



INSTITUT NATIONAL DES POSTES ET TÉLÉCOMMUNICATIONS

RAPPORT DE PROJET

Dynamic Web project : Psychologue en ligne

Realisé par :
Driouch Meryam
Azzim Taha

Sous l'encadrement de :
Dr Mahmoud Rlhamlaoui

Année universitaire :2020-2021

Contents

1	Analyse et spécification des besoins	3
1.1	Introduction	3
1.2	Analyse des besoins	3
1.2.1	Les besoins fonctionnels	3
2	Diagramme des cas d'utilisation	5
2.1	définition	5
2.2	Diagramme des cas d'utilisation	5
2.3	Description textuelle des principaux cas d'utilisation	6
2.4	Conclusion	7
3	Modélisation conceptuelle	7
3.1	Introduction	7
3.2	Diagramme de classes	8
3.2.1	Présentation des classes	8
3.2.2	Présentations des attributs et des méthodes	9
3.2.3	Diagramme de classe	11
4	Page Login	11
4.1	Page d'identification	11
4.2	Connexion avec une base de données MySQL : vérification des identifiants	13
4.2.1	Classe Session	13
4.2.2	Classe DB	14
4.2.3	Création de la base de données "userdb"	16
4.3	Login servlet	17
4.4	Teste de login	19
4.5	Configuration des différents acteurs : Psychologue, RH et Util- isateur	19
4.6	Partie CSS	23
5	Conception de la base de données	25
6	Interface "Utilisateur"	27
6.1	Classe Question	27
6.2	Chargement des questions depuis la base de données	28

6.3	Affichage des questions	30
6.4	Partie CSS	31
6.5	Récupération des réponses par la base de données	33
7	Interface "RH"	36
7.1	Classe Formulaire	36
7.2	Chargement des formulaires depuis la base de donnée	37
7.3	Affichage de formulaire	38
7.4	Partie CSS	40
7.5	Récupération des états par la base de données	43

1 Analyse et spécification des besoins

1.1 Introduction

Pour assurer une bonne compréhension des différents fonctionnalités de notre projet, nous allons consacrer cette section pour identifier les acteurs de notre plateforme, cataloguer les besoins fonctionnels et les besoins non fonctionnels et terminer avec une présentation du diagramme des cas d'utilisation générale ainsi que le raffinement de chaque cas d'utilisation de notre application.

1.2 Analyse des besoins

Notre but de cette partie est de définir avec détails l'ensemble des fonctionnalités offertes par l'application. Les besoins dégagés ont été répartis en deux catégories fonctionnnels et non fonctionnels.

1.2.1 Les besoins fonctionnels

Les besoins fonctionnels se sont les fonctionnalités du système. Ce sont d'autre part les besoins spécifiant un comportement d'entrée ou bien sortie du système, ces derniers sont classés par acteurs.

Présentation des acteurs

Un acteur représente une personne ou un matériel ou un logiciel qui interagit d'une manière directer avec le système. Un acteur a le droit de consulter ou modifier directement l'état du système en émettant ou recevant des messages susceptible d'être porteur des données.

Les acteurs qui interagient dans notre système sont:

- **Utilisateur:** Un utilisateur est déjà inscrit dans l'application,il peut répondre aux questions proposer par le psychologue et approuver par le RH.
- **Psychologue :** Ajoute un questionnaire et envoie des recommandations à l'utilisateur.
- **RH:** il approuve ou affecte le questionnaire à passer.

Les besoins fonctionnels par Acteur

Le tableau ci dessus représente les besoins fonctionnels de notre système:

Acteur	Fonctionnalités
Utilisateur	-S'authentifier -Répondre au questionnaire
RH	-S'authentifier -Approuver le questionnaire
Psychologue	-S'authentifier -Ajouter des questions manuellement ou en chargeant un fichier CSV -Envoyer des recommandations à l'utilisateur

Les besoins non fonctionnels

Notre système doit répondre à des besoins qui ne sont pas indispensables pour son fonctionnement mais qui sont d'autre part importants pour sa qualité de ses services. Les besoins non fonctionnel sont importants à leurs rôles car ils interagissent d'une manière indirecte sur le résultat et sur le rendement de l'utilisateur.

On peut donc résumer les principaux besoins non fonctionnels de notre système dans :

- **La rapidité** : il est nécessaire que la durée d'exécution des traitements s'approche le plus possible du temps réel.
- **L'ergonomie de l'application** : L'application doit présenter des interfaces simples et ergonomiques pour que l'utilisateur puisse surfer facilement.
- **Sécurité** : Une authentification est obligatoire lors du démarrage pour accéder et effectuer les opérations désirées.
- **Le code** : Le code de l'application doit être clair pour permettre des futures améliorations.

2 Diagramme des cas d'utilisation

2.1 définition

Le diagramme des cas d'utilisation permet de formaliser les besoins et de modéliser les services offerts par le système. C'est donc une vue du système dans son environnement extérieur. Il modélise à la fois des fonctionnalités et des interactions pour les acteurs.

2.2 Diagramme des cas d'utilisation

Le diagramme suivant est réalisé à l'aide de l'outil **Enterprise Architect** et qui décrit l'ensemble des cas d'utilisation :

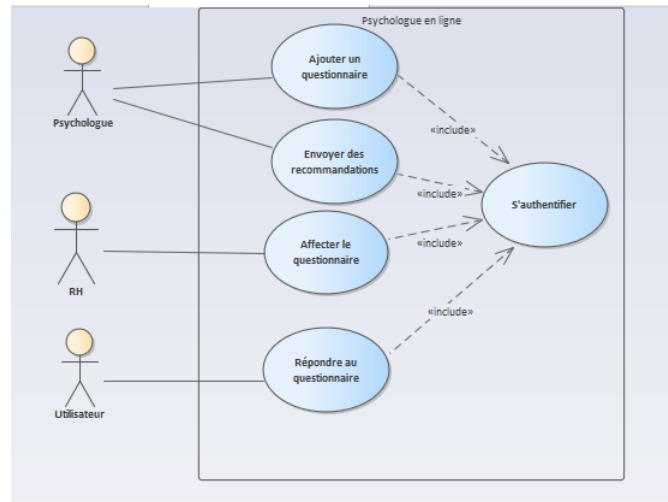


Figure 1: Diagramme de cas d'utilisation

2.3 Description textuelle des principaux cas d'utilisation

- Description textuelle du cas d'utilisation "S'authentifier"

Cas d'utilisation	S'authentifier
Acteur	Utilisateur, RH et Psychologue
Objectif	Permet aux acteurs d'accéder à son propre espace
Précondition	L'acteur doit posséder un compte
Post condition	Permet à l'acteur d'accéder à son propre espace
Scénario nominal	<p>1-Le système invite l'acteur à entrer son login et son mot de passe.</p> <p>2-L'acteur saisit le login et le mot de passe.</p> <p>3-le système vérifie les paramètres.</p> <p>4-Le système affiche l'espace correspondant à l'acteur.</p> <p>5-l'instance de cas d'utilisation se termine.</p>
Exception	Si un des champs est erroné , le système actualise à nouveau la page

- Description textuelle du cas d'utilisation "Ajouter un questionnaire"

Cas d'utilisation	Ajouter le questionnaire
Acteur	Psychologue
Précondition	L'acteur accède à l'application
Post condition	Ajouter le questionnaire
Scénario nominal	<p>1-Une mappe s'affiche contient la case d'entrer le nom de l'acteur(Psychologue) le choix de sélectionner l'utilisateur une autre case pour rentrer la question et la dernière pour le choix de charger en format CSV.</p> <p>2-L'acteur rentre les informations suivantes.</p> <p>3-L'acteur fait envoyer les informations à l'aide du boutons envoyer.</p>
Exception	Pas d'exception

- Description textuelle du cas d'utilisation "Affecter le questionnaire"

Cas d'utilisation	Affecter le questionnaire
Acteur	RH
Précondition	L'acteur accède à l'application
Post condition	Approuver le questionnaire
Scénario nominal	1-Une mappe s'affiche contient un tableau des noms des utilisateurs avec leurs psychologues et les états. 2-L'acteur peut approuver le questionnaire par sélectionner oui dans la case état et le rejeter par sélectionner non 3-L'acteur fait envoyer les informations à l'aide du boutons envoyer.
Exception	Pas d'exception

- Description textuelle du cas d'utilisation "Répondre au questionnaire"

Cas d'utilisation	Répondre au questionnaire
Acteur	Utilisateur
Précondition	L'acteur accède à l'application
Post condition	Répondre au questionnaire
Scénario nominal	1-Une mappe s'affiche contient des questions et une case pour sélectionner le choix 2-L'acteur peut répondre par oui ou non en sélectionnant dans le case répondre 3-L'acteur fait envoyer les informations à l'aide du boutons envoyer.
Exception	Pas d'exception

2.4 Conclusion

Nous avons essayé tout le long de cette partie de présenter les besoins fonctionnels et non fonctionnels de l'application pour éclaircir et faciliter la compréhension des tâches à réaliser dont le but de faire une conception qui fera l'objet de la partie suivante.

3 Modélisation conceptuelle

3.1 Introduction

La conception est une étape importante dans la réalisation de l'application. Son objectif principal est de présenter une architecture stable qui permet de définir

la réalisation et le fonctionnement du système et aussi éliminer les risques techniques.

Pour donner une description abstraite du système, on est intéressé de faire une modélisation de notre application via un diagramme de classe qui va nous offrir une représentation simplifiée du système permettant de comprendre et de stimuler ses activités et ses motivations internes.

3.2 Diagramme de classes

Nous allons d'abord présenter le diagramme de classe de notre application avec ses éléments.

Le diagramme de classes représente la structure statique d'un système. Il contient les classes, leurs attributs ainsi que leurs associations.

L'intérêt principal de ce diagramme de classe est de modéliser les entités du système informatique.

3.2.1 Présentation des classes

Une classe représente la structure d'un objet, la déclaration de l'ensemble des entités qui le composent. Une classe est composée donc de:

- Des attributs : il s'agit des données, dont les valeurs représentent l'état de l'objet.
- Des méthodes : il s'agit des applications applicables aux objets.

Les classes qui forment notre application sont les suivantes:

- **Utilisateur:** représente les personnes déjà inscrit dans l'application et qui répond sur le questionnaire ou le formulaire.
- **Psychologue:** elle représente l'acteur qui fournit les questions.
- **RH:** l'acteur qui approuve le formulaire.

- **Formulaires:** la classe qui contient les informations des utilisateur, les psychologues et les états.
- **Questions:** la classe contient les questions et les réponses.
- **Notification:** l'entité qui contient notifications relatives à la réponse des utilisateurs envoyée au RH
- **Recommandation:** un objet qui représente une recommandation envoyée par le psychologue à l'utilisateur

3.2.2 Présentations des attributs et des méthodes

Les attributs et les méthodes de nos classes sont présentés dans le tableau :

Nom classe	La liste des attributs	Les méthodes
Utilisateur	user mot_de_passe	getuser() getpasse() authentification() répondre()
Psychologue	psy mot_de_passe	getpsy() getpasse() authentification() insérer()
RH	RH mot_de_passe	getRH() getpasse() authentification() approuver()
Formulaires	id_formulaire utilisateur psychologue etat	getuser() getpsy() getid_formulaire() getetat() setuser() setpsy() setetat()
Questions	id_question id_formulaire question reponse	getid_question() getid_formulaire() getquestion() getreponse() setid_question() setid_formulaire() setquestion() setreponse()
notification	id notification	getid() getnotification() envoienotification()
Recommandation	id objet contenuRecommandation	getid() getobjet() getcontenuRecommandation() setcontenuRecommandation() envoiRecommandation()

3.2.3 Diagramme de classe

La figure ci-dessous récapitule le tableau précédent dans un diagramme de classe réalisé à l'aide de l'outil Enterprise Architect et qui englobe toutes les informations qu'on a cité avant:

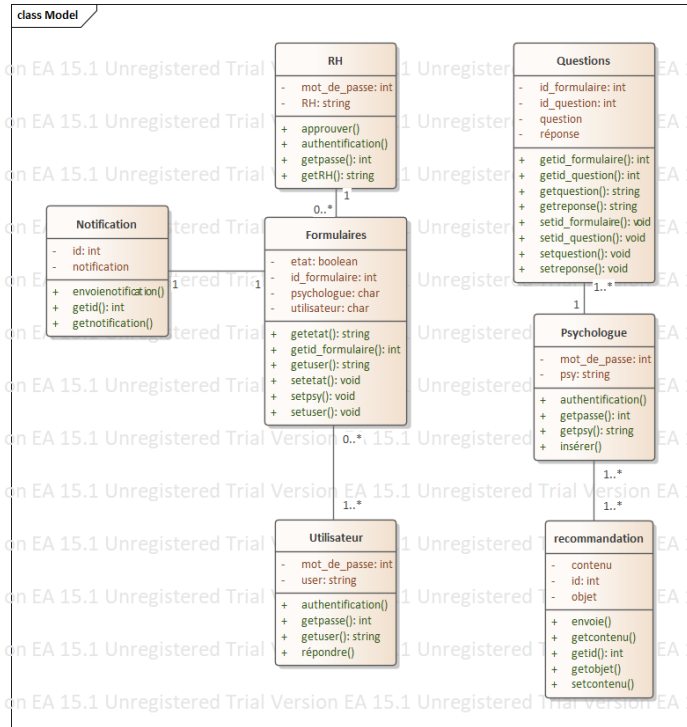


Figure 2: Diagramme de classe

4 Page Login

4.1 Page d'identification

Créons premièrement un fichier Login.jsp dans lequel on aura la description de la page d'identification des différents utilisateurs.

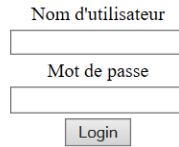
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Login</title>
```

```

6  </head>
7  <body>
8  <div align = "center">
9    <form action="login" method="post">
10     <table>
11       <tr>
12         <td align ="center">Nom d'utilisateur</td>
13       </tr>
14       <tr>
15         <td><input type="text" name="nom"></td>
16       </tr>
17       <tr>
18         <td align ="center">Mot de passe</td>
19       </tr>
20       <tr>
21         <td><input type="password" nom="mot de passe"></td>
22       </tr>
23       <tr>
24         <td align ="center"><input type="submit" value="Login"></td>
25       </tr>
26     </table>
27   </form>
28 </div>
29 </body>
30 </html>

```

On aura le resultat simple suivant



The image shows a simple login form. It consists of two text input fields stacked vertically. The first field is labeled 'Nom d'utilisateur' and the second is labeled 'Mot de passe'. Below the second field is a button labeled 'Login'.

Figure 3: Login.jsp

4.2 Connexion avec une base de données MySQL : vérification des identifiants

4.2.1 Classe Session

On aura besoin d'une classe *Session* (package : loginseccion) qui va récupérer pendant chaque identification le nom et le mot de passe entrés.

```
1
2 package loginseccion;
3
4 public class session {
5     private String nom;
6     private String passe;
7     public String returnNom() {
8         return nom;
9     }
10    public String affecteNom(String nom) {
11        this.nom = nom;
12    }
13    public String returnPasse() {
14        return passe;
15    }
16    public String affectePasse(String passe) {
17        this.passe = passe;
18    }
19 }
```

4.2.2 Classe DB

La classe DB (Package : base_donnees) va permettre dans un premier lieux la connexion avec une base de données MySQL (userdb) qu'on va créer par la suite, puis vérifie si les identifiants (nom et mot de passe) entrés figurent dans cette base.

```
1 package base_donnees;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import loginsession.*;
8 public class DB {
9     private String dbUrl = "jdbc:mysql://localhost:3306/userdb?
10                             useJDBCCompliantTimezoneShift=true&
11                             useLegacyDatetimeCode=false&serverTimezone=UTC";
12     private String dbUname = "AzzimDriouich";
13     private String dbPassword = "0000";
14     private String dbDriver = "com.mysql.cj.jdbc.Driver";
15     public void loadDriver(String dbDriver)
16     {
17         try {
18             Class.forName(dbDriver);
19         } catch (ClassNotFoundException e) {
20             e.printStackTrace();
21         }
22     }
23     public Connection getConnection()
24     {
25         Connection con = null;
26         try {
27             con = DriverManager.getConnection(dbUrl, dbUname, dbPassword);
28         } catch (SQLException e) {
29             e.printStackTrace();
30         }
31     }
32 }
```

```

31         return con;
32     }
33     public boolean valider_donnees(Session session)
34     {
35         boolean status = false;
36
37         loadDriver(dbDriver);
38         Connection con = getConnection();
39         String sql = "SELECT *
40                       FROM login
41                       WHERE nom = ?
42                       AND mot_de_passe =?";
43         PreparedStatement ps;
44         try {
45             ps = con.prepareStatement(sql);
46             ps.setString(1, session.returnNom());
47             ps.setString(2, session.returnPasse());
48             ResultSet rs = ps.executeQuery();
49             status = rs.next();
50
51         } catch (SQLException e) {
52             e.printStackTrace();
53         }
54         return status;
55     }
56 }

```


4.2.3 Création de la base de données "userdb"

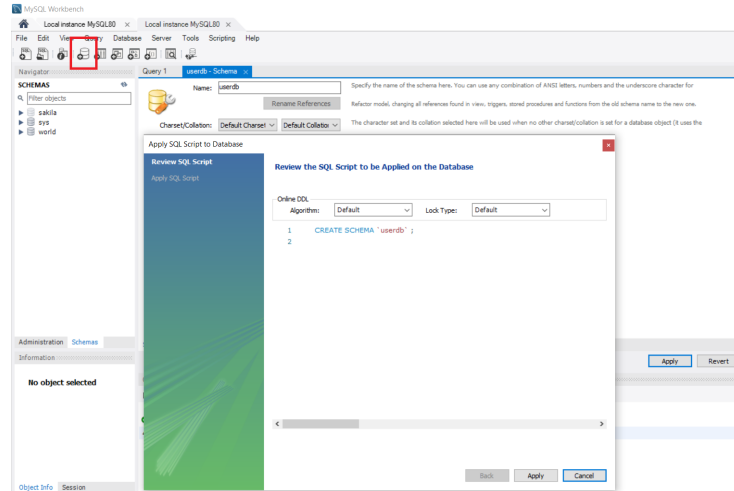


Figure 4: create schema

Après, on doit créer note tableau *login* avec les deux colonnes *nom* (Clé primaire et non null) et *mot_de_passe* (non null).

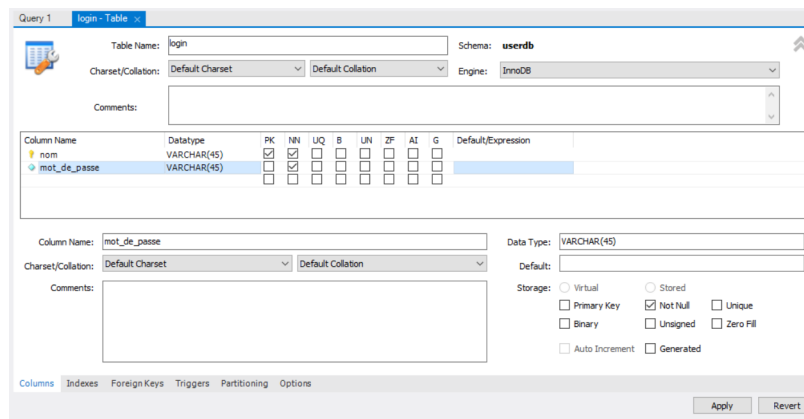


Figure 5: Création de la table

Afin de se connecter, on va insérer quelques utilisateurs à la table *login*.

Review the SQL Script to be Applied on the Database

```
1 INSERT INTO `userdb`.`login` (`nom`, `mot_de_passe`) VALUES ('Azzim', '111');
2 INSERT INTO `userdb`.`login` (`nom`, `mot_de_passe`) VALUES ('Driouich', '222');
3
```

Figure 6: Insertion des utilisateurs

4.3 Login servlet

Dans le package web, introduisant la première servlet qui va se servir de l'authentification et diriger l'utilisateur vers son compte si les identifiants sont corrects ou actualiser la page login sinon.

Pour cela, ajoutons un simple fichier Succes.jsp

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Succes</title>
6 </head>
7 <body>
8 Succes
9 </body>
10 </html>
```

Puis la servlet serait comme suit :

```
1 package web;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.annotation.WebServlet;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9 import base_donnees.*;
10 import loginsession.*;
11
12 @WebServlet("/login")
13 public class LoginServlet extends HttpServlet {
14
15     protected void doPost(HttpServletRequest request,
16                           HttpServletResponse response)
17     throws ServletException, IOException {
18         String nom = request.getParameter("nom");
19         String passe = request.getParameter("mot de passe");
20
21         Session session = new Session();
22         session.affecteNom(nom);
23         session.affectePasse(passe);
24
25         DB connexion_db = new DB();
26         if(connexion_db.valider_donnees(session)) {
27             response.sendRedirect("Succes.jsp");
28         }
29         else {
30             response.sendRedirect("login.jsp");
31         }
32     }
33 }
```

4.4 Teste de login

Exécutons le programme et essayons une authentification avec l'un des utilisateurs déclarés dans la base de données :

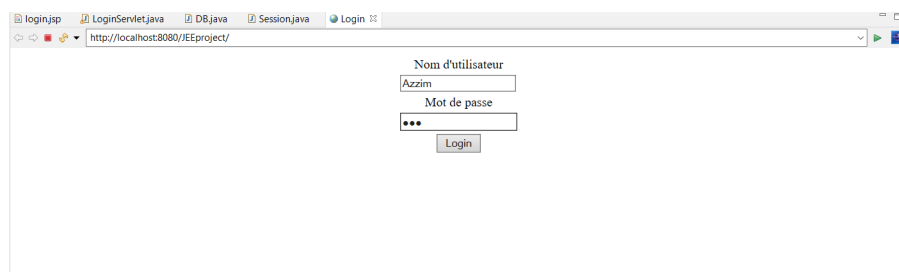


Figure 7: Exécution

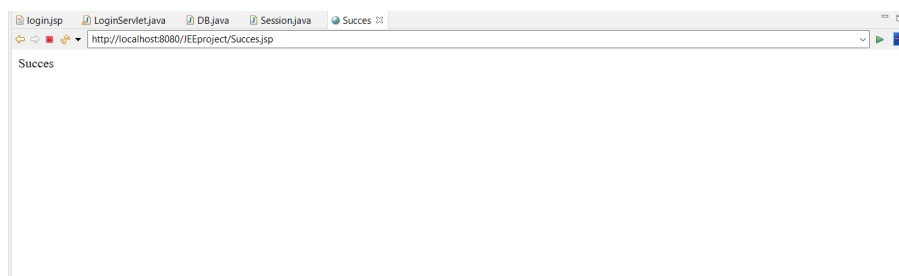
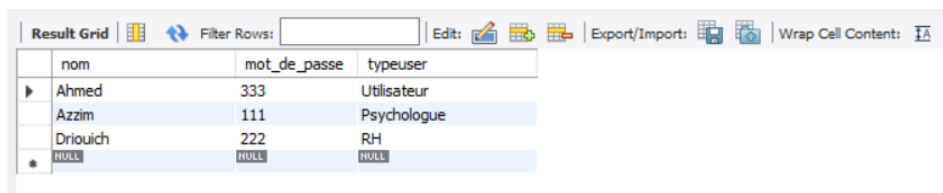


Figure 8: Authentification avec succès

4.5 Configuration des différents acteurs : Psychologue, RH et Utilisateur

Après l'authentification, l'inscrit doit être dirigé vers sa page personnelle. On distingue entre 3 type d'inscrits : Psychologue, RH et utilisateur. De ce fait, on doit modifier la table *login* en ajoutant la colonne *type*.



	nom	mot_de_passe	typeuser
▶	Ahmed	333	Utilisateur
	Azzim	111	Psychologue
	Driouch	222	RH
*	NULL	NULL	NULL

Figure 9: Table login

Ensuite on doit modifier la classe *Session* en ajoutant le String type et les méthodes affectType et returnType.

```
1 package loginsession;
2
3 public class Session {
4     private String nom;
5     private String passe;
6     private String typeUser;
7     public void affectType(String typeUser) {
8         this.typeUser = typeUser;
9     }
10    public String returnType() {
11        return typeUser;
12    }
13    public String returnNom() {
14        return nom;
15    }
16    public void affecteNom(String nom) {
17        this.nom = nom;
18    }
19    public String returnPasse() {
20        return passe;
21    }
22    public void affectePasse(String passe) {
23        this.passe = passe;
24    }
25 }
```

Pour la classe DB, pendant la validation des identifiants (valide_donnees), on récupère le type de l'inscrit et on affecte sa valeur à userType en faisant appel à affectType

```
1 public boolean valider_donnees(Session session)
2 {
3     boolean status = false;
4
5     loadDriver(dbDriver);
6     Connection con = getConnection();
7     String sql = "select * from login where nom =? and mot_de_passe =?";
8     PreparedStatement ps;
9     try {
10        ps = con.prepareStatement(sql);
11        ps.setString(1, session.returnNom());
12        ps.setString(2, session.returnPasse());
13        ResultSet rs = ps.executeQuery();
14        status = rs.next();
15        session.affectType(rs.getString("typeuser"));
16    } catch (SQLException e) {
17        e.printStackTrace();
18    }
19    return status;
20 }
```

Finalement la servlet doit diriger chaque type d'inscrit vers sa page personnelle (Psychologue.jsp, RH.jsp ou Utilisateur.jsp)

```
1  @WebServlet("/login")
2  public class LoginServlet extends HttpServlet {
3
4      protected void doPost(HttpServletRequest request,
5                          HttpServletResponse response)
6      throws ServletException, IOException {
7          String nom = request.getParameter("nom");
8          String passe = request.getParameter("mot de passe");
9
10         Session session = new Session();
11         session.affecteNom(nom);
12         session.affectePasse(passe);
13         DB connexion_db = new DB();
14         if(connexion_db.valider_donnees(session)) {
15             if(session.returnType().equals("Psychologue")) {
16                 response.sendRedirect("Psychologue.jsp");
17             }
18             else if(session.returnType().equals("Utilisateur")) {
19                 response.sendRedirect("Utilisateur.jsp");
20             }
21             else if(session.returnType().equals("RH")) {
22                 response.sendRedirect("RH.jsp");
23             }
24         }
25         else {
26             response.sendRedirect("login.jsp");
27         }
28     }
29
30 }
```

4.6 Partie CSS

Pour finir cette partie, introduisant un fichier css *loginCSS.css* pour les raisons esthétiques.

```
1  button {
2      background-color: white;
3      color: black;
4      padding: 14px 20px;
5      border-radius: 10px;
6      margin: 8px 0;
7      border: none;
8      cursor: pointer;
9      width: 100%;
10 }
11 input[type=text], input[type=password] {
12     width: 100%;
13     border-radius: 10px;
14     padding: 12px 20px;
15     margin: 8px 0;
16     display: inline-block;
17     border: 1px solid #ccc;
18     box-sizing: border-box;
19 }
20 button:hover {
21     opacity: 0.8;
22     background-color: black;
23     color: white;
24 }
25 body {
26     background-image: url("1.gif");
27     background-attachment: fixed;
28     background-repeat: no-repeat;
29     background-position: center;
30     background-size: cover;
31     background-color: white;
32 }
```



```

33
34 .login {
35     overflow: hidden;
36     opacity: 0.8;
37     background-color: #8e8e8e;
38     padding: 20px 30px 30px 30px;
39     border-radius: 10px;
40     top: 100px;
41     width: 400px;
42     box-shadow: 5px 10px 10px rgba(green, 0.2);
43 }

```

le fichier *login.jsp* sera aussi changé

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Login</title>
6 </head>
7 <link href="loginCSS.css" rel="stylesheet" type="text/css"><body>
8 <div align = "center">
9 <form class = login action="login" method="post">
10 <table>
11 <tr>
12 <td><input type="text" name="nom" placeholder="Nom d'utilisateur"></td>
13 </tr>
14 <tr>
15 <td><input type="password" name="mot de passe" placeholder="Password"></td>
16 </tr>
17 <tr>
18 <td align ="center"><button type="submit" value="Login">login</button></td>
19 </tr>
20 </table>
21 </form>
22 </div>
23 </body>
24 </html>

```

Résultat final :



Figure 10: Page login

5 Conception de la base de données

De plus du tableau *login* on aura besoin d'autres pour stocker les questions posés par les psychologues et les réponses récupérés de la part des utilisateurs.

On propose l'ajout de deux tableaux :

- **Formulaires** : renferme l'id du formulaire (Clé primaire), le nom du psychologue (Créateur du formulaire), le nom d'utilisateur (Destinataire) et l'état du formulaire (Approuvé ou non par le RH).
- **Questions** : contient la question, son l'id (Clé primaire), l'id du formulaire où se trouve et les réponses fournies par les utilisateurs.

Formulaires			
id_formulaire	utilisateur	psychologue	etat
.	.	.	.
.	.	.	.
.	.	.	.

Questions			
id_question	question	id_formulaire	reponse
.	.	.	.
.	.	.	.
.	.	.	.

La création des deux tableaux consiste à lancer les deux requêtes :

```

1  CREATE TABLE `userdb`.`formulaires` (
2    `id_formulaire` INT NOT NULL,
3    `nom` VARCHAR(45) NOT NULL,
4    `psychologue` VARCHAR(45) NOT NULL,
5    `etat` TINYINT(1) NOT NULL,
6    PRIMARY KEY (`id_formulaire`));
7

```

Figure 11: table formulaires

```

1  CREATE TABLE `userdb`.`questions` (
2    `id_question` INT NOT NULL,
3    `id_formulaire` INT NOT NULL,
4    `question` MEDIUMTEXT NOT NULL,
5    `reponse` TINYINT(1) NULL,
6    PRIMARY KEY (`id_question`));
7

```

Figure 12: table questions

NOTE : Pour éviter la confusion entre les id des formulaires et ceux des questions, les id des formulaires débuteraient toujours par un "10" (exemple : 101, 102 ...) alors que ceux des questions débuteraient avec un "20" (exemple : 201 202 ...)

Insérant par suite quelques données dans les tables pour qu'elles nous aident pendant la construction des interfaces des utilisateurs (formulaire 101 avec 2 questions 201 et 202 et formulaire 102 avec une question 203)

```

1  INSERT INTO `userdb`.`formulaires` (`id_formulaire`, `nom`, `psychologue`, `etat`) VALUES ('101', 'Ahmed', 'Azzim', '1');
2  INSERT INTO `userdb`.`formulaires` (`id_formulaire`, `nom`, `psychologue`, `etat`) VALUES ('102', 'Karima', 'Azzim', '0');
3

```

Figure 13: table formulaires

```

1  INSERT INTO `userdb`.`questions` (`id_question`, `id_formulaire`, `question`) VALUES ('201', '101', 'Ouvrais-tu un enveloppe contenant la date de ta propre mort ?');
2  INSERT INTO `userdb`.`questions` (`id_question`, `id_formulaire`, `question`) VALUES ('202', '101', 'Pourrais-tu etre ami avec toi meme ?');
3  INSERT INTO `userdb`.`questions` (`id_question`, `id_formulaire`, `question`) VALUES ('203', '102', 'Avez-vous deja ete surpris par la qualite du travail que quelqu'un vous a presente ?');
4

```

Figure 14: table questions

Maintenant que tout est prêt, construisons les interfaces des utilisateurs.

6 Interface "Utilisateur"

L'utilisateur doit être capable de voir les questions du formulaire fournit par le psychologue et approuvé par le RH. Il doit aussi avoir le droit de répondre à chaque question avec oui ou non. Finalement, il confirme ses réponses pour qu'elles soient envoyés au psychologue.

6.1 Classe Question

La première étape consiste à créer une classe *Question* (Similaire à *Session*) dont les attributs sont les colonnes du tableau Questions.

```

1  package loginsession;
2
3  public class Question {
4      public int id_question;
5      public int id_formulaire;
6      public String questiontext;
7      public boolean reponse;
8
9      public void affectIdQuestion(int id_question) {

```

```

10         this.id_question =id_question;
11     }
12     public int returnIdQuestion() {
13         return id_question;
14     }
15     public void affectIdFormulaire(int id_formulaire) {
16         this.id_formulaire =id_formulaire;
17     }
18     public int returnIdformulaire() {
19         return id_formulaire;
20     }
21     public String returnQuestion() {
22         return questiontext;
23     }
24     public void affectQuestion(String questiontext) {
25         this.questiontext = questiontext;
26     }
27     public boolean returnReponse() {
28         return reponse;
29     }
30     public void affectReponse(boolean reponse) {
31         this.reponse =reponse;
32     }
33 }

```

6.2 Chargement des questions depuis la base de données

Ensuite, on déclare la fonction `question_utilisateur()` dans la classe *DB* qui prend en argument une *Session* et une liste des objets de la classe *Question* puis ajoute dans cette liste les questions du formulaire envoyé à cet utilisateur, qui sont approuvés par le RH et n'ont pas encore de réponses (Champ null).

Le tableau désiré est résultat de la requête :

```

SELECT *
FROM userdb.questions q
INNER JOIN userdb.formulaires f
ON f.id_formulaire = q.id_formulaire

```

WHERE f.nom=?
AND f.etat=1
AND q.reponse is NULL

Le champ f.nom serait remplis par session.nom.

```

1  public void question_utilisateur(Session session, List<Question> userquestion){
2      boolean status;
3
4      loadDriver(dbDriver);
5      Connection con = getConnection();
6      String sql = "SELECT *
7                      FROM userdb.questions q
8                      INNER JOIN userdb.formulaires f
9                      ON f.id_formulaire = q.id_formulaire
10                     WHERE f.nom=? AND f.etat=1 AND q.reponse is NULL";
11      PreparedStatement ps;
12      try {
13          ps = con.prepareStatement(sql);
14          ps.setString(1, session.returnNom());
15          ResultSet rs = ps.executeQuery();
16          status = rs.next();
17          while(status) {
18              Question question = new Question();
19              question.affectIdQuestion(rs.getInt("id_question"));
20              question.affectIdFormulaire(rs.getInt("id_formulaire"));
21              question.affectReponse(rs.getBoolean("reponse"));
22              question.affectQuestion(rs.getString("question"));
23
24              userquestion.add(question);
25              status = rs.next();
26          }
27      } catch (SQLException e) {
28          e.printStackTrace();
29      }
30  }

```

6.3 Affichage des questions

Finalement, dans la partie *Utilisateur* de la servlet, on introduit la liste *userquestion* des objets de la classe *Question* qui serait remplies par la fonction *question_utilisateur()* et envoyée vers *Utilisateur.jsp*. Au cas où il n'y a pas de questions pour le moment, l'utilisateur serait redirigé vers *noQuestion.jsp*.

```
1  else if(session.returnType().equals("Utilisateur")) {
2      List<Question> userquestion = new ArrayList<Question>();
3      connexion_db.question_utilisateur(session, userquestion);
4      if(userquestion.size() == 0)
5          response.sendRedirect("noQuestions.jsp");
6      else {
7          request.setAttribute("userquestion", userquestion);
8          request.setAttribute("utilisateur", session.returnNom());
9          RequestDispatcher rst = request.getRequestDispatcher("Utilisateur.jsp");
10         rst.forward(request, response);
11     }
12 }
```

Finalement, la liste des questions peut être affichée à l'aide de *foreach* dans *Utilisateur.jsp*.

```
1  <html>
2  <head>
3  <meta charset="UTF-8">
4  <title>${utilisateur}</title>
5
6  </head>
7  <link href="UtilisateurCSS.css" rel="stylesheet" type="text/css">
8  <body>
9  <form action="utilisateur" method="post">
10     <br>
11     <br>
12     <h1 align=center>Veuillez repondre aux questions suivantes</h1>
13     <br>
14     <br>
15     <div class="container">
16         <table align=center>
17             <thead>
18             <tr>
```

```

19     <th>Question</th>
20     <th>Reponse</th>
21 </tr>
22 </thead>
23 <tbody>
24   <c:forEach items="${userquestion}" var="question">
25     <tr>
26       <td><c:out value="${question.returnQuestion()}" /></td>
27       <input type="hidden" name="idQuestion" value="${question.returnIdQuestion()}" />
28       <td><select name="reponse" class="dropbtn">
29         <option value="1">Oui</option>
30         <option value="0">Non</option>
31       </select></td>
32     </tr>
33   </c:forEach>
34 </tbody>
35 <thead>
36   <tr>
37     <td></td>
38     <th align=center><button type="submit">Envoyer</button></th>
39   </tr>
40 </thead>
41 </table>
42 </div>
43 </form>
44 </body>
45 </html>

```

6.4 Partie CSS

Le fichier CSS décrivant la page utilisateur est donné par :

```

1 @import url(https://fonts.googleapis.com/css?family=Open+Sans:400,600);
2 h1{
3     font-size: 20px;
4     font-family: 'Courier New';
5     color : white;
6 }
7 button {
8     background-color: white;

```



```

9     color: black;
10    display:inline-block;
11    padding: 10px 30px;
12    border-radius: 10px;
13    font-size: 20px;
14    margin: 30px 0;
15    border: none;
16    cursor: pointer;
17    display: inline-block;
18    transition: all 0.25s;
19 }
20
21 button:hover {
22     opacity: 0.8;
23     background-color: #E2B842;
24     color: white;
25 }
26
27 *, *:before, *:after {
28     margin: 0;
29     padding: 0;
30     box-sizing: border-box;
31 }
32 .dropbtn {
33     background-color: #012B39;
34     color: #ffffff;
35     padding: 10px 30px;
36     font-size: 15px;
37     border: none;
38     cursor: pointer;
39 }
40 body {
41     background: #105469;
42     font-family: 'Open Sans', sans-serif;
43 }
44 table {

```

```

45     width: 70%;
46     background: #012B39;
47     border-radius: 0.25em;
48     border-collapse: collapse;
49     margin: 1em;
50 }
51 th {
52     border-bottom: 1px solid #364043;
53     color: #E2B842;
54     font-size: 20px;
55     font-weight: 600;
56     padding: 0.5em 1em;
57     text-align: Left;
58 }
59 td {
60     color: #fff;
61     font-weight: 400;
62     padding: 0.65em 1em;
63 }
64 .disabled td {
65     color: #4F5F64;
66 }
67 tbody tr {
68     transition: background 0.25s ease;
69 }
70 tbody tr:hover {
71     background: #014055;
72 }

```

6.5 Récupération des réponses par la base de données

Dans la classe DB, on introduit la fonction *updateQuestions()* qui prend en argument une liste des id des question et une autre contenant les réponses. Puis insère les réponses correspondant aux id dans la base de données en exécutant la requête :

```
UPDATE 'userdb'. 'questions'
```

```
SET 'reponse' = ?
WHERE ('id_question' = ?)
```

```

1 public void updateQuestion(int[] idQuestions, int[] reponses){
2     loadDriver(dbDriver);
3     Connection con = getConnection();
4     String sql = "UPDATE 'userdb'.'questions'
5                 SET 'reponse' = ?
6                 WHERE ('id_question' = ?)";
7     PreparedStatement ps;
8     try {
9         for(int j = 0 ; j < idQuestions.length ; j++) {
10             ps = con.prepareStatement(sql);
11             ps.setInt(1, reponses[j]);
12             ps.setInt(2, idQuestions[j]);
13             ps.executeUpdate();
14         }
15     } catch (SQLException e) {
16         e.printStackTrace();
17     }
18 }
```

Le bouton **'Envoyer'** fait appel à la servlet *Utilisateur* qui, à son rôle, extrait les réponses et les id des questions et les insèrent dans la base de données à l'aide de *updateQuestion()*.

```

1 package web;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.annotation.WebServlet;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9 import base_donnees.DB;
10
11 @WebServlet("/utilisateur")
12 public class Utilisateur extends HttpServlet {
13     protected void doPost(HttpServletRequest request,
```

```

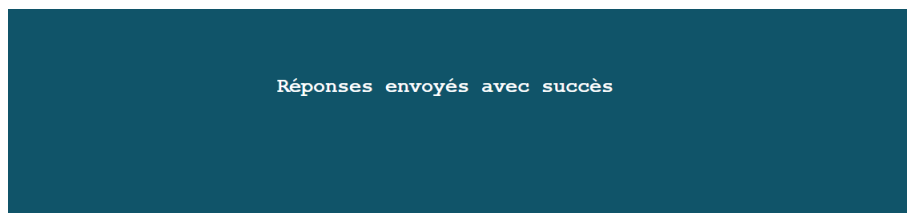
14         HttpServletResponse response)
15     throws ServletException, IOException {
16         String[] reponsesExtraits = request.getParameterValues("reponse");
17         String[] idQuestionsExtraits = request.getParameterValues("idQuestion");
18         int[] idQuestions = new int[idQuestionsExtraits.length];
19         int[] reponses = new int[reponsesExtraits.length];
20         for(int i =0 ;i<idQuestionsExtraits.length;i++) {
21             idQuestions[i] = Integer.parseInt(idQuestionsExtraits[i]);
22             reponses[i] = Integer.parseInt(reponsesExtraits[i]);
23         }
24         DB db_connexion = new DB();
25         db_connexion.updateQuestion(idQuestions, reponses);
26         response.sendRedirect("EnvoieSucces.jsp");
27     }
28 }

```

Résultat donné pour utilisateur "Ahmed" :

Figure 15: Interface utilisateur

Au cas de succès de l'envoi des questions la page ci dessous s'affiche



Au cas où aucune question n'est posée pour cet utilisateur la page suivante s'affiche



Pas de questions pour le moment

7 Interface "RH"

Un RH doit être capable de voir les noms des utilisateurs accompagnés par les noms des psychologues pour les approuver ou les rejeter.

7.1 Classe Formulaire

Dans un premier pas on va créer une classe Formulaire dont les attributs sont les colonnes du tableau formulaires.

```
1 package loginsession;
2
3 public class Formulaire {
4     public int id_formulaire;
5     public String nom;
6     public String psychologue;
7     public boolean etat;
8
9     public String getuser() {
10         return nom;}
11
12     public void setuser(String nom) {
13         this.nom = nom;
14     }
15     public String getpsy() {
16         return psychologue;
```

```

17     }
18
19     public void setpsy(String psychologue) {
20         this.psychologue = psychologue;
21     }
22     public boolean returnetat() {
23         return etat;
24     }
25     public void affectetat(boolean etat) {
26         this.etat =etat;
27     }
28     public void affectIdFormulaire(int id_formulaire) {
29         this.id_formulaire = id_formulaire;
30     }
31
32     public int returnIdFormulaire() {
33         return id_formulaire;
34     }
35
36 }

```

7.2 Chargement des formulaires depuis la base de donnée

On déclare la fonction `liste_formulaire()` qui prend en argument deux paramètres *Session* et *liste des Objets de la classe Formulaire* puis il rajoute dans cette l'approuvement ou le rejet du RH.

```

1     public void liste_formulaire(Session session, List<Formulaire> RH) {
2         Statement statement = null;
3         ResultSet resultat = null;
4
5         loadDriver(dbDriver);
6         Connection connexion = getConnection();
7         try {
8             statement = connexion.createStatement();
9             resultat = statement.executeQuery("SELECT * FROM
10         userdb.formulaires;");

```

```

11         while (resultat.next()) {
12             String user = resultat.getString("nom");
13             String psy = resultat.getString("psychologue");
14             int id_formulaire = resultat.getInt("id_formulaire");
15             boolean etat = resultat.getBoolean("etat");
16             Formulaire formulaire = new Formulaire();
17             formulaire.setUser(user);
18             formulaire.affectEtat(etat);
19             formulaire.setPsy(psy);
20             formulaire.affectIdFormulaire(id_formulaire);
21             RH.add(formulaire);
22             System.out.println(user);           }
23     } catch (SQLException e) {
24         e.printStackTrace();}
25
26     }

```

7.3 Affichage de formulaire

Finalement, on fait un autre changement dans la partie RH de la servlet, on introduit la liste RH des objets de la classe *Formulaire* qui serait remplie par la fonction *liste_formulaire()* et envoyée vers RH.jsp.

```

1  else if(session.returnType().equals("RH")) {
2      List<Formulaire> RH = new ArrayList<Formulaire>();
3      connexion_db.liste_formulaire(session, RH);
4      if(RH.size() == 0)
5          response.sendRedirect("noFormulaire.jsp");
6      else {
7          request.setAttribute("RH", RH);
8          RequestDispatcher rst = request.getRequestDispatcher("RH.jsp");
9          rst.forward(request, response);}
10
11
12     }

```

Maintenant pour le fichier JSP ,la liste des utilisateurs et des psychologues est affichée par foreach

```

1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="UTF-8">
7  <title>${formulaire}</title>
8
9  </head>
10 <link href="RHCSS.css" rel="stylesheet" type="text/css">
11 <body>
12 <form action="RH" method="post">
13 <br>
14 <br>
15 <h1 align=center>Veuillez approuver les etats suivants</h1>
16 <br>
17 <br>
18 <div class="container">
19 <table align=center>
20 <thead>
21 <tr>
22 <th>Utilisateur</th>
23 <th>Psychologue</th>
24 <th>l'etat</th>
25 </tr>
26 </thead>
27 <tbody>
28 <c:forEach items="${RH}" var="formulaire">
29 <tr>
30 <td><c:out value="${formulaire.getuser()}" /></td>
31 <td><c:out value="${formulaire.getpsy()}" /></td>
32 <input type="hidden" name="id_formulaire" value="${formulaire.returnI
33 <td>
34 <select name="etat" class="dropbtn">
35 <option value="0">Non</option>
36 <option value="1">Oui</option>

```



```

37         </select></td>
38     </tr>
39 </c:forEach>
40 </tbody>
41 <thead>
42     <tr>
43         <td></td>
44         <th align=center><button type="submit" >Envoyer</button></th>
45     </tr>
46 </thead>
47 </table>
48 </div>
49 </form>
50 </body>
51 </html>

```

7.4 Partie CSS

Le fichier CSS décrivant la page RH est donné par:

```

1 @import url(https://fonts.googleapis.com/css?family=Open+Sans:400,600);
2 h1{
3     font-size: 20px;
4     font-family: 'Courier New';
5     color : white;
6 }
7 button {
8     background-color: white;
9     color: black;
10    display:inline-block;
11    padding: 10px 30px;
12    border-radius: 10px;
13    font-size: 20px;
14    margin: 30px 0;
15    border: none;
16    cursor: pointer;
17    display: inline-block;

```

```

18     transition: all 0.25s;
19 }
20
21 button:hover {
22     opacity: 0.8;
23     background-color: #E2B842;
24     color: white;
25 }
26
27 *, *:before, *:after {
28     margin: 0;
29     padding: 0;
30     box-sizing: border-box;
31 }
32 .dropbtn {
33     background-color: #012B39;
34     color: #ffffff;
35     padding: 10px 30px;
36     font-size: 15px;
37     border: none;
38     cursor: pointer;
39 }
40 body {
41     background: #105469;
42     font-family: 'Open Sans', sans-serif;
43 }
44 table {
45     width: 70%;
46     background: #012B39;
47     border-radius: 0.25em;
48     border-collapse: collapse;
49     margin: 1em;
50 }
51 th {
52     border-bottom: 1px solid #364043;
53     color: #E2B842;

```

```

54     font-size: 20px;
55     font-weight: 600;
56     padding: 0.5em 1em;
57     text-align: Left;
58 }
59 td {
60     color: #fff;
61     font-weight: 400;
62     padding: 0.65em 1em;
63 }
64 .disabled td {
65     color: #4F5F64;
66 }
67 tbody tr {
68     transition: background 0.25s ease;
69 }
70 tbody tr:hover {
71     background: #014055;
72 }

```

Et on obtient le resultat suivant:

Veuillez approuver les états suivants

Utilisateur	Psychologue	l'état
Ahmed	Azzim	Non <input checked="" type="checkbox"/>
Karima	Azzim	Non <input checked="" type="checkbox"/>

Envoyer

Figure 16: Interface RH

7.5 Récupération des états par la base de données

Revenant à la classe DB, on rajoute la fonction `updateliste_formulaire` qui prend en argument une liste des id des formulaires et une autre contenant les états. On peut insérer les états correspondant au id dans la base des données en exécutant la requête suivante:

```
UPDATE 'userdb'.'formulaires'
SET 'etat' = ?
WHERE ('id_formulaire' = ?)
```

```
1
2      public void updateliste_formulaire(int[] id_formulaire, int[] etat) {
3          loadDriver(dbDriver);
4          Connection con = getConnection();
5          String sql = "UPDATE 'userdb'.'formulaires'
6          SET 'etat' = ? WHERE ('id_formulaire' = ?)";
7          PreparedStatement ps;
8          try {
9              for(int j = 0 ; j < id_formulaire.length ; j++) {
10                  ps = con.prepareStatement(sql);
11                  ps.setInt(1, etat[j]);
12                  ps.setInt(2, id_formulaire[j]);
13                  ps.executeUpdate();
14              }
15              } catch (SQLException e) {
16                  e.printStackTrace();
17          }
18
19
20      }
```

Le bouton **Envoyer** il fait appel à son rôle à la servlet **RH** qui fait extraire les états et les id des formulaire et les insèrent dans la base des données à l'aide de `updateliste_formulaire()`.

```
1 package web;
2
3 import java.io.IOException;
```

```

4  import javax.servlet.ServletException;
5  import javax.servlet.annotation.WebServlet;
6  import javax.servlet.http.HttpServlet;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9
10 import base_donnees.DB;
11
12 @WebServlet("/RH")
13 public class RH extends HttpServlet {
14
15
16     protected void doPost(HttpServletRequest request, HttpServletResponse etats)
17
18
19     String[] etatExtraits = request.getParameterValues("etat");
20
21     String[] id_formulaireExtraits = request.getParameterValues("id_formulaire");
22
23     int[] id_formulaire = new int[id_formulaireExtraits.length];
24
25     int[] etat = new int[etatExtraits.length];
26
27     for(int i =0 ;i<id_formulaireExtraits.length;i++) {
28         id_formulaire[i] = Integer.parseInt(id_formulaireExtraits[i]);
29         etat[i] = Integer.parseInt(etatExtraits[i]);
30     }
31     DB db_connexion = new DB();
32
33     db_connexion.updateliste_formulaire(id_formulaire, etat);
34
35     etats.sendRedirect("EnvoieSucces.jsp");
36 }
37
38 }

```

Résultat donné pour un RH :

Veuillez approuver les états suivants

Utilisateur	Psychologue	l'état
Ahmed	Azzim	Non <input checked="" type="checkbox"/>
Karima	Azzim	Non <input checked="" type="checkbox"/>
ali	azzim	Non <input checked="" type="checkbox"/>

Envoyer

Après avoir cliquer sur le button **Envoyer**

Réponses envoyées avec succès

Au cas où aucun formulaire existe.

Pas de formulaire pour le moment