipuler les données contenues dans la base données. Dans le package « views.fxml » se trouvent les fichiers utilisés pour l'interface graphique.

- ▼ ppe1 [ppe1 master]
 - ▼ src
 - ▼ application
 - > Main.java
 - application.css
 - ▼ controllers
 - > AdministrateurController.java
 - > AdministrateurInscriptionController.java
 - > AdministrateurModificationController.java
 - > ConfigurationController.java
 - > ConnexionController.java
 - > JeuneController.java
 - > JeuneInscriptionController.java
 - > JeuneModificationController.java
 - > MenuController.java
 - > OffreController.java
 - > OffreDescriptionController.java
 - > PartenaireController.java
 - > PartenaireInscriptionController.java
 - > PartenaireModificationController.java
 - > StatistiqueController.java
 - > SuperAdministrateurMenuController.java
 - > doc
 - > library
 - ▼ library.encryption
 - > BCrypt.java
 - > MotDePasse.java
 - > Salt.java
 - ▼ models.base
 - > Formation.java
 - > Offre.java
 - > User.java
 - ▼ models.dao
 - > ConfigurationConnexionBaseDeDonneesDAO.java
 - > Connect.java
 - > OffreDAO.java
 - > UserDAO.java
 - > views.css
 - ▼ views.fxml
 - Administrateur.fxml
 - AdministrateurInscription.fxml
 - AdministrateurModification.fxml
 - Configuration.fxml
 - Connexion.fxml
 - Jeune.fxml
 - JeuneInscription.fxml
 - JeuneModification.fxml
 - Menu.fxml
 - Offre.fxml

LES CLASSES METIER

Avec l'utilisation de JavaFX les classes métier changent et n'utilisent plus les types qui définissent les variables habituelles. Les integer et les String sont remplacé par IntegerProperty et StringProperty.

Par exemple la classe « User » :

```
public class User
{
    private IntegerProperty id;
    private StringProperty roles;
    private StringProperty username;
    private StringProperty nom;
    private StringProperty prenom;
    private IntegerProperty siret;
    private StringProperty password;
    private StringProperty telephone;
    private StringProperty email;
    private StringProperty adresse;
    private StringProperty ville;
    private StringProperty codepostal;
    private StringProperty dateajout;
    public static final Pattern VALID_EMAIL_ADDRESS_REGEX = Pattern.compile("^[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z]{2,6}$", Pattern.CASE_INSENSITIVE);

    public User()
    {
        this.id = new SimpleIntegerProperty();
        this.roles = new SimpleStringProperty();
        this.username = new SimpleStringProperty();
        this.nom = new SimpleStringProperty();
        this.prenom = new SimpleStringProperty();
        this.siret = new SimpleIntegerProperty();
        this.password = new SimpleStringProperty();
        this.telephone = new SimpleStringProperty();
        this.email = new SimpleStringProperty();
        this.adresse = new SimpleStringProperty();
        this.ville = new SimpleStringProperty();
        this.codepostal = new SimpleStringProperty();
        this.dateajout = new SimpleStringProperty();
    }

    public void setIdProp(IntegerProperty administrateur_id){this.id = administrateur_id;}
    public void setRolesProp(StringProperty roles){this.roles = roles;}
    public void setUsernameProp(StringProperty administrateur_super) {this.username = administrateur_super;}
    public void setNomProp(StringProperty nom){this.nom = nom;}
    public void setPrenomProp(StringProperty prenom){this.prenom = prenom;}
    public void setSiretProp(IntegerProperty siret){this.siret = siret;}
    public void setPasswordProp(StringProperty password){this.password = password;}
    public void setEmailProp(StringProperty email){this.email = email;}
    public void setTelephoneProp(StringProperty telephone){this.telephone = telephone;}
    public void setadresseProp(StringProperty adresse){this.adresse = adresse;}
    public void setvilleProp(StringProperty ville){this.ville = ville;}
    public void setcodepostalProp(StringProperty codepostal){this.codepostal = codepostal;}
    public void setDateajoutProp(StringProperty dateajout){this.dateajout = dateajout;}

    public IntegerProperty getIdProp(){return this.id;}
    public StringProperty getRolesProp(){return this.roles;}
    public StringProperty getUsernameProp(){return username;}
    public StringProperty getPrenomProp(){return prenom;}
    public StringProperty getNomProp(){return nom;}
    public IntegerProperty getSiretProp(){return siret;}
    public StringProperty getPasswordProp(){return password;}
```

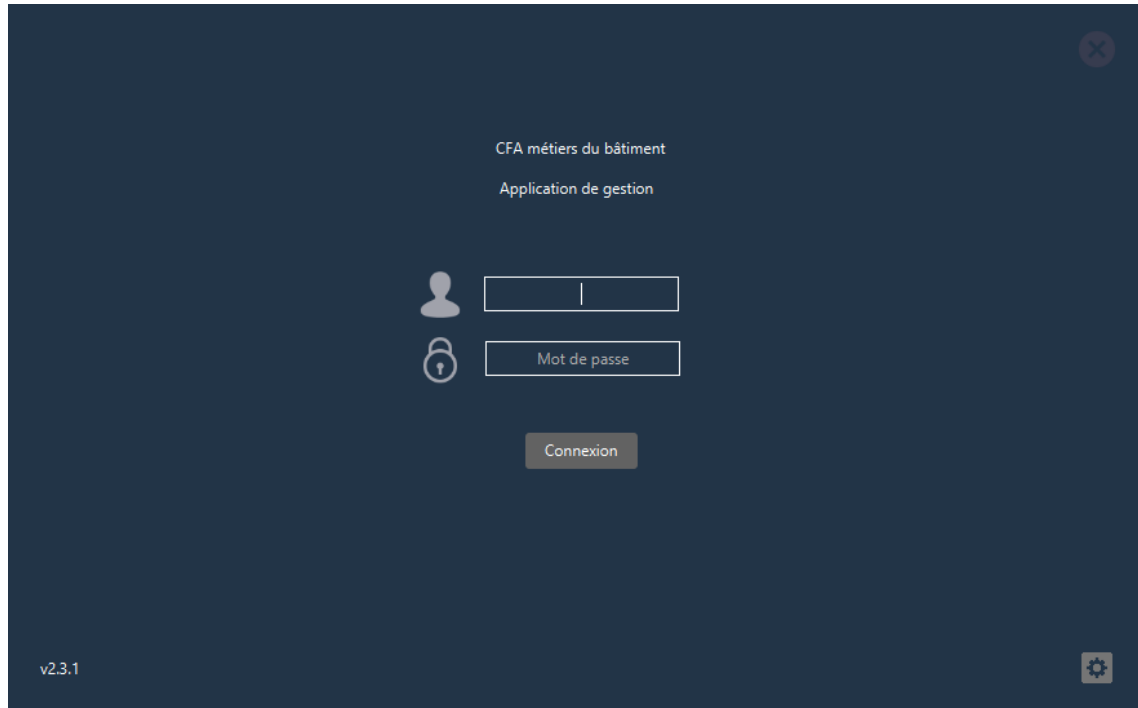

Dans la vue il faut spécifier le controller à utiliser en l'indiquant dans le « AnchorPane ».

```
fx:controller="controllers.ConnexionController";
```

Les actions de chaque champ est spécifié avec « onAction ».

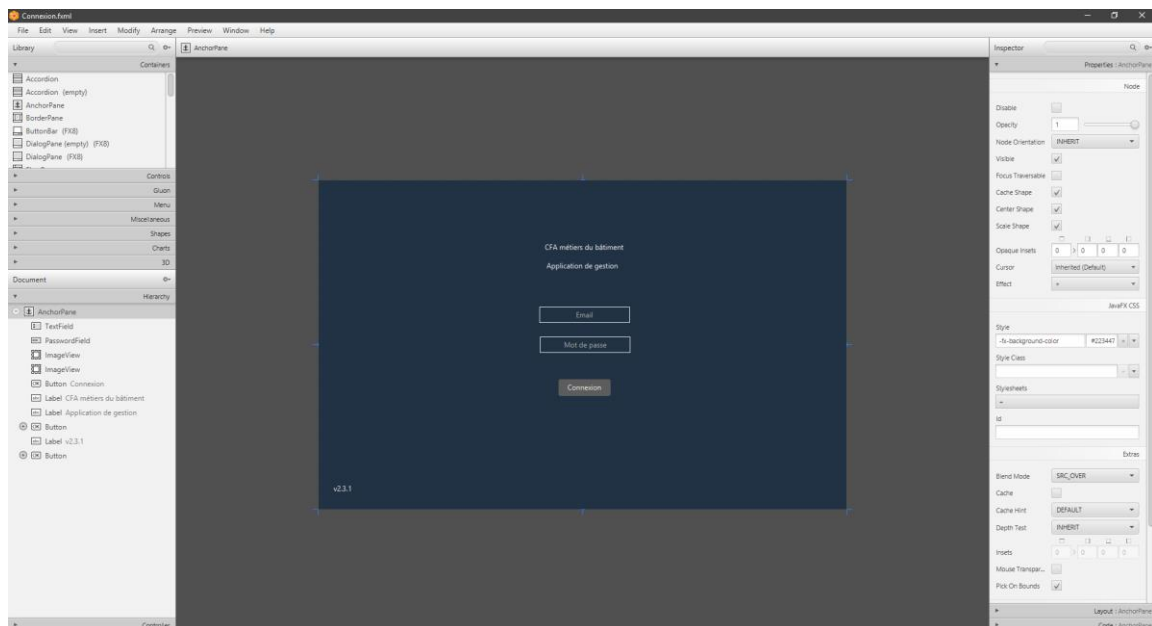
```
onAction="#connecter_un_utilisateur"
```

Ce que le code de la page « connexion.fxml » donne :



CREER UNE VUE

Pour créer une vue j'utilise l'application SceneBuilder et configure chaque aspect l'interface graphique dedans, le positionnement de chaque éléments ainsi que leur couleur.



CONTROLLERS

Les controllers contrôlent les actions des vues ainsi que les données qui y sont transféré, chaque vue à un controller lié à celui-ci. Dans le controller nous déclarons les éléments qui se trouvent dans l'interface graphique auxquels nous avons donné un id.

```
public class ConnexionController {
    @FXML
    private Stage main;
    @FXML
    private AnchorPane mainPane;
    @FXML
    private TextField emailInput;
    @FXML
    private PasswordField motDePasseInput;
    @FXML
    private Button connexionButton;
    @FXML
    private Button configurationButton;
    @FXML
    private Button fermetureButton;
```

Chaque déclaration faite dans un controller est précédé par l'annotation @FXML même les fonctions.

REDIRECTION VERS UNE AUTRE PAGE

ATTRIBUER UNE ACTION A UN BOUTON

Lorsqu'une personne se connecte cliquer sur le bouton ayant l'id « connexionButton » fait que l'action utilisé est « connecter_un_utilisateur ».

```
onAction="#connecter_un_utilisateur"
```

Dans le controller « ConnexionController » se trouve la fonction « connecter_un_utilisateur » qui est l'action effectué après avoir cliqué sur le bouton de connexion ayant l'id « connexionButton ». « onAction » indique quelle fonction doit être appelé.

```
@FXML
private void connecter_un_utilisateur(ActionEvent actionEvent)
```

Le code dans la capture d'écran effectue la redirection vers la page « Menu.fxml ».

```
mainPane.getChildren().clear();
FXMLLoader loader = new FXMLLoader();
loader.setLocation(Main.class.getClassLoader().getResource("views/fxml/Menu.fxml"));
AnchorPane userFrame = (AnchorPane) loader.load();
Scene sc = new Scene(userFrame);
Stage stage = (Stage) mainPane.getScene().getWindow();
stage.setScene(sc);
MenuController menuController = loader.<MenuController>getController();
menuController.recuperer_le_nom_de_la_personne_connecte(nomDeLaPersonneConnecte);
menuController.recuperer_le_status_super_administrateur_de_la_personne_connecte(roles);
```

AFFICHAGE D'UN TABLEAU CONTENANT DES DONNEES

Lorsque l'on veut afficher un tableau dans le fichier fxml il faut utiliser la balise « TableView », pour cet exemple nous utilisons « Jeune.fxml » :

```
<TableView fx:id="table" layoutX="15.0" layoutY="110.0" prefHeight="428.0" prefWidth="853.0">
    <columns>
        <TableColumn fx:id="nomColumn" prefWidth="115.0" text="Nom" />
        <TableColumn fx:id="prenomColumn" prefWidth="115.0" text="PrÃ©nom" />
        <TableColumn fx:id="usernameColumn" prefWidth="117.0" text="Username" />
        <TableColumn fx:id="emailColumn" prefWidth="185.0" text="Email" />
        <TableColumn fx:id="telephoneColumn" prefWidth="161.0" text="TÃ©lÃ©phone" />
        <TableColumn fx:id="creationColumn" prefWidth="159.0" text="CrÃ©ation" />
    </columns>
    <effect>
        <Blend />
    </effect>
</TableView>
```

Dans le controller qui est « JeuneController » il faut déclarer le tableau et ses colonnes comme ceci :

```
@FXML
private TableView<User> table;
@FXML
private TableColumn<User, Integer> idColumn;
@FXML
private TableColumn<User, String> nomColumn;
@FXML
private TableColumn<User, String> prenomColumn;
@FXML
private TableColumn<User, String> usernameColumn;
@FXML
private TableColumn<User, String> emailColumn;
@FXML
private TableColumn<User, String> telephoneColumn;
@FXML
private TableColumn<User, String> creationColumn;
```

Dans la fonction « initialize() » qui applique le code qui est dedans dès que la page est appelé il faut attribuer les données récupérées par la requête SQL aux colonnes du tableau.

```
@FXML
private void initialize() throws ClassNotFoundException, SQLException
{
    ObservableList<User> empData = UserDAO.obtenir_la_liste_de_tout_les_utilisateur_d_un_certain_role(roles_jeune);
    table.setItems(empData);
    nomColumn.setCellValueFactory(cellData -> cellData.getValue().getNomProp());
    prenomColumn.setCellValueFactory(cellData -> cellData.getValue().getPrenomProp());
    usernameColumn.setCellValueFactory(cellData -> cellData.getValue().getUsernameProp());
    emailColumn.setCellValueFactory(cellData -> cellData.getValue().getEmailProp());
    telephoneColumn.setCellValueFactory(cellData -> cellData.getValue().getTelephoneProp());
    creationColumn.setCellValueFactory(cellData -> cellData.getValue().getDateajoutProp());
}
```

La variable « empData » récupère les informations en appelant la fonction « obtenir_la_liste_de_tout_les_utilisateur_d_un_certain_role() ». Cette fonction retourne une liste contenant des instances de la classe « User ».

```
public static ObservableList<User> obtenir_la_liste_de_tout_les_utilisateur_d_un_certain_role(String [] roles) throws ClassNotFoundException, SQLException {
    Connection connexion = Connect.getInstance().getConnection();
    String requete = "SELECT id, roles, siret, nom, prenom, username, email, telephone, adresse, ville, codepostal, dateajout FROM user WHERE roles LIKE ?";

    ObservableList<User> users = FXCollections.observableArrayList();

    int id;
    String role;
    String siret;
    String nom;
    String prenom;
    String email;
    String telephone;
    String adresse;
    String ville;
    String code_postal;
    String creation;

    PreparedStatement prepared_statement = connexion.prepareStatement(requete);
    prepared_statement.setString(1, "%" + roles[0] + "%");

    ResultSet resultat = prepared_statement.executeQuery();

    while(resultat.next())
    {
        id = resultat.getInt("id");
        role = resultat.getString("roles");
        siret = resultat.getString("siret");
        nom = resultat.getString("nom");
        prenom = resultat.getString("prenom");
        email = resultat.getString("email");
        telephone = resultat.getString("telephone");
        adresse = resultat.getString("adresse");
        ville = resultat.getString("ville");
        code_postal = resultat.getString("codepostal");
        creation = resultat.getString("dateajout");

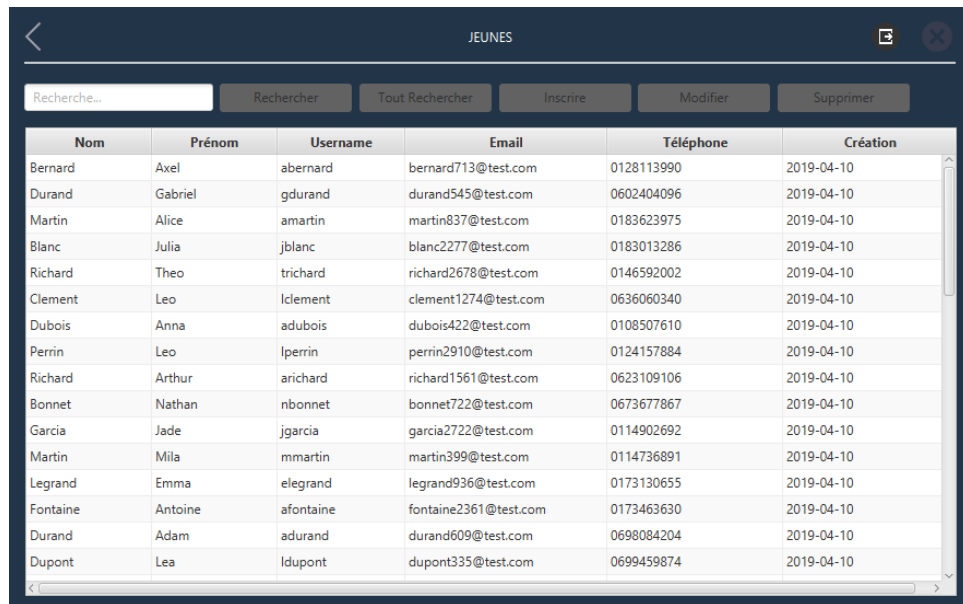
        User user = new User();

        user.setId(id);
        user.setRoles(role);
        if(siret != null) {
            user.setSiret(Integer.parseInt(siret));
        }
        user.setNom(nom);
        user.setPrenom(prenom);
        user.setUsername("");
        user.setEmail(email);
        user.setTelephone(telephone);
        user.setAdresse(adresse);
        user.setVille(ville);
        user.setCodepostal(code_postal);
        user.setDateajout(creation);

        users.add(user);
    }

    resultat.close();
    prepared_statement.close();
    connexion.close();
    return users;
}
```

Voici ce que ça donne :



Nom	Prénom	Username	Email	Téléphone	Création
Bernard	Axel	abernard	bernard713@test.com	0128113990	2019-04-10
Durand	Gabriel	gdurand	durand545@test.com	0602404096	2019-04-10
Martin	Alice	amartin	martin837@test.com	0183623975	2019-04-10
Blanc	Julia	jblanc	blanc2277@test.com	0183013286	2019-04-10
Richard	Theo	trichard	richard2678@test.com	0146592002	2019-04-10
Clement	Leo	lclement	clement1274@test.com	0636060340	2019-04-10
Dubois	Anna	adubois	dubois422@test.com	0108507610	2019-04-10
Perrin	Leo	lperrin	perrin2910@test.com	0124157884	2019-04-10
Richard	Arthur	arichard	richard1561@test.com	0623109106	2019-04-10
Bonnet	Nathan	nbonnet	bonnet722@test.com	0673677867	2019-04-10
Garcia	Jade	kgarcia	garcia2722@test.com	0114902692	2019-04-10
Martin	Mila	mmartin	martin399@test.com	0114736891	2019-04-10
Legrand	Emma	elegrand	legrand936@test.com	0173130655	2019-04-10
Fontaine	Antoine	afontaine	fontaine2361@test.com	0173463630	2019-04-10
Durand	Adam	adurand	durand609@test.com	0698084204	2019-04-10
Dupont	Lea	ldupont	dupont335@test.com	0699459874	2019-04-10

RECHERCHE

La fonctionnalité de recherche utilise champ de type « TextField » pour savoir ce que la personne veut chercher et un autre champ de type « Button » pour exécuter la recherche et appeler la fonction nécessaire.

Comme pour l’affichage de données dans un tableau il faut placer les balises dans le fichier fxml et l’action appelé par le bouton est « recherche_filtre » comme ceci :

```
<Button fx:id="rechercheFiltreButton"
<TextField fx:id="rechercheInput" layc
onAction="#recherche_filtre"
```

Dans le controller qui est « JeuneController » nous déclarons les champs que nous utilisons comme ceci :

```
@FXML
private Button rechercheFiltreButton;

@FXML
private TextField rechercheInput;
```

Dans la fonction « recherche_filtre() » nous faisons appel à la fonction « obtenir_la_liste_filtre() » pour obtenir les données contenues dans la base de données puis nous mettons à jour le contenu du tableau dans la ligne « table.setItems() » comme ceci :

```
@FXML
private void recherche_filtre(ActionEvent actionEvent) throws SQLException, ClassNotFoundException, IOException
{
    ObservableList<User> listeFiltreDesJeunes = UserDAO.obtenir_la_liste_filtre(rechercheInput.getText(), this.roles_jeune);
    table.setItems(listeFiltreDesJeunes);
}
```

Voici ce que cela donne :

[illegible]

INSCRIPTION

L'inscription utilise la récupération de données contenu dans les champs du fichier fxml et l'insertion dans la base de données via une requête SQL.

Pour l'inscription dans le fichier fxml il faut placer les balises dont nous avons besoin, par exemple l'inscription d'un jeune utilisateur.

```
<Button fx:id="inscriptionButton" layoutX="381.0" layoutY="464.0" mnemonicParsing="false" onAction="#inscrive_un_
<TextField fx:id="nomInput" alignment="CENTER" layoutX="365.0" layoutY="148.0" promptText="Nom" style="-fx-backg
<TextField fx:id="prenomInput" alignment="CENTER" layoutX="365.0" layoutY="182.0" promptText="Pr nom" style="-fx
<TextField fx:id="motDePasseInput" accessibleRole="PASSWORD_FIELD" alignment="CENTER" layoutX="365.0" layoutY="21
<TextField fx:id="emailInput" alignment="CENTER" layoutX="365.0" layoutY="255.0" promptText="Email" style="-fx-ba
<TextField fx:id="telephoneInput" alignment="CENTER" layoutX="365.0" layoutY="293.0" promptText="T l phone" st
<TextField fx:id="adresseInput" alignment="CENTER" layoutX="365.0" layoutY="330.0" promptText="Adresse" style="-f
<TextField fx:id="villeInput" alignment="CENTER" layoutX="365.0" layoutY="367.0" promptText="Ville" style="-fx-ba
<TextField fx:id="codePostalInput" alignment="CENTER" layoutX="366.0" layoutY="405.0" promptText="Code postal" st
```

Dans le controller nous déclarons les champs dont nous avons besoin avec leur id comme ceci :

```
@FXML
private Button inscriptionButton;
@FXML
private TextField nomInput;
@FXML
private TextField prenomInput;
@FXML
private TextField identifiantInput;
@FXML
private TextField motDePasseInput;
@FXML
private TextField emailInput;
@FXML
private TextField telephoneInput;
@FXML
private TextField adresseInput;
@FXML
private TextField villeInput;
@FXML
private TextField codePostalInput;
```

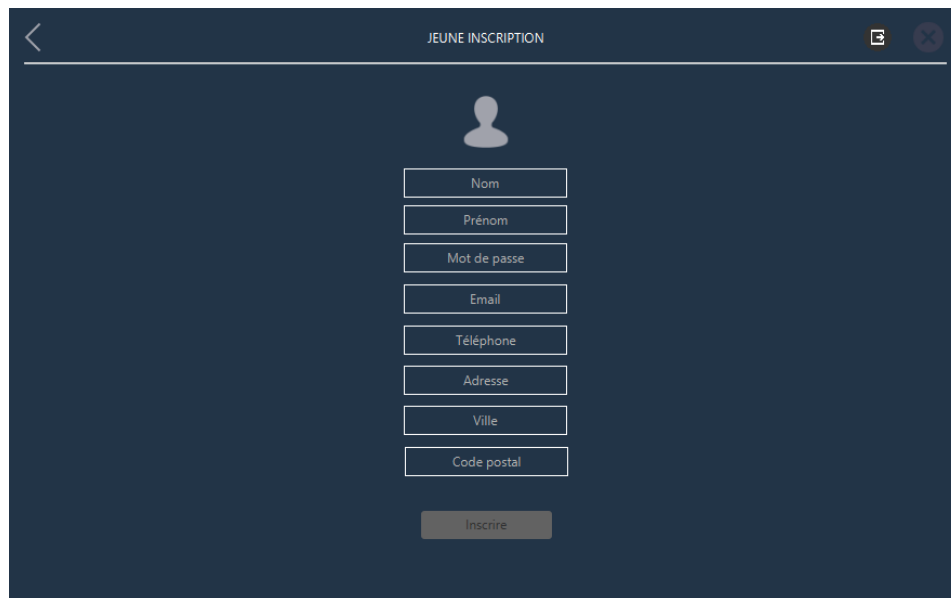
La fonction appelé par le clique du bouton d'inscription est « inscrire_un_nouveau_jeune() ». Dans cette fonction je récupère les informations des champs rempli avec « getText() ».

```
User jeune = new User();
jeune.setNom(nomInput.getText());
jeune.setPrenom(prenomInput.getText());
jeune.setUsername();
jeune.setRoles(roles);
jeune.setPassword(hash);
jeune.setEmail(emailInput.getText());
jeune.setTelephone(telephoneInput.getText());
jeune.setAdresse(adresseInput.getText());
jeune.setVille(villeInput.getText());
jeune.setCodepostal(codePostalInput.getText());
```

J'insère les informations en appelant la fonction dont j'ai besoin qui se trouve dans une des classes DAO.

```
boolean empdata = UserDAO.inscrire(jeune);
```

Voici ce que cela donne :



MODIFICATION

SELECTIONNER UNE LIGNE DANS UN TABLEAU

La modification des informations d'un utilisateur fonctionne en récupérant les informations de celui-ci dans la base de données. Une fois ces données récupérées nous les plaçons dans les champs comme nous le souhaitons.

Pour expliquer comment la modification des données fonctionne nous allons utiliser la modification des données d'un jeune, tout d'abord dans le controller de la page précédente qui est « JeuneController ». Dans la fonction « acceder_a_la_page_de_modification_des_informations_du_jeune_selectionne » qui gère le passage vers la page de modification je récupère les informations de la ligne du tableau sur laquelle l'utilisateur a cliqué.

```
if(table.getSelectionModel().getSelectedItem() != null)
```

Voici à quoi ressemble la sélection d'un utilisateur à modifier :

Nom	Prénom	Username	Email	Téléphone	Création
Bernard	Axel	abernard	bernard713@test.com	0128113990	2019-04-10
Durand	Gabriel	gdurand	durand545@test.com	0602404096	2019-04-10

Dans cette même fonction je récupère les informations de l'utilisateur que j'ai sélectionné et je passe ces informations au contrôleur de la page de modification des informations des jeunes. Chaque ligne du tableau correspond à une instance de la classe « User », je récupère les informations et les place dans des variables en utilisant les fonctions « get() » de la classe métier. Après avoir placé ces données dans des variables je les passe au contrôleur qui gère la modification via l'appel de la fonction « obtenir_les_informations_du_jeune_a_modifie() » en plaçant les données à faire passer autant que paramètres de cette fonction.

```
User jeune = table.getSelectionModel().getSelectedItem();

int id = jeune.getId();
String nom = jeune.getNom();
String prenom = jeune.getPrenom();
String email = jeune.getEmail();
String telephone = jeune.getTelephone();
String adresse = jeune.getAdresse();
String ville = jeune.getVille();
String code_postal = jeune.getCodepostal();

try
{
    mainPane.getChildren().clear();
    FXMLLoader loader = new FXMLLoader();
    loader.setLocation(Main.class.getResource("views/fxml/JeuneModification.fxml"));
    AnchorPane userFrame = (AnchorPane) loader.load();
    Scene sc = mainPane.getScene();
    sc.setRoot(userFrame);
    JeuneModificationController jeuneModificationController = loader.<JeuneModificationController>getController();
    jeuneModificationController.obtenir_les_informations_du_jeune_a_modifier(id, nom, prenom, email, telephone, adresse, ville, code_postal);
    jeuneModificationController.recuperer_le_nom_de_la_personne_connectee(this.nomDeLaPersonneConnectee);
    jeuneModificationController.recuperer_le_status_super_administrateur_de_la_personne_connectee(this.roles);
}
```

Dans Le contrôleur de la page de modification qui est « JeuneModificationController » je place les données dans les champs avec la fonction « setText() » dans la fonction « obtenir_les_informations_du_jeune_a_modifie() ».

```
public void obtenir_les_informations_du_jeune_a_modifier(int idJeuneModification, String nom, String prenom, String email,
    String telephone, String adresse, String ville, String code_postal)
{
    this.idJeuneModification = idJeuneModification;
    idLabel.setText("ID: " + Integer.toString(idJeuneModification));
    nomInput.setText(nom);
    prenomInput.setText(prenom);
    emailInput.setText(email);
    telephoneInput.setText(telephone);
    adresseInput.setText(adresse);
    villeInput.setText(ville);
    codePostalInput.setText(code_postal);
}
```

Ces manipulations donnent ceci :

The screenshot shows a mobile application interface titled "JEUNE MODIFICATION". At the top, there is a back arrow, the title, and icons for a list and a close button. Below the title, there is a user profile icon and the text "ID: 36". The main part of the screen contains a form with the following fields and values:

Nom	Bernard
Prénom	Axel
Email	bernard713@test.com
Téléphone	0128113990
Adresse	506 Rue Pasteur Bordeaux
Ville	Bordeaux
Code postal	33

At the bottom of the form, there is a button labeled "Modification".

Une fois les modifications effectuées, dans le contrôleur la fonction « `modifier_les_informations_du_jeune()` » est appelée lorsque le bouton de modification est cliqué. Cette fonction vérifie les informations récupérées, crée une instance de la classe « `User` » avec ces informations et fait appel à la fonction qui contient la requête SQL.

```
String roles = "[\\\"ROLE_JEUNE\\\"]";
User user = new User();
user.setId(Integer.parseInt(idLabel.getText().substring(4, 6)));
user.setRoles(roles);
user.setNom(nomInput.getText());
user.setPrenom(prenomInput.getText());
user.setUsername();
user.setEmail(emailInput.getText());
user.setTelephone(telephoneInput.getText());
user.setAdresse(adresseInput.getText());
user.setVille(villeInput.getText());
user.setCodepostal(codePostalInput.getText());

boolean statusModificationInformationsDuJeune = UserDAO.modifier(user);
```

La fonction qui effectue la requête SQL ressemble à ça.

```
public static boolean modifier(User user) throws SQLException {
    Connection connexion = Connect.getInstance().getConnection();
    String requete = "UPDATE user SET siret = ?, roles = ?, nom = ?, prenom = ?, username = ?, email = ?, telephone = ?, adresse = ?, ville = ?, codepostal = ?"
        + "WHERE id = ?";

    //String [] roles = user.getRoles();
    String role = user.getRoles();
    boolean retour = false;

    PreparedStatement prepared_statement = null;
    prepared_statement = connexion.prepareStatement(requete);
    prepared_statement.setString(1, Integer.toString(user.getSiret()));
    prepared_statement.setString(2, role);
    prepared_statement.setString(3, user.getNom());
    prepared_statement.setString(4, user.getPrenom());
    prepared_statement.setString(5, user.getUsername());
    prepared_statement.setString(6, user.getEmail());
    prepared_statement.setString(7, user.getTelephone());
    prepared_statement.setString(8, user.getAdresse());
    prepared_statement.setString(9, user.getVille());
    prepared_statement.setString(10, user.getCodepostal());
    prepared_statement.setInt(11, user.getId());
```

SUPPRESSION

Dans les pages de gestion des utilisateurs partenaire ou encore jeune, l'administrateur peut supprimer leurs informations en les [sélectionnant dans le tableau](#) puis en cliquant sur le [bouton](#) supprimer.

```
@FXML
private void supprimer_un_jeune(ActionEvent actionEvent) throws SQLException, ClassNotFoundException
{
    if(table.getSelectionModel().getSelectedItem() != null)
    {
        User jeuneSelectionne = table.getSelectionModel().getSelectedItem();
        int id = jeuneSelectionne.getId();
        UserDAO.supprimer(id);
    }
}
```

Recherche...	Rechercher	Tout Rechercher	Inscrire	Modifier	Supprimer
Nom	Prénom	Username	Email	Téléphone	Création
Bernard	Axel	abernard	bernard713@test.com	0128113990	2019-04-10
Durand	Gabriel	gdurand	durand545@test.com	0602404096	2019-04-10

Dans le controller la fonction appelé par le bouton supprimer est « `supprimer_un_jeune()` » dedans nous récupérons l'id de l'utilisateur sélectionné puis nous faisons appel à la fonction qui effectuera la requête SQL dans ce cas il s'agit de la fonction « `supprimer` » dans la classe « `UserDAO` ».

```
public static boolean supprimer(int id) throws SQLException, ClassNotFoundException {
    Connection connexion = Connect.getInstance().getConnection();
    String requeteSQL = "DELETE FROM user WHERE id = ?";

    boolean reponse = false;

    PreparedStatement prepared_statement = null;
    prepared_statement = connexion.prepareStatement(requeteSQL);
    prepared_statement.setInt(1, id);

    int nblignes = 0;

    try {
        nblignes = prepared_statement.executeUpdate();
    } catch (SQLException ex) {
        {
            System.out.println(ex.getMessage());
        }
    }

    if(nblignes == 1) {
        reponse = true;
    } else {
        reponse = false;
    }

    prepared_statement.close();
    connexion.close();
    return reponse;
}
```

DEPLACER LA FENETRE

Pour pouvoir retirer les bordures Windows sur l'application et pouvoir déplacer l'application sur l'écran nous devons faire quelques manipulations car par défaut ce n'est pas possible.

Tout d'abord dans la classe « `Main` » j'ajoute une méthode pour obtenir le « `Stage` » de l'application pour pouvoir ensuite mettre à jour sa position via un « `EventHandler` » qui utilisera le mouvement de la souris faire cela. La fonction qui met à jour la position de l'application doit être appelé dans chaque controller. Par exemple dans le controller de la page de connexion « `ConnexionController` ». Ces fonctions doivent être appelé dans les fonctions « `initialize()` » de chaque controller.

```
mainPane.setOnMousePressed(new EventHandler<MouseEvent>()
{
    @Override
    public void handle(MouseEvent event)
    {
        xOffset = event.getSceneX();
        yOffset = event.getSceneY();
    }
});

mainPane.setOnMouseDragged(new EventHandler<MouseEvent>()
{
    @Override
    public void handle(MouseEvent event)
    {
        Main.obtenir_le_primaryStage().setX(event.getScreenX()- xOffset);
        Main.obtenir_le_primaryStage().setY(event.getScreenY()- yOffset);
    }
});
```

STATISTIQUES

Les statistiques montrés sont le pourcentage des formations par rapport à toutes les offres publiées et le pourcentage d'offres publié par entreprises.

Dans le fichier fxml « statistique.fxml » nous ajoutons le champ « PieChart » .

```
<fx:PieChart fx:id="statPieChart" data-bbox="62 142 975 155"/>
```

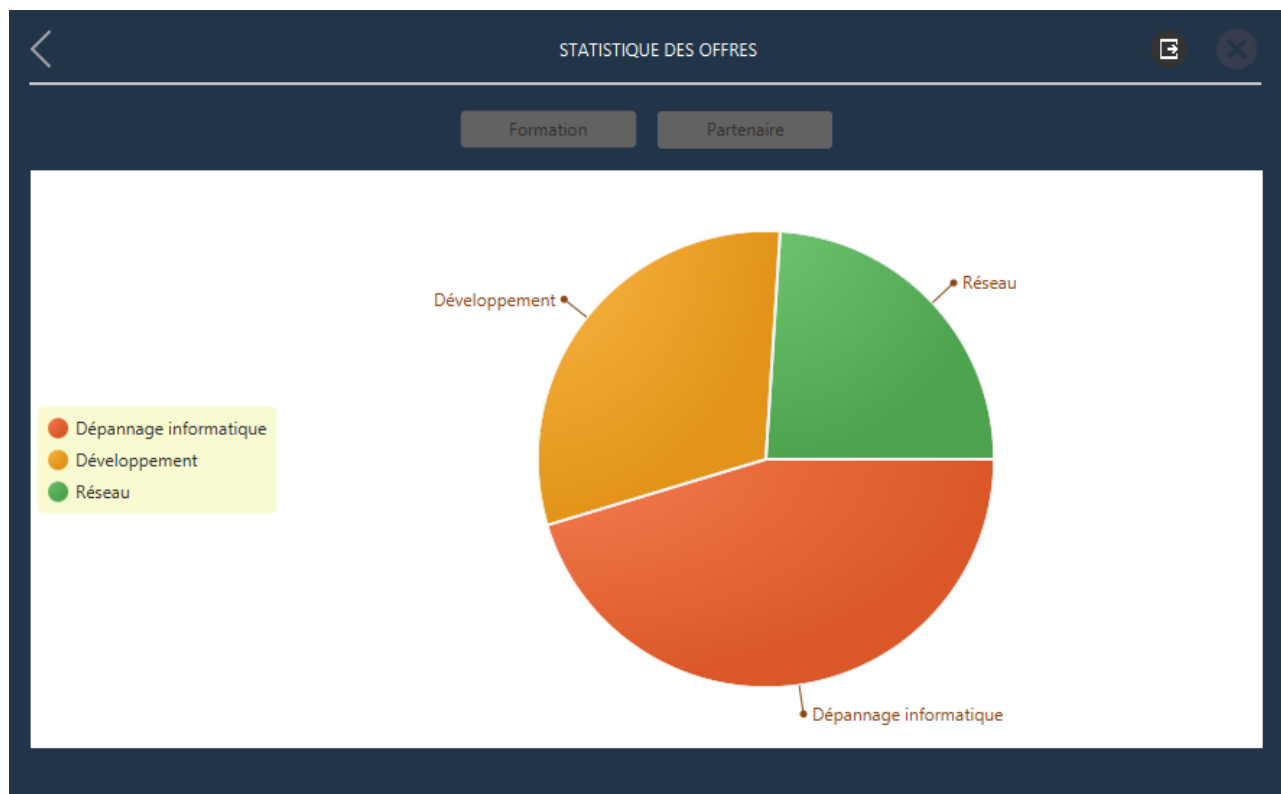
Dans le controller qui est « StatistiqueController » il faut déclarer les champs que nous utiliserons.

```
@FXML
private AnchorPane mainPane;
@FXML
private Button deconnexionButton;
@FXML
private Button fermetureButton;
@FXML
private Button retourButton;
@FXML
private Button partenaireButton;
@FXML
private Button formationButton;
@FXML
private PieChart StatPieChart;
```

Dans la fonction « initialize() » je récupère les informations contenu dans la base de données en faisant appel à une des fonctions qui se trouve dans la classe DAO de l'objet « User » qui est « UserDAO ». Puis je donne ces informations au « PieChart » avec la fonction « setData() ».

```
ObservableList<PieChart.Data> pieChartData = FXCollections.observableArrayList(
    OffreDAO.recuperer_les_statistiques_des_offres_par_formation());
StatPieChart.setData(pieChartData);
```

Ces manipulations donnent ceci :



ERREURS

Les erreurs sont affichées dans une nouvelle fenêtre, il est possible de spécifier le titre de l'alerte avec « setTitle() » et avec « setHeaderText() » il est possible d'expliquer plus en détail le problème. Pour ce projet ce code est utilisé dans les conditions principales.

```
Alert a1 = new Alert(Alert.AlertType.ERROR);  
a1.setTitle("Erreur: n°4");  
a1.setContentText("Cet utilisateur n'existe pas vérifiez votre mot de passe et votre email.");  
a1.setHeaderText(null);  
a1.showAndWait();
```

Exemple d'erreur :

