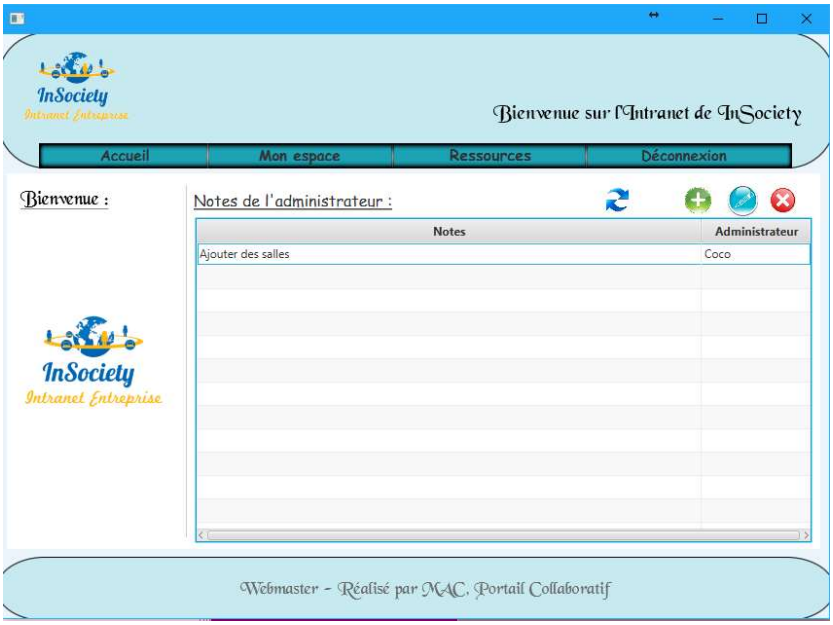
Session 2018

Projet InSociety

Documentation Technique



BTS SERVICES INFORMATIQUES AUX ORGANISATIONS
SOLUTIONS LOGICIELS ET APPLICATIONS METIERS

COCO MARIE-ANGE

Tables des matières

I. Description du projet

A. Présentation générale du projet

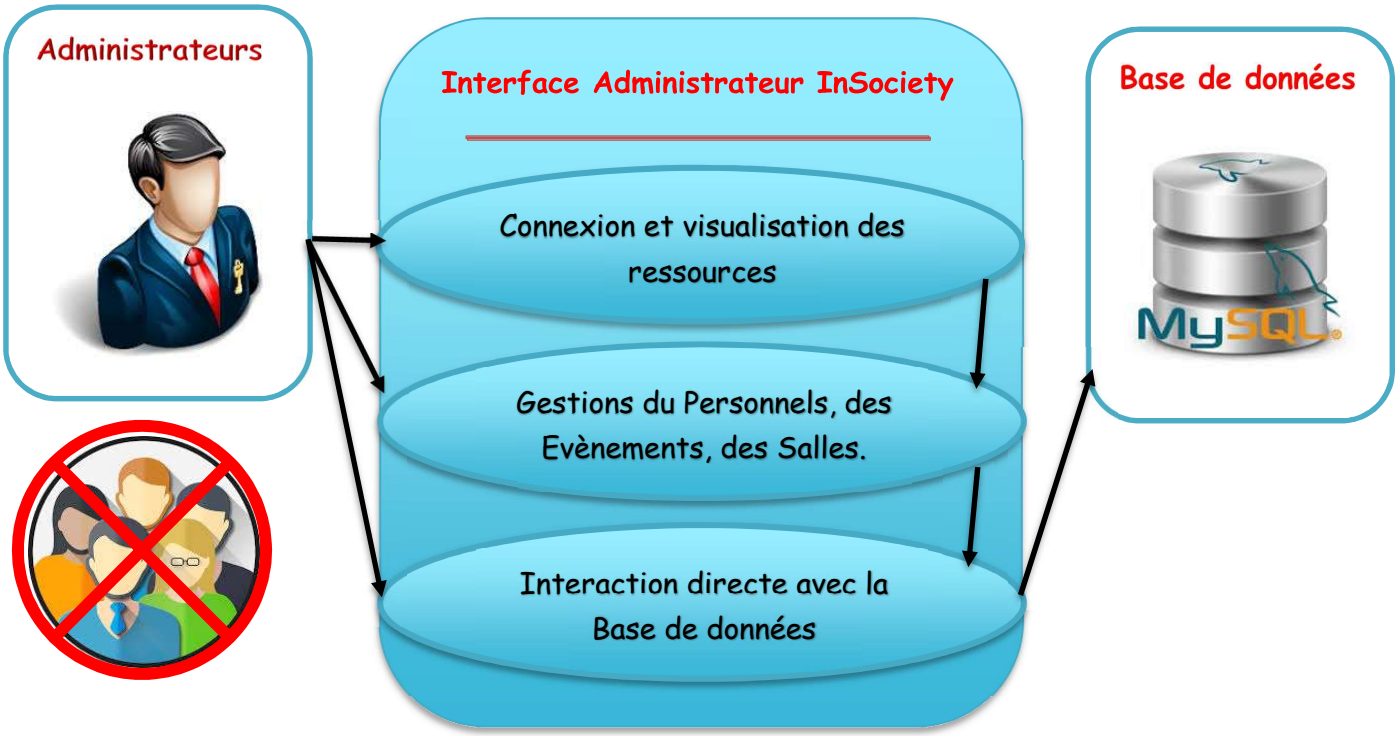
1) Contexte

La société Z-Media est une SSII qui répond aux demandes de ses principaux clients, qui sont essentiellement des établissements scolaires et universitaires.
Dans le cadre de ma formation en alternance au sein de l'entreprise, je vais contribuer à la mise en place d'un projet de développement d'une interface administrateur, c'est-à-dire une application ou l'administrateur du Site pourra se connecter (login et mot de passe) et grâce à celle-ci gérer les utilisateurs, les évènements et les salles de sa base de données. Seuls les administrateurs sont autorisés à se connecter pour plus de sécurité.

2) Définition du besoin

L'origine de cette demande est le Service Informatique de la Société Bollig & Kemper, industrie chimique spécialisée dans la peinture et le vernis pour le secteur automobile principalement, qui souhaite simplifier sa gestion des ressources concernant leurs utilisateurs, évènements et salles pour une utilisation plus efficace sur une seule et même interface et pour pouvoir mettre à jour leur intranet en conséquence.

3) Diagramme de cas d'utilisation

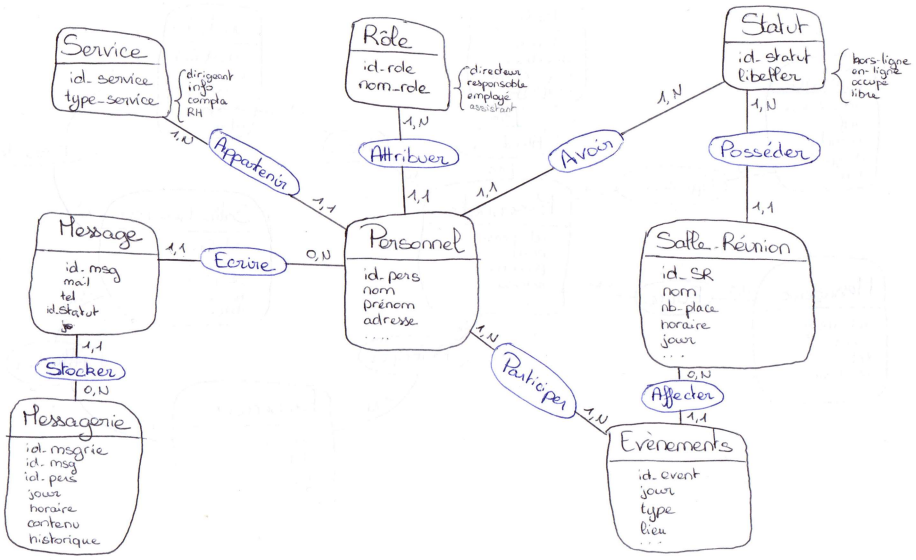


II - Bases de données

A partir du cahier des charges et après plusieurs réunions, j'ai pu réaliser le modèle conceptuel de données (voir figure ci-dessous). Et enfin à partir du modèle conceptuel j'ai pu dresser le modèle relationnel de données (voir figure ci-dessous), qui est le schéma qui se rapproche le plus d'une véritable base de données. Une fois terminé je l'ai fait valider par le chef de projet de l'entreprise.

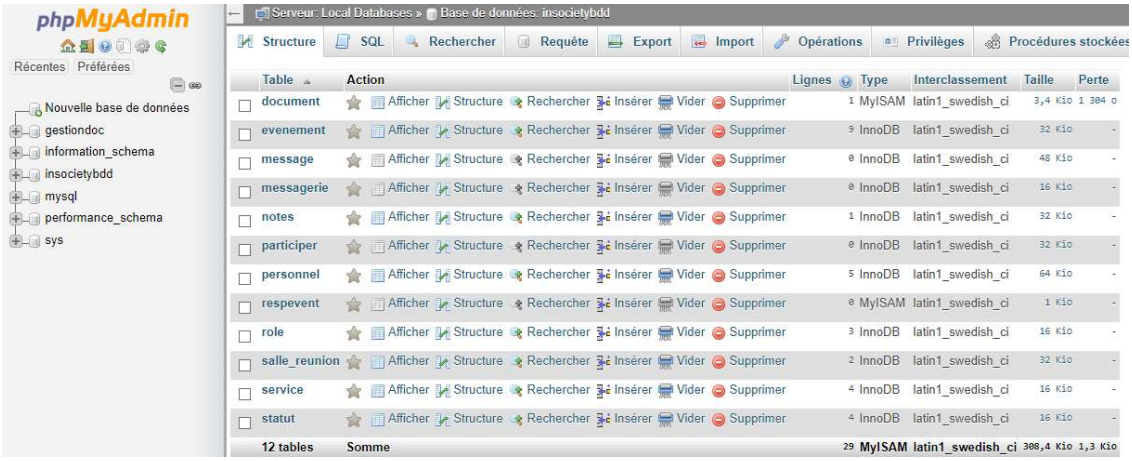
A. MCD (Modèle Conceptuel de Données)

Ce MCD est commun à mes 2 projets, c'est pourquoi vous pouvez y voir une table message et messagerie.

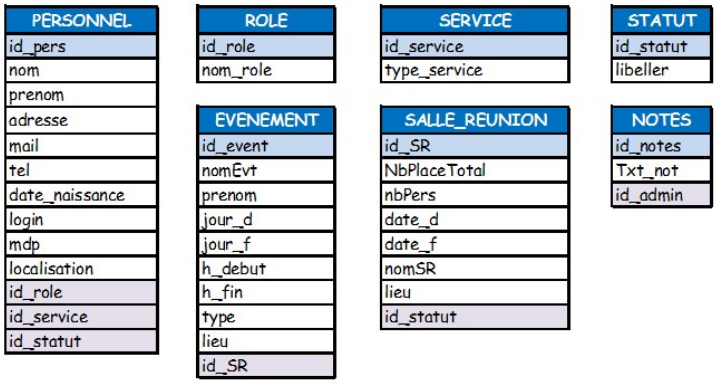


B. Mise en place de la Base de Données

Création de la base de données « insocietybdd » dans PhpMyAdmin, utilisation de **JMerise** pour la création du script MySQL et l'intégration des données dans la base créer dans PhpMyAdmin.



Aperçu des tables utilisés pour ce projet :



III - Outils de programmation

Le langage de programmation choisi pour ce projet est JAVA car l'application ou logiciel que nous allons créer est une application qui tourne sur une station fixe donc dite « lourde ».

A. Logiciels & Librairies

Plusieurs logiciels ont été nécessaires à la réalisation de ce projet :

- **WampServer 3.0.6** qui est une plate-forme de développement Web sous Windows pour des applications Web dynamiques à l'aide du serveur Apache2, du langage de scripts PHP et d'une base de données MySQL.
  Il possède également PHPMyAdmin pour gérer plus facilement vos bases de données.

- **JMerise** application Merise qui est une méthode d'analyse, de conception et de gestion de projet informatique. Outil choisi pour concevoir ma base de données.

- **Eclipse Java Oxygen 4.7** qui est un IDE qui consiste à développer ses projets comportant une interface graphique, d'un éditeur de code source avec auto-complétions du langage utilisés, un compilateur et un débogueur. Eclipse possède une interface évolutive pour s'adapter à chacun.

- **Scène Builder 9.0.1** intégrer directement dans Eclipse après installation et configuration.
  Cet outil permet de concevoir des interfaces utilisateur d'applications Java FX c'est-à-dire la mise en page visuelle des vues que le développeur souhaite créer sous forme de fichier FXML.

- **GitHub** pour la gestion des versions de mon projet, extension directement intégrée dans Eclipse appelé « EGit - Git Integration for Eclipse 4.6.0 » et envoyer sur mon profil GitHub dans le projet concerné.

De plus, les librairies suivantes ont été utilisées :

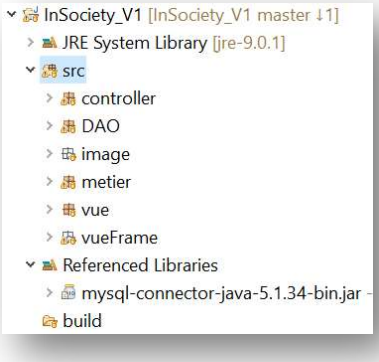
- JRE System Library [jre 9.0.1] : (Java Runtime Environnement) permet l'exécution du programme.
- mysql-connector-java-5.1.34-bin.jar : permet la connexion à la base de données sous MySQL.

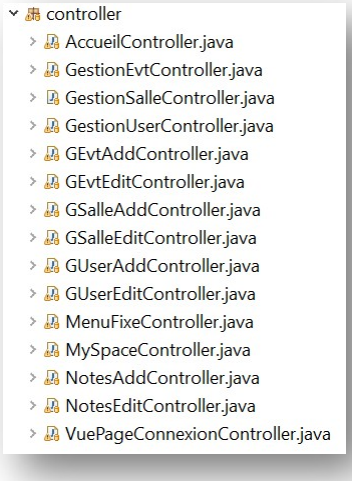
B. Code source & Fonctionnalités

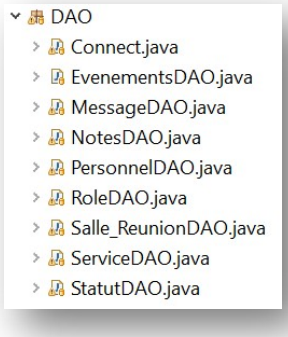
1) Code source

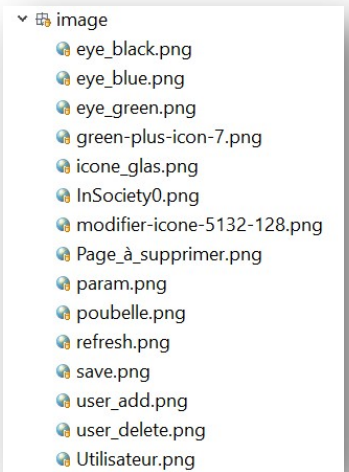
Après création du projet « InSociety » sous Eclipse, nous procédons à la création de plusieurs sous dossier dans le dossier source (src) pour mieux organiser notre code.
Dans un premier temps nous créons les classes « vue », « métiers » et « DAO » puis dans un deuxième temps nous créons les classes « vueFrame », « controller » et « image ».

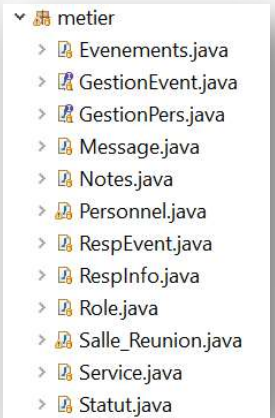
- Le Controller avec un Controller par vue FXML
- Le DAO (Data Access Object ou Couche d'accès aux données)
- Les images utilisées pour l'interface graphique
- Les classes métiers qui correspond aux tables de la base
- La vue qui correspond au « Main » pour le lancement du projet
- Les vueFrame ou on trouve tout notre visuel des pages en FXML avec utilisation de Scène Builder pour la modification.

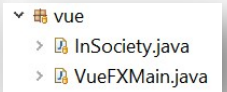


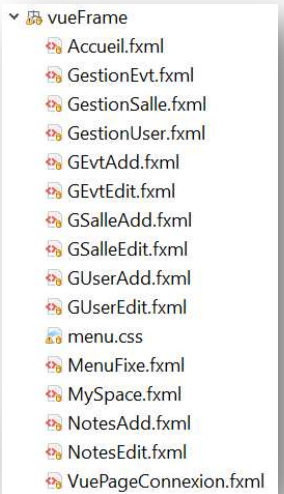












2) Fonctionnalités

- Le dossier « Controller » qui contient le code qui gère les actions des différentes vues de l'interface graphique (un Controller par vue).
- Le dossier « DAO » contient toutes les classes (une classe par table présente en base) qui ont besoin de se connecter à la base de données composer des requêtes SQL nécessaires à leurs utilisations.
- Le dossier « image » qui contient toutes les images utilisées dans les différentes vues pour l'interactivité avec les interfaces graphiques. Elles sont appelées dans les vues pour l'affichage et dans le contrôleur pour la gestion c'est-à-dire l'attribution d'une action par exemple.
- Le dossier « métier » qui correspond aux différentes tables de ma base, nous les créons pour initialiser les données, les crées et les récupérer par le biais des « getters et setters » notamment.
- Le dossier « vue », c'est le « main » qui permet d'exécuter le programme
- Le dossier « vueFrame » permet de créer les vues ou interfaces graphiques de chacune des pages, ce sont des fichiers FXML directement modifiable avec l'interface de Scène Builder.

3) Explication de la classe « personnel »

La classe « personnel » est utilisé lors de la connexion à l'application car les administrateurs sont aussi des membres du personnel.

Pour commencer nous créons la classe « Personnel.java » en tant que classe métier c'est-à-dire :

- Initialisation des variables contenu dans la classe personnel (id, nom, prénom, adresse, etc…).
  Etant donné que nous utilisons du Java FX, l'initialisation des variables est un peu différente, au lieu de mettre simplement le type (string) de la variable il faut mettre le type propre à Java FX, c'est-à-dire ici « SimpleStringProperty ».
  Pour utiliser ces variables, il faut importer la bibliothèque de Java Fx :
  `import javafx.beans.property.*;`

```
public class Personnel {
    // Initialisation des variables
    // CrÃ©ation de la classe Statut, Role, Service

    Statut statut;
    Role role;
    Service service;
    private IntegerProperty id;
    private StringProperty nom;
    private StringProperty prenom;
    private StringProperty adresse;
    private StringProperty mail;
    private StringProperty tel;
    private StringProperty login;
    private ObjectProperty<Date> date_naissance;
    private StringProperty localisation;
    private StringProperty mdp;
```

- Création des classes connectées à la classe personnel comme la classe « Statut », « Rôle » et « Service » avec le même principe que celle-ci
- Récupération des données de ses autres classes. Ex : « Statut statut ; »
- Création des getters et setters propre à Java FX et basique.

```
// CrÃ©ation des getters et setters Property
public IntegerProperty getIdPro() {
    return id;
}

public void setIdPro(IntegerProperty id) {
    this.id = id;
}
```

```
// CrÃ©ation des getters et setters
public int getId() {
    return id.get();
}

public void setId(int id) {
    this.id.set(id);
}
```

Ensuite dans le dossier DAO, nous créons la classe « PersonnelDAO.java », c'est elle qui communiquera avec la base de données. Dedans nous y mettrons toutes les requêtes que nous avons besoins pour faire fonctionner notre application (ajout, modification, suppression, vérification, etc…)

```
public class PersonnelDAO {

    public static boolean adminVerif(String login, String mdp) throws SQLException {

        Connection cnx = Connect.getInstance().getConnection();
        String req = "SELECT nom_role, type_service FROM Personnel, Service, Role "
                + "WHERE login=? AND mdp=? AND Personnel.id_role = Role.id_role AND Service.id_service = Personnel.id_service";
        PreparedStatement pst = cnx.prepareStatement(req);
        pst.setString(1, login);
        pst.setString(2, mdp);
        ResultSet userInformation = pst.executeQuery();
        while (userInformation.next()) {

            String f = userInformation.getString("nom_role");
            String g = userInformation.getString("type_service");

            if (f.startsWith("Resp") && g.startsWith("Info")) {
                return true;
            }
        }

        return false;
    }
```

Ici la requête permet de vérifier qu'il s'agit d'un administrateur en vérifiant, s'il fait bien partie du service Informatique et avec le Rôle de Responsable.

```
public static void insertPers(Personnel pers) throws SQLException, ClassNotFoundException {
    // Je me connecte
    Connection co = Connect.getInstance().getConnection();

    // CrÃ©ation de la requÃªte inserer new pers
    String requeteSQL = "INSERT INTO `personnel`(`nom`, `prenom`, `adresse`, `mail`, `Tel`,"
            + "`id_role`, `id_service`, `id_statut`, `login`, `date_naissance`, `localisation`, `mdp`)"
            + " VALUES (?,?,?,?,?,?,?,?,?,?,?,?)";

    // prÃ©parer la requÃªte
    PreparedStatement pst = co.prepareStatement(requeteSQL);

    // renvoyer et verifier les donnÃ©es de la requÃªte
    pst.setString(1, pers.getNom());
    pst.setString(2, pers.getPrenom());
    pst.setString(3, pers.getAdresse());
    pst.setString(4, pers.getMail());
    pst.setString(5, pers.getTel());

    // RÃ©cupÃ¨re et vÃ©rifier la clÃ© Ã©trangÃ¨re de la table Role puis pareil pour
    // Service, Statut

    pst.setInt(6, RoleDAO.GetIdRole(pers.getRole().getNom_role()));
    pst.setInt(7, ServiceDAO.GetIdServ(pers.getService().getType_service()));
    pst.setInt(8, StatutDAO.GetIdStat(pers.getStatut().getLibeller()));

    // renvoyer et vÃ©rifier les donnÃ©es de la requÃªte (suite)
    pst.setString(9, pers.getLogin());
    pst.setDate(10, pers.getDate_naissance());
    pst.setString(11, pers.getLocalisation());
    pst.setString(12, pers.getMdp());

    int nbligne = pst.executeUpdate();

}
```

Celui-ci est une requête pour ajouter du personnel dans la base de données.

Dans le dossier « vueFrame » avec Scène Builder, nous créons la vue correspondante pour afficher notre personnel par exemple :
Pour faire cela, faire un clic droit sur le dossier « vueFrame » puis New… > Other… et Sélectionner « New FXML Document », lui donner un nom ici « GestionUser.fxml » et cliquer sur « Finish ». Une fois le fichier créer, faire un clic droit sur celui-ci et sélectionner « Open with Scène Builder ». L'interface de scène builder s'ouvre et vous pouvez créer votre vue (le contenu est à gauche et la configuration du contenu à droite).

Le code se fait alors automatiquement quand vous sauvegardez votre vue :

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.image.*?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.*?>
<Pane fx:id="GestionUserView" prefHeight="386.0" prefWidth="836.0" style="-fx-background-color: white; -fx-border-radius: 100px;
   <children>
      <Text layoutX="13.0" layoutY="26.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Gestion Personnel : " underline="true">
         <font>
            <Font name="BlackChancery" size="19.0" />
         </font>
      </Text>
      <Label fx:id="addUserView" layoutX="68.0" layoutY="118.0" onMouseClicked="#addUser" text="Ajouter Utilisateur">
         <font>
            <Font name="Comic Sans MS" size="11.0" />
         </font>
      </Label>
      <Label fx:id="editUserView" layoutX="67.0" layoutY="176.0" onMouseClicked="#editUser" text="Modifier Utilisateur">
         <font>
            <Font name="Comic Sans MS" size="11.0" />
         </font>
      </Label>
      <Label fx:id="suppUserView" layoutX="62.0" layoutY="247.0" onMouseClicked="#suppUser" text="Supprimer Utilisateur">
         <font>
            <Font name="Comic Sans MS" size="11.0" />
```

Ne pas oublier d'ajouter cette ligne avec le nom du Controller en fin du fichier FXML : `fx:controller="controller.GestionUserController"`

Nous procédons ensuite à la gestion de cette vue grâce au Controller « GestionUserController.java », c'est pourquoi il faut ajouter `fx:controller="controller.GestionUserController"` dans la vue pour faire la connexion entre les deux.
Dans un premier temps, pour pouvoir donner des actions aux différents objets (images, bouton, etc…), il faut initialiser les variables correspondantes à leur « id » dans la vue.

```
<ImageView fx:id="allUser" fitHeight="37.0" fitWidth="32.0" layoutX="18.0" layoutY="314.0"
onMouseClicked="#afficher" pickOnBounds="true" preserveRatio="true">
   <image>
      <Image url="@../image/eye_green.png" />
   </image>
</ImageView>
```

```
public class GestionUserController {
    @FXML
    private Label alluser;
    @FXML
    private ImageView allUser;
```

Appel de l'image dans le Controller :

Ensuite, nous pouvons donner une action a cette image :

```
// Label + image Afficher tout
@FXML
public void afficher(MouseEvent actionEvent) throws SQLException, ClassNotFoundException {
    try {
        System.out.println("TEST");
        ObservableList<Personnel> empData = PersonnelDAO.GetListePersonnel();
        tabUser.setItems(empData);
    } catch (ClassNotFoundException | SQLException e) {
        // TODO Bloc catch gï¿½nï¿½rï¿½ automatiquement
        e.printStackTrace();
    }
}
```

Pour les images sont stockées dans le dossier image et récupérer dans « GestionUser.fxml » de la façon suivante : `<Image url="@../image/eye_green.png" />`

Pour finir le dossier « vue », se trouve le « main » qui permet de lancer les différentes pages de l'application, fichier appelé ici « VueFXMain.java » :

```
public class VueFXMain extends Application {
    private static Stage primaryStage;

    @SuppressWarnings("static-access")
    @Override
    public void start(Stage primaryStage) {
        try {
            this.setPrimaryStage(primaryStage);
            new BorderPane();

            FXMLLoader acc = new FXMLLoader();
            acc.setLocation(VueFXMain.class.getClassLoader().getResource("vueFrame/VuePageConnexion.fxml"));
            Pane rootL = (Pane) acc.load();
            Scene scene = new Scene(rootL, 850, 600);
            // scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
            this.getPrimaryStage().setScene(scene);
            this.getPrimaryStage().show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Ici voici le code pour lancer la page de connexion de l'application.

4) Difficultés rencontrées

Pour commencer ce projet, première difficulté :

- réaliser un MCD correspondant au mieux à la demande, plusieurs schémas ont été fait avant d'arriver à celui présenté dans ce document.
- Trouver un langage de programmation adapté à la demande
- Choix des outils de programmation, Netbeans pour finir sur Eclipse car plus complet et Java FX déjà intégrer grâce à un plugin.

Pour coder le projet, les difficultés ont été :

- Les erreurs de code parfois difficiles à trouver et/ou décrypter.
- Compréhension des différentes erreurs.
- La version de java 8.161 qui rentrait en conflit avec le JavaFX donc installation du JDK (Java Développement Kit) et JRE (Java Runtime Environnement) ou à proprement parler Java 9.0.1
- La complexité de certaines requête SQL dans le dossier « DAO ».
- L'intégration du langage FXML
- L'assimilation d'utilisation de Scène Builder
- Affichage des formats de date et de temps ainsi que ceux de choix multipes.
- L'intégration de GitHub dans Eclipse via un plugin

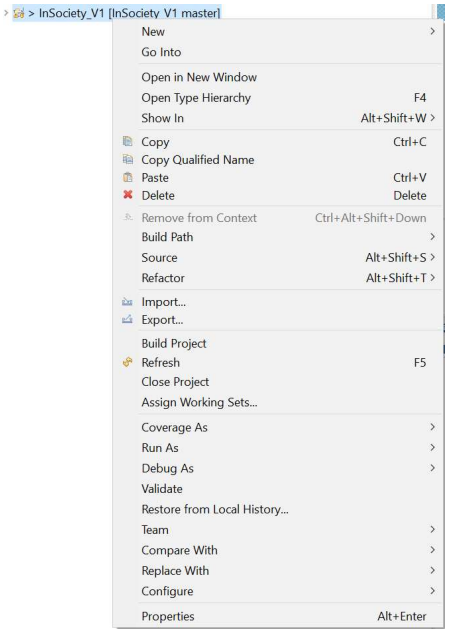
Pour finaliser celui-ci :

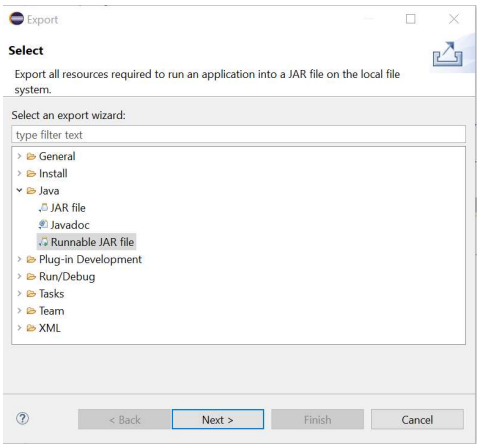
- La mise en place d'un exécutable en « .jar »

IV - Installation

A. Exécution

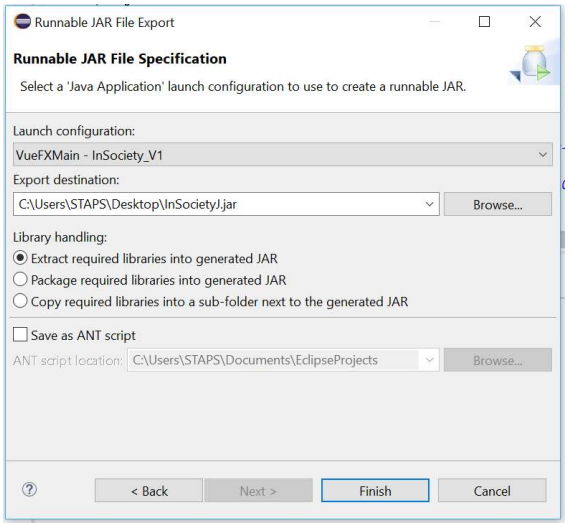
Pour créer un fichier exécutable, il faut faire un clic droit sur le nom de son projet dans Eclipse. Sélectionner « Export… », une fenêtre s'ouvre :





Cliquer sur le dossier Java et sélectionner « Runnable JAR file », ensuite faire « Next > »

Une autre fenêtre apparaît pour choisir les différentes données d'export.



Choisir dans « Export destination : » l'emplacement d'enregistrement du fichier « .jar » en cliquant sur « Browse »
Choisir ensuite dans « Library handling : »

- La première option si votre projet contient des librairies comme celui-ci
- La deuxième option si vous voulez créer un dossier librairies avec le « .jar »
- La troisième option si vous voulez créer un sous dossier avec ses librairies.

Pour finir cliquez sur « Finish », il vous dit que vous allez créer un export de votre projet, cliquer sur « ok ».
Vous trouverez maintenant un fichier « .jar » à l'emplacement ou vous l'avez enregistré comme ceci :

B. Lancement

Pour pouvoir lancer le programme, double cliquer sur le fichier « .jar » 

et le programme se lance.

Pour l'utilisation de l'application voire « Documentation d'utilisation administrateur ».