Projet / Missions

AP2.3 : Projet Messagerie

Présentation de l'équipe et de la mission	
Présentation	Nous sommes deux élèves de BTS SIO SLAM, Christian et Mélanie, et, dans le cadre de notre deuxième AP, nous avons eu pour consigne de développer la gestion des messages du projet Messagerie.
Tâches	Réalisation des tâches suivantes : 1er partie : Voir les messages / 2ème partie : Envoyer un message / 3ème partie : Supprimer un message / 4ème partie : Répondre à un message
Outils	Afin de réaliser ces tâches, nous avons utilisé le logiciel Netbeans dans lequel nous avons utilisé le langage Java.

Rappel des objectifs

Après avoir étudier le code des classes métiers déjà développées vous devez coder les deux classes suivantes : GestionMessage.java : la classe gère les messages JdbcMessage.java : la classe devra être développée selon les mêmes principes que JdbcUsers.java

Voir les messages

Comme indiqué dans la consigne, nous devons faire afficher tous les messages en fonction de l'utilisateur connecté. Pour cela, nous avons créé deux fonctions.

La première, comme nous pouvons le voir ci-dessous, que avons nommé **GetAllMessage**, va récupérer tous les messages en fonction de l'**origineUsers** et du **destinataireUsers**.

```
public ResultSet getAllMessage() {

        String query = "SELECT * FROM message WHERE `message`.`origineUsers` = 4 or
`message`.`destinataireUsers` = 4 ";

        ResultSet rs = null;

        try {



            Statement st = conn.createStatement();



            rs = st.executeQuery(query);

        } catch (SQLException e) {

            System.err.print(e);

        }
```

```
        return rs;

    }
```

La deuxième comme nous pouvons le voir ci-dessous, que avons nommé **getAllUtilisateurs**, va récupérer tout les utilisateurs.

```
public ResultSet getAllUtilisateurs() {

        String query = "SELECT * FROM users";

        ResultSet rs = null;

        try {



            Statement st = conn.createStatement();



            rs = st.executeQuery(query);

        } catch (SQLException e) {

            System.err.print(e);

        }



        return rs;

    }
```

Il ne nous reste plus qu'à ajouter dans le case 1 de notre class **GestionMessage** le code qui nous permettra d'appeler et de faire fonctionner les fonctions ci-dessus.

```
case 1:

                    ResultSet rs = jdbc.getAllMessage();

                    ResultSet rso = jdbc.getAllUtilisateurs();

                    String nom[] = new String[20];

                    int Id[] = new int[20];

                    int i = 0;

                    while (rso.next()) {
```

```
            String NomUsers = rso.getString("identifiant");
            int IdUsers = rso.getInt("id");
            nom[i] = NomUsers;
            Id[i] = IdUsers;
            i++;
        }

        while (rs.next()) {
            String oUsers = "";
            String dUsers = "";
            int id = rs.getInt("id");
            int origineUsers = rs.getInt("origineUsers");
            int destinataireUsers = rs.getInt("destinataireUsers");
            String objet = rs.getString("objet");
            String message = rs.getString("message");
            Date dateEnvoi = rs.getDate("dateEnvoi");
            String etat = rs.getString("etat");

            for (i = 0; i < Id.length; i++) {
                if (origineUsers == Id[i]) {
                    oUsers = nom[i];

                }

            }

            for (i = 0; i < Id.length; i++) {
```

```
                    if (destinataireUsers == Id[i]) {
                        dUsers = nom[i];
                    }
                }
                System.out.format("%s\n%s\n%s\n%s\n%s\n%s\n\n",
                        "Objet: " + objet,
                        "Message de " + oUsers,
                        "À " + dUsers,
                        "Le " + dateEnvoi,
                        "Texte:",
                        message);
            }
            break;
```

Nous pouvons à présent tester l'affichage des messages.

Comme le montre la capture ci-dessous, nous nous sommes connectés avec l'identifiant agnes.bourgeois, et nous voyons qu'elle a reçu un message. Elle peut donc consulter les informations concernant ce message (objet, destinataire, date, message)

L'affichage des messages a fonctionné 🙂

Envoyer un message

Comme indiqué dans la consigne, nous devons envoyer un message à un autre utilisateur. Pour cela, nous avons créé deux fonctions.

La première, comme nous pouvons le voir ci-dessous, que avons nommé **GetUtilisateur2**, va récupérer tout les utilisateurs en fonction de l'identifiant.

```
public ResultSet getUtilisateurById(String identifiant) {

        String query = "SELECT * FROM users where identifiant='" + identifiant + "'";

        ResultSet rs = null;

        try {



            Statement st = conn.createStatement();



            rs = st.executeQuery(query);

        } catch (SQLException e) {

            System.err.print(e);

        }



        return rs;

    }
```

La deuxième, comme nous pouvons le voir ci-dessous, que nous avons nommé **insertMessage** va permettre l'insertion d'un message.

```
public void insertMessage(String[] données) {

        String sql = "INSERT into message "

                + "(origineUsers,"

                + "destinataireUsers,"

                + "objet,"

                + "message,"

                + "dateEnvoi,"

                + "etat)"

                + " VALUES(?,?,?,?,?,?)";

        try {

            PreparedStatement prepare;
```

```
            prepare = conn.prepareStatement(sql);
            prepare.setString(1, données[0]);
            prepare.setString(2, données[1]);
            prepare.setString(3, données[2]);
            prepare.setString(4, données[3]);
            prepare.setString(5, données[4]);
            prepare.setString(6, données[5]);

            int r = prepare.executeUpdate();
            prepare.close();
        } catch (SQLException e) {
            System.err.println(e);
        }
    }
```

Il ne nous reste plus qu'à ajouter dans le case 2 de notre class **GestionMessage** le code qui nous permettra d'appeler et de faire fonctionner les fonctions ci-dessus.

```
case 2:
                    reponse = GForm.show("AJOUTER UN MESSAGE", champ, null);
                    int ori = 0;
                    rs = jdbc.getUtilisateurById(identifiant);
                    while (rs.next()) {
                        ori = rs.getInt("id");
                    }
                    String s = String.valueOf(ori);
                    rs = jdbc.getUtilisateur2(reponse[0]);
                    int des = 0;
                    while (rs.next()) {
```

```
                    des = rs.getInt("id");

                }

                String p = String.valueOf(des);

                if (reponse != null) {

                    String envoyer[] = {s, p, reponse[1], reponse[2], reponse[3], "non
lu"};

                    jdbc.insertMessage(envoyer);

                }

                break;
```

Nous pouvons à présent tester l'envoi des messages.

Toujours connecté avec l'identifiant agnes.bourgeois, nous allons envoyer un message à claude.pasqualini.

Pour vérifier si l'envoi du message a bien fonctionné, nous pouvons nous rendre dans la base de donnée du projet via phpmyadmin.

id	origineUsers	destinataireUsers	objet	message	dateEnvoi	etat
15	6	0	message	bonjour	2021-09-22	non lu

L'envoi d'un message a fonctionné 🙂

Supprimer un message

Comme indiqué dans la consigne, nous devons avoir également la possibilité de supprimer un message. Pour cela, nous n'avons besoin que de créer une seule fonction. Mais, nous allons nous resservir des fonctions précédemment créé, à savoir **getAllMessage**, **getMessage** et **getAllUtilisateurs**.

La fonction, comme nous pouvons le voir ci-dessous, que avons nommé **deleteMessage**, va permettre la suppression d'un message.

```
public void deleteMessage(int id) {

        String sql = "DELETE FROM message"

                + " where id=?";
```

```
        try {
            conn.setAutoCommit(false);

            PreparedStatement prepare = conn.prepareStatement(sql);

            prepare.setInt(1, id);

            int r = prepare.executeUpdate();
            conn.commit();

        } catch (SQLException e) {
            System.err.print(e);
        }
    }
```

Il ne nous reste plus qu'à ajouter dans le case 3 de notre class **GestionMessage** le code qui nous permettra d'appeler et de faire fonctionner la fonction ci-dessus.

```
case 3:
                    rs = jdbc.getAllMessage();
                    rso = jdbc.getAllUtilisateurs();
                    String nom3[] = new String[20];
                    int Id3[] = new int[20];
                    i = 0;
                    while (rso.next()) {
                        String NomUsers = rso.getString("identifiant");
                        int IdUsers = rso.getInt("id");
                        nom3[i] = NomUsers;
```

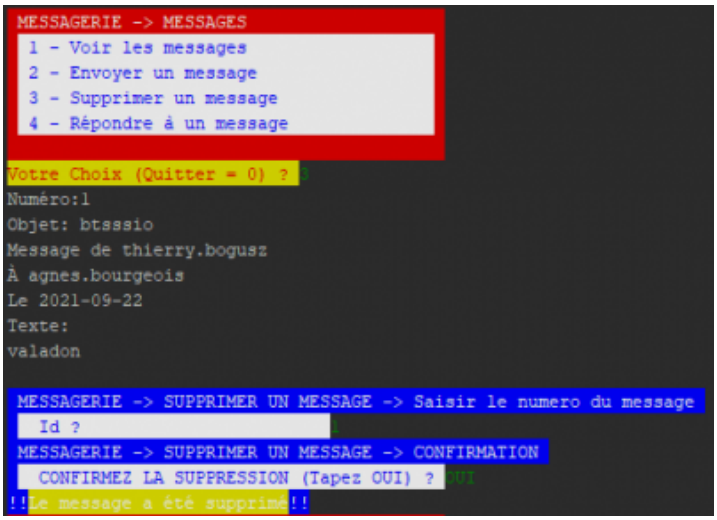
```
                Id3[i] = IdUsers;
                i++;
            }
            int numero = 0;
            while (rs.next()) {
                   String oUsers = "";
                String dUsers = "";
                numero++;
                int id = rs.getInt("id");
                int origineUsers = rs.getInt("origineUsers");
                int destinataireUsers = rs.getInt("destinataireUsers");
                String objet = rs.getString("objet");
                String message = rs.getString("message");
                Date dateEnvoi = rs.getDate("dateEnvoi");
                String etat = rs.getString("etat");
                // print the results

                for (i = 0; i < Id3.length; i++) {
                    if (origineUsers == Id3[i]) {
                        oUsers = nom3[i];
                    }
                }
                for (i = 0; i < Id3.length; i++) {
                    if (destinataireUsers == Id3[i]) {
                        dUsers = nom3[i];
                    }
                }
```

```
                    System.out.format("%s\n%s\n%s\n%s\n%s\n%s\n%s\n\n",
                            "Numéro:" + id,
                            "Objet: " + objet,
                            "Message de " + oUsers,
                            "À " + dUsers,
                            "Le " + dateEnvoi,
                            "Texte:",
                            message);
                }
                String[] champSup = {"Id"};
                String[] usersup = GForm.show("SUPPRIMER UN MESSAGE -> Saisir le numero du message", champSup, null);

                ResultSet rss = jdbc.getMessage(usersup[0]);

                if (rss.next() == false) {
                    GForm.message("Numero inconnu...");
                    break;
                }
                String confirm[] = {"CONFIRMEZ LA SUPPRESSION (Tapez OUI)"};
                reponse = GForm.show("SUPPRIMER UN MESSAGE -> CONFIRMATION", confirm, null);
                if (reponse != null && reponse[0].equals("OUI")) {
                    jdbc.deleteMessage(rss.getInt("id"));
                    GForm.message("Le message a été supprimé");
                } else {
                    GForm.message("AUCUNE SUPPRESSION...");
                }
```

```
                    break;
```

Nous pouvons à présent tester la suppression d'un message.

Toujours connecté avec l'identifiant agnes.bourgeois, nous allons supprimer le message qu'elle avait reçu lorsque nous avons testé l'affichage des messages.



Pour vérifier si la suppression du message a bien fonctionné, nous pouvons nous rendre dans la base de données du projet via phpmyadmin.

Comme le montre la capture ci-dessus, le message a bien été supprimé car il ne se trouve plus dans la base de données.

La suppression a fonctionné ☺

Répondre à un message

Comme indiqué dans la consigne, nous devons répondre à un message envoyé par un autre utilisateur. Pour cela, nous n'avons besoin que de créer une seule fonction. Mais, nous allons nous resservir de la fonction **getMessage**.

La fonction, comme nous pouvons le voir ci-dessous, que avons nommé **ReponseMessage**, va permettre de répondre à un message que l'on nous a envoyé.

```
public void ReponseMessage(String[] données) {

        String sql = "INSERT into message "

                + "(origineUsers,"

                + "destinataireUsers,"

                + "objet,"
```

```
                + "message,"
                + "dateEnvoi,"
                + "etat)"
                + " VALUES(?,?,?,?,?,?)";

        try {
            PreparedStatement prepare;
            prepare = conn.prepareStatement(sql);
            prepare.setString(1, données[0]);
            prepare.setString(2, données[1]);
            prepare.setString(3, données[2]);
            prepare.setString(4, données[3]);
            prepare.setString(5, données[4]);
            prepare.setString(6, données[5]);

            int r = prepare.executeUpdate();
            prepare.close();
        } catch (SQLException e) {
            System.err.println(e);
        }

    }
```

Il ne nous reste plus qu'à ajouter dans le case 4 de notre class **GestionMessage** le code qui nous permettra d'appeler et de faire fonctionner la fonction ci-dessus.

```
case 4:
                    rs = jdbc.getAllMessage();
                    rso = jdbc.getAllUtilisateurs();
```

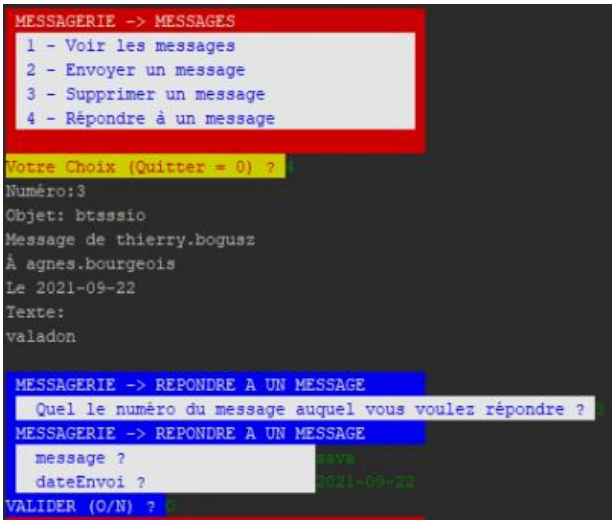
```
                String nom2[] = new String[20];

                int Id2[] = new int[20];

                i = 0;

                while (rso.next()) {

                    String NomUsers = rso.getString("identifiant");

                    int IdUsers = rso.getInt("id");

                    nom2[i] = NomUsers;

                    Id2[i] = IdUsers;

                    i++;

                }

                numero = 0;

                while (rs.next()) {

                    String oUsers = "";

                    String dUsers = "";

                    numero++;

                    int id = rs.getInt("id");

                    int origineUsers = rs.getInt("origineUsers");

                    int destinataireUsers = rs.getInt("destinataireUsers");

                    String objet = rs.getString("objet");

                    String message = rs.getString("message");

                    Date dateEnvoi = rs.getDate("dateEnvoi");

                    String etat = rs.getString("etat");

                    // print the results


                    for (i = 0; i < Id2.length; i++) {

                        if (origineUsers == Id2[i]) {

                            oUsers = nom2[i];
```

```
            }

        }

        for (i = 0; i < Id2.length; i++) {
            if (destinataireUsers == Id2[i]) {
                dUsers = nom2[i];

            }

        }
        System.out.format("%s\n%s\n%s\n%s\n%s\n%s\n%s\n\n",
                "Numéro:" + id,
                "Objet: " + objet,
                "Message de " + oUsers,
                "À " + dUsers,
                "Le " + dateEnvoi,
                "Texte:",
                message);

    }
    reponse = GForm.show("REPONDRE A UN MESSAGE", champReponse, null);

    ResultSet rsm = jdbc.getMessage(reponse[0]);

    if (rsm.next() == false) {
```

```
                    GForm.message("Message inconnu...");

                    break;

                }

                String origine = rsm.getString("origineUsers");

                String destinataire = rsm.getString("destinataireUsers");

                String objet = rsm.getString("objet");

                reponse = GForm.show("REPONDRE A UN MESSAGE", champMessage, null);


                String ReMessage[] = {destinataire, origine, objet, reponse[0],
reponse[1], "non lu"};

                if (reponse != null) {

                    jdbc.ReponseMessage(ReMessage);

                }

                break;
```

Nous pouvons à présent tester la réponse à un message.

Toujours connecté avec l'identifiant agnes.bourgeois, nous allons répondre à un message qu'elle a reçu.



Comme le montre la capture ci-dessus, la réponse à un message a fonctionné 🙂