Anthony Di Stasio, Adrien Mousty

**2016 - 2017**

**Forum**

**Projet JEE**

Table des matières

[Enoncé 3](#_Toc469332132)

[Description du site web 4](#_Toc469332133)

[Fonctionnalités du forum 5](#_Toc469332134)

[Les différents diagrammes 6](#_Toc469332135)

[Diagramme des cas d’utilisations 6](#_Toc469332136)

[Diagramme de classes 6](#_Toc469332137)

[Diagrammes de séquences 6](#_Toc469332138)

[Base de données 7](#_Toc469332139)

[Schéma conceptuel 7](#_Toc469332140)

[Connexion oracle 8](#_Toc469332141)

[PL/SQL 8](#_Toc469332142)

[Les tables 8](#_Toc469332143)

[Les clés étrangères 8](#_Toc469332144)

[Les séquences 9](#_Toc469332145)

[Les procédures stockées 9](#_Toc469332146)

[Détails de certaines partie du code 11](#_Toc469332147)

[Java 11](#_Toc469332148)

[Lambda expression 11](#_Toc469332149)

[Procédures stockées 12](#_Toc469332150)

[Screenshots du site web 15](#_Toc469332151)

[Connexion 15](#_Toc469332152)

[Affichage propre à chaque utilisateur 15](#_Toc469332153)

[Vue générale du forum 15](#_Toc469332154)

[Nouveautés apportées 16](#_Toc469332155)

[Bootstrap 16](#_Toc469332156)

[GIT et GitHub 17](#_Toc469332157)

[GIT 17](#_Toc469332158)

[GitHub 18](#_Toc469332159)

[Problèmes rencontrés 19](#_Toc469332160)

[Conclusion 20](#_Toc469332161)

[Table des figures 21](#_Toc469332162)

[Bibliographie 22](#_Toc469332163)

# Enoncé

Réalisez le développement d’un site web par groupe de 2(ou 3) en utilisant les technologies vues au cours JEE à savoir :

* Des JSP
* Des servlets
* Mettre en place le modèle MVC
* Utiliser une base de données + DAO

Le développement se fera avec Eclipse Neon et GlassFish 4.0.

En Option :

* Mettre en place un service Web.
* Utiliser REST.
* Utilisation Ajax.

Ce projet sera à remettre le vendredi 13 janvier 2016 à 8h15’ dans le local de l’examen de programmation avancée. L’étudiant remettra à cette date un rapport expliquant son projet et la version électronique du projet (prendre le **Workspace)**.

Aucun délai supplémentaire ne sera accordé.

Si le projet n’est pas rendu dans les délais, l’étudiant se verra attribuer un 0 pour ce travail et donc pour l’examen du cours d’applications informatiques.

# Description du site web

Nous allons mettre en place sur un serveur un site web ayant pour but de servir de forum. Ce forum est destiné à être généraliste. Il y a aura plusieurs parties, qui eux-mêmes contiennent des sous-parties :

* Sport
  + Football
  + Cyclisme
  + Hockey
  + …
* Jeux-vidéo
  + PS4
  + Xbox One
  + PC
  + Wii U
  + PS3
  + iOS
  + Android
* Technologie
  + Programmation
  + Domotique
  + Réseaux
  + ….
* Blabla
  + -15 ans
  + 15-18 ans
  + 18-25 ans
  + 25-35 ans
  + +35 ans

Pour ce qui est de la gestion des utilisateurs, nous avons plusieurs types d’utilisateurs. Ceux-ci seront décrit plus en profondeur dans le chapitre précédent.

Quant à la gestion de leurs droits, nous avons choisi de traiter ça de manière empirique. Les personnes placées deux rangs au-dessus d’une autre hiérarchiquement parlant pourra la promouvoir.

*Ex : Un modérateur ne pourra pas promouvoir un utilisateur lambda. Par contre, un administrateur pourra le promouvoir en tant que modérateur.*

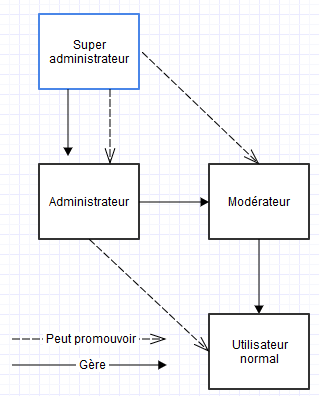
**

Figure 1 Hiérarchie

# Fonctionnalités du forum

* En tant qu’invité
  + Se connecter
  + S’inscrire
  + Voir certaines parties du forum
* En tant qu’utilisateur connecté
  + Modifier son profil
  + Supprimer son compte
  + Créer un sujet
  + Modifier/supprimer son(ses) sujet(s)
  + Répondre aux autres sujets
  + Gagner des points d’expérience (?)
  + Envoyer des messages privés aux autres utilisateurs
* En tant qu’utilisateur-modérateur
  + Gérer les sujets
    - Supprimer un sujet
    - Modifier un sujet
  + Donner des sanctions
    - Bannir un utilisateur (?)
* En tant qu’utilisateur-administrateur
  + Gérer les utilisateurs
    - Supprimer un utilisateur
    - Modifier un utilisateur
    - Ajouter un utilisateur
  + Voir les statistiques du forum
  + Voir l’historique de connexion

# Les différents diagrammes

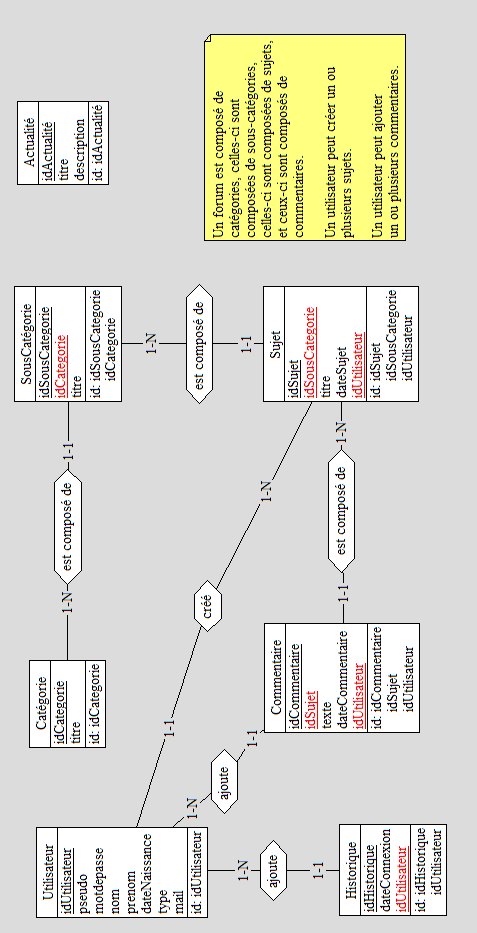
## Diagramme des cas d’utilisations

## Diagramme de classes

## Diagrammes de séquences

# Base de données

## Schéma conceptuel



## Connexion oracle

Expliquer comment nous avons fait la bd (avec sql developer)

Screen du code avec le driver etc

* Ajout d’une bibliothèque

## PL/SQL

### Les tables

Nous avons d’abord créé chaque table, avec leur clé primaire, et clé étrangère si nécessaire. En respectant cette syntaxe :

|  |
| --- |
| CREATE TABLE UTILISATEUR (  IDUTILISATEUR INTEGER NOT NULL,  PSEUDO VARCHAR2(100),  MOTDEPASSE VARCHAR2(100),  NOM VARCHAR2(100),  PRENOM VARCHAR2(100),  DATENAISSANCE DATE,  TYPEUTILISATEUR VARCHAR2(100),  MAIL VARCHAR2(100),  CONSTRAINT PK\_UTILISATEUR PRIMARY KEY (IDUTILISATEUR)  ); |

### Les clés étrangères

Pour ajouter les clés étrangères dans les tables ou nous en avions besoin, nous avons alors modifié les tables grâce à ce code :

Dans cet exemple, nous modifions la table « Historique », et comme celle-ci possède un attribut « IDUTILISATEUR », nous ajoutons alors une clé étrangère pour celui-ci.

|  |
| --- |
| ALTER TABLE HISTORIQUE  ADD CONSTRAINT FK\_HISTORIQUE\_UTILISATEUR FOREIGN KEY (IDUTILISATEUR)  REFERENCES UTILISATEUR (IDUTILISATEUR) ; |

### Les séquences

Etant donné qu’Oracle 11 ne propose pas de syntaxe déjà faite pour incrémenter les clés primaires, nous devons alors créer des séquences. Pour chaque table, nous aurons une séquence. Nous effectuons des « COMMIT » pour sauvegarder les informations (modifiées, créées, ou supprimées) dans la base de données.

Voici un exemple de séquence pour la table « Utilisateur » :

|  |
| --- |
| CREATE SEQUENCE SEQ\_UTILISATEUR START WITH 1 INCREMENT BY 1; |

Dans cette séquence, nous lui disons qu’elle doit commencer par 1, et qu’elle incrémente de 1 chaque ajout dans la base de données.

Par exemple, l’« IDUTILISATEUR », à chaque nouvel ajout, il sera incrémenté de 1, pour ne pas avoir de conflit de clé primaire.

### Les procédures stockées

Voici les procédures stockées que nous avons créé pour gérer notre base de données. Nous avons effectué ces procédures stockées pour chaque table présente dans notre base de données.

Nous allons expliquer les procédures stockées que nous effectuons sur la table « Utilisateur », car celles que nous avons utilisées pour les autres tables sont pareilles, la seule différence est le nombre de paramètre.

#### Insert

Pour effectuer un ajout dans la base de données, dans notre exemple ci-dessous, nous allons utiliser une procédure stockée nommée « INSERTUTILISATEUR ».

Celle-ci reçoit en paramètre ces attributs :

* Pseudo
* Mot de passe
* Nom
* Prénom
* Date de naissance
* Type
* Mail

Ensuite, nous effectuons une requête simple d’ajout, « INSERT INTO … ». Pour les valeurs que nous voulons entrer, nous devons aussi ajouter un id.

Comme nous avions créé des séquences auparavant, nous pouvons alors incrémenter cette séquence grâce à « NEXTVAL », nous prenons la valeur suivante de la base de données.

|  |
| --- |
| create or replace PROCEDURE INSERTUTILISATEUR (  P\_PSEUDO IN UTILISATEUR.PSEUDO%TYPE,  P\_MOTDEPASSE IN UTILISATEUR.MOTDEPASSE%TYPE,  P\_NOM IN UTILISATEUR.NOM%TYPE,  P\_PRENOM IN UTILISATEUR.PRENOM%TYPE,  P\_DATENAISSANCE IN UTILISATEUR.DATENAISSANCE%TYPE,  P\_TYPE IN UTILISATEUR.TYPEUTILISATEUR%TYPE,  P\_MAIL IN UTILISATEUR.MAIL%TYPE)  IS  BEGIN  INSERT INTO UTILISATEUR  VALUES (SEQ\_UTILISATEUR.NEXTVAL, P\_PSEUDO, P\_MOTDEPASSE, P\_NOM, P\_PRENOM, P\_DATENAISSANCE, P\_TYPE, P\_MAIL);    COMMIT;  END; |

#### Delete

Pour supprimer un utilisateur, nous utiliserons la procédure stockée « DELETEUTILISATEUR », celle-ci reçoit un paramètre, le pseudo de l’utilisateur.

Cette procédure stockée fera en sorte de supprimer l’utilisateur grâce à son pseudo.

|  |
| --- |
| create or replace PROCEDURE DELETEUTILISATEUR (  P\_PSEUDO IN UTILISATEUR.PSEUDO%TYPE)  IS  BEGIN  DELETE FROM UTILISATEUR  WHERE P\_PSEUDO = PSEUDO;    COMMIT;  END; |

#### Update

Pour mettre à jour ou modifier un utilisateur, une procédure stockée « UPDATEUTILISATEUR » est disponible. Celle-ci reçoit tous les attributs de la table « Utilisateur » sauf l’id, en paramètre.

Ensuite, nous modifions chaque attribut de la table « Utilisateur » par le paramètre reçu qui lui correspond.

L’utilisateur ne pourra pas modifier son pseudo. Nous faisons alors une condition sur celui-ci. La base de données modifiera tous les attributs ou son pseudo sera égal à celui reçu en paramètre.

|  |
| --- |
| create or replace PROCEDURE UPDATEUTILISATEUR (  P\_MOTDEPASSE IN UTILISATEUR.MOTDEPASSE%TYPE,  P\_NOM IN UTILISATEUR.NOM%TYPE,  P\_PRENOM IN UTILISATEUR.PRENOM%TYPE,  P\_DATENAISSANCE IN UTILISATEUR.DATENAISSANCE%TYPE,  P\_TYPE IN UTILISATEUR.TYPEUTILISATEUR%TYPE,  P\_MAIL IN UTILISATEUR.MAIL%TYPE,  P\_PSEUDO IN UTILISATEUR.PSEUDO%TYPE)  IS  BEGIN  UPDATE UTILISATEUR  SET MOTDEPASSE = P\_MOTDEPASSE, NOM = P\_NOM, PRENOM = P\_PRENOM, DATENAISSANCE = P\_DATENAISSANCE, TYPEUTILISATEUR = P\_TYPE, MAIL = P\_MAIL  WHERE PSEUDO = P\_PSEUDO;    COMMIT;  END; |

#### Select

Pour récupérer un utilisateur de la base de données, nous utilisons la procédure stockée « SELECTUTILISATEUR ». Nous le recherchons par rapport à l’id, nous avons alors une condition sur celui-ci.

Pour cette procédure stockée, il y a une différence par rapport aux autres, nous devons ressortir des informations, nous utilisons alors « OUT ». Et IN pour ceux qui entre en paramètre.

Ensuite, nous sélectionnons chaque attribut de la base de données et les stockons dans les variables sortantes, « O\_PSEUDO » par exemple. Et bien sûr, ou l’id correspond à celui reçu en paramètre.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE SELECTUTILISATEUR (  -- PARAMETRE ENTRANT  P\_IDUTILISATEUR **IN** UTILISATEUR.IDUTILISATEUR%TYPE,  -- LES PARAMETRES SORTANTS  O\_PSEUDO **OUT** UTILISATEUR.PSEUDO%TYPE,  O\_MOTDEPASSE **OUT** UTILISATEUR.MOTDEPASSE%TYPE,  O\_NOM **OUT** UTILISATEUR.NOM%TYPE,  O\_PRENOM **OUT** UTILISATEUR.PRENOM%TYPE,  O\_DATENAISSANCE **OUT** UTILISATEUR.DATENAISSANCE%TYPE,  O\_TYPE **OUT** UTILISATEUR.TYPEUTILISATEUR%TYPE,  O\_MAIL **OUT** UTILISATEUR.MAIL%TYPE)  IS  BEGIN  SELECT PSEUDO, MOTDEPASSE, NOM, PRENOM, DATENAISSANCE, TYPEUTILISATEUR, MAIL  INTO O\_PSEUDO, O\_MOTDEPASSE, O\_NOM, O\_PRENOM, O\_DATENAISSANCE, O\_TYPE, O\_MAIL  FROM UTILISATEUR WHERE IDUTILISATEUR = P\_IDUTILISATEUR;  END; |

#### GetList

Pour récupérer une liste dans une base de données, il suffit de faire :

|  |
| --- |
| SELECT \* FROM UTILISATEUR |

Mais comme demandé, nous avons effectué une procédure stockée qui fera le nécessaire. Dans cette procédure stockée, un paramètre sortant existe, il s’agit du curseur.

Nous utilisons alors un curseur pour manipuler les enregistrements. Ce curseur retourne tous les enregistrements de la table « Utilisateur ».

|  |
| --- |
| create or replace PROCEDURE GETLISTUTILISATEUR (  CUR\_UTILISATEUR **OUT** SYS\_REFCURSOR)  IS  BEGIN  OPEN CUR\_UTILISATEUR FOR  SELECT \* FROM UTILISATEUR;  END; |

# Détails de certaines partie du code

## Java

### Lambda expression

### Procédures stockées

#### Dans le DAO

Pour bien comprendre comment nous avons implémenter les procédures stockées dans notre programme, nous allons expliquer chaque méthode présente dans les classes DAO. Comme exemple, nous allons prendre la classe « UtilisateurDAO », car les autres classes possèdent quasiment les mêmes méthode, la seule différence dépend du nombre de paramètre de la procédure stockée.

##### Méthode « create() »

Pour implémenter les procédures stockées que nous avons créé en PL/SQL, nous avons dû modifier notre code dans les classes DAO.

Cela revient à presque pareil qu’avant, sauf qu’au lieu d’un « PreparedStatement », nous avons un « CallableStatement », car nous devons appeler cette procédure stockée. Cela ressemblera à ceci :

|  |
| --- |
| public final static String *INSERTUTILISATEUR* = "{CALL INSERTUTILISATEUR(?,?,?,?,?,?,?)}"; |

Nous avons créé ces appels aux procédures stockées de la base de données dans une classe appelée « Sprocs » pour bien séparer les couches. Ce « CALL » nécessite 7 paramètres d’où le nombre de « ? ».

Dans notre méthode create de la classe « UtilisateurDAO », nous définissons alors ces paramètres. Pour cette partie, le code est pareil qu’avant. Cette méthode sert à ajouter un utilisateur dans la base de données.

|  |
| --- |
| **public** **void** create(UtilisateurPOJO utilisateurPOJO) {  CallableStatement cst = **null**;  **try** {  //Appel de la procédure stockée pour ajouter un utilisateur  cst = connect.prepareCall(Sprocs.***INSERTUTILISATEUR***);    cst.setString (1, utilisateurPOJO.getPseudo());  cst.setString (2, utilisateurPOJO.getMotdepasse());  cst.setString (3, utilisateurPOJO.getNom());  cst.setString (4, utilisateurPOJO.getPrenom());  cst.setDate (5, (Date) utilisateurPOJO.getDateNaissance());  cst.setString (6, utilisateurPOJO.getType());  cst.setString (7, utilisateurPOJO.getMail());    cst.executeUpdate();  } **catch** (SQLException e) {  e.printStackTrace();  } **finally** {  **if** (cst != **null**) {  **try** {  cst.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  } |

##### Méthode « delete() »

Pour supprimer un utilisateur, comme dit pour la méthode « create() », nous avons aussi un appel à la procédure stockée dans la classe « Sprocs ». Celui ne nécessite qu’un seul paramètre.

|  |
| --- |
| **public** **final** **static** String ***DELETEUTILISATEUR*** = "{CALL DELETEUTILISATEUR(?)}"; |

La méthode « delete() » reçoit un objet de la classe UtilisateurPOJO comme paramètre. Nous récupérons le pseudo de cet objet et nous l’insérons comme paramètre de la procédure stockée. Car celle-ci supprime un utilisateur en fonction de son pseudo.

|  |
| --- |
| **public** **void** delete(UtilisateurPOJO utilisateurPOJO) {  CallableStatement cst = **null**;  **try** {  //Appel de la procédure stockée pour supprimer un utilisateur  cst = connect.prepareCall(Sprocs.***DELETEUTILISATEUR***);  cst.setString(1, utilisateurPOJO.getPseudo());  cst.executeUpdate();  } **catch** (SQLException e) {  e.printStackTrace();  } **finally** {  **if** (cst != **null**) {  **try** {  cst.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  } |

##### Méthode « update() »

Pour effectuer une mise à jour de l’utilisateur dans la base de données, nous avons un appel avec 7 paramètres. Un utilisateur pourra modifier tout sauf son pseudo.

|  |
| --- |
| **public** **final** **static** String ***UPDATEUTILISATEUR*** = "{CALL UPDATEUTILISATEUR(?,?,?,?,?,?,?)}"; |

Comme vous voyez dans le code ci-dessous, nous insérons alors les 7 paramètres dans la procédure stockée.

|  |
| --- |
| **public** **void** update(UtilisateurPOJO utilisateurPOJO) {  CallableStatement cst = **null**;  **try** {  cst = connect.prepareCall(Sprocs.***UPDATEUTILISATEUR***);    cst.setString (1, utilisateurPOJO.getMotdepasse());  cst.setString (2, utilisateurPOJO.getNom());  cst.setString (3, utilisateurPOJO.getPrenom());  cst.setDate (4, (Date) utilisateurPOJO.getDateNaissance());  cst.setString (5, utilisateurPOJO.getType());  cst.setString (6, utilisateurPOJO.getMail());  cst.setString (7, utilisateurPOJO.getPseudo());    cst.executeUpdate();  } **catch** (SQLException e) {  e.printStackTrace();  } **finally** {  **if** (cst != **null**) {  **try** {  cst.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  } |

##### Méthode « find() »

Appel à la procédure stockée « SELECTUTILISATEUR », celle-ci permet de récupérer un utilisateur grâce à son id.

Celle-ci est différente, car nous possédons des paramètres sortants, étant donné que nous récupérons des informations. 8 paramètres seront donc présents dans cette procédure stockée.

|  |
| --- |
| **public** **final** **static** String ***SELECTUTILISATEUR*** = "{CALL SELECTUTILISATEUR(?,?,?,?,?,?,?,?)}"; |

Pour rentrer des paramètres dans la procedure stockée, c’est pareil qu’avant. Par contre, pour récupérer des informations de cette procédure stockée, nous devons utiliser la méthode « registerOutParameter() » qui reçoit l’ordre, et le type de l’attribut concerné, en paramètre.

Par exemple, pour récupérer le pseudo, comme c’est le deuxième attribut dans notre table, nous mettons « 2 » dans cette méthode, et comme c’est un attribut de type texte, nous mettons « java.sql.Types.***VARCHAR***»***.***

Mais ce n’est pas fini, nous devons créer l’objet avec les informations retournées de la base de données. Nous créons alors cet objet, et nous faisons appel aux méthodes « getString() » ou « getInt()» etc, cela dépendra du type de variable reçue. Et nous insérons le numéro de l’ordre que nous avons déjà placé auparavant.

|  |
| --- |
| **public** UtilisateurPOJO find(**int** id) {  UtilisateurPOJO utilisateurPOJO = **null**;  CallableStatement cst = **null**;  **try** {  cst = connect.prepareCall(Sprocs.***SELECTUTILISATEUR***);    cst.setInt(1, id);  cst.registerOutParameter(2, java.sql.Types.***VARCHAR***);  cst.registerOutParameter(3, java.sql.Types.***VARCHAR***);  cst.registerOutParameter(4, java.sql.Types.***VARCHAR***);  cst.registerOutParameter(5, java.sql.Types.***VARCHAR***);  cst.registerOutParameter(6, java.sql.Types.***DATE***);  cst.registerOutParameter(7, java.sql.Types.***VARCHAR***);  cst.registerOutParameter(8, java.sql.Types.***VARCHAR***);  cst.executeUpdate();    utilisateurPOJO = **new** UtilisateurPOJO(  id,  cst.getString (2),  cst.getString (3),  cst.getString (4),  cst.getString (5),  cst.getDate (6),  cst.getString (7),  cst.getString (8)  );  } **catch** (SQLException e) {  e.printStackTrace();  } **finally** {  **if** (cst != **null**) {  **try** {  cst.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  **return** utilisateurPOJO;  } |

##### Méthode « getList() »

Pour cette méthode, nous avons opéré de manière différente. Nous avons un seul paramètre dans la procédure stockée, celui-ci est un paramètre sortant, il correspond à un curseur.

|  |
| --- |
| **public** **final** **static** String ***GETLISTUTILISATEUR*** = "{CALL GETLISTUTILISATEUR(?)}"; |

Nous faisons l’appel. Comme vu juste avant, nous avons un paramètre sortant, alors nous utilisons « registerOutParameter() », ici il sera de type « OracleTypes.CURSOR »***.*** Nous l’éxécutons et pour récupérer les informations du curseur, nous le « castons » à un « ResultSet ». Ensuite, tout est pareil qu’avant, nous créons un objet « UtilisateurPOJO » pour chaque enregistrement, et nous les ajoutons dans une liste, que nous retournons.

|  |
| --- |
| **public** ArrayList<UtilisateurPOJO> getList() {  UtilisateurPOJO utilisateurPOJO = **null**;  ArrayList<UtilisateurPOJO> listUtilisateur = **new** ArrayList<UtilisateurPOJO>();  CallableStatement cst = **null**;  ResultSet rs = **null**;  **try** {  cst = connect.prepareCall(Sprocs.***GETLISTUTILISATEUR***);  cst.registerOutParameter(1, OracleTypes.***CURSOR***);  cst.executeUpdate();  // On récupère le curseur et on le cast à ResultSet  rs = (ResultSet) cst.getObject(1);  **while** (rs.next()) {  utilisateurPOJO = **new** UtilisateurPOJO(  rs.getInt ("idUtilisateur"),  rs.getString("pseudo"),  rs.getString("motdepasse"),  rs.getString("nom"),  rs.getString("prenom"),  rs.getDate ("dateNaissance"),  rs.getString("typeUtilisateur"),  rs.getString("mail")  );  listUtilisateur.add(utilisateurPOJO);  }  } **catch** (SQLException e) {  e.printStackTrace();  } **finally** {  **if** (cst != **null**) {  **try** {  cst.close();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  **return** listUtilisateur;  } |

# Screenshots du site web

## Connexion

## Affichage propre à chaque utilisateur

## Vue générale du forum

# Nouveautés apportées

## Bootstrap

Installation de Bootstrap sur notre forum.

C’est le framework HTML, CSS et JS le plus populaire pour développer des projets mobiles réactifs et des projets sur le Web.

# GIT et GitHub

Git est un outil, vue en classe, qui se veut simple et performant. Sa fonction première est de gérer l'évolution du contenu d'une arborescence.

Son fonctionnement repose essentiellement sur cette gestion de fichier : quand un fichier n'est pas modifié, le fichier n'est stocké qu'une seule fois. Par contre, si le fichier est modifié, les deux versions sont stockées sur le disque.

## GIT

Lors de la réalisation de ce projet, l’utilisation de Git nous a été imposée. Néanmoins, nous n’avons pas vu cela comme tel. Le fait d’utiliser Git nous a grandement facilité la tâche. En effet, le fait de pouvoir code tranquillement de son côté et ensuite de n’avoir qu’a « push » notre projet sur le serveur est un gain de temps et d’énergie considérable. Etant donné que nous possédions nos comptes respectifs, nous n’avions pas du tout envie de recréer un compte commun, ce qui serait aller à l’encontre du principe même de Git. Nous nous sommes donc renseignés afin de trouver une alternative. Que chacun puisse accéder au même projet sans aucune restriction.

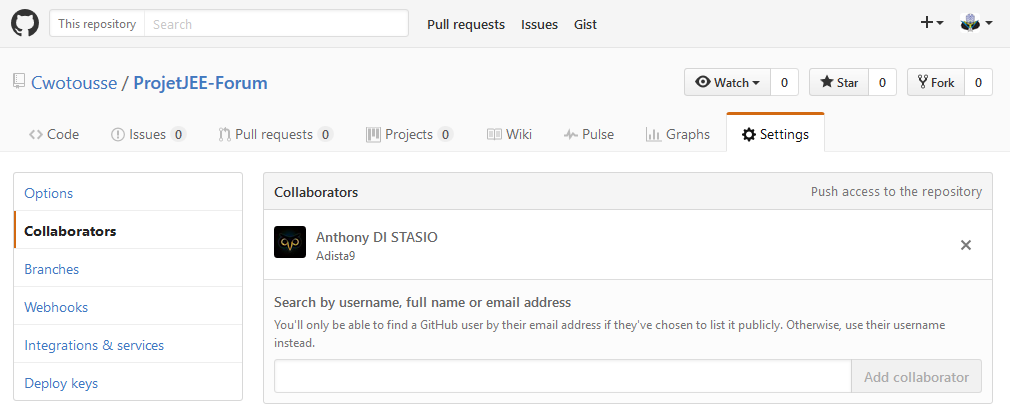


Figure 2 Collaboration GitHub

Pour se faire faire, il suffisait simplement pour la personne hébergeant le projet, d’ajouter l’autre personne en tant que collaborateur. Suite à cela, le « collaborateur » n’a plus qu’à valider l’invitation.

## GitHub

Grâces à l’interface épurée de l’application desktop GitHub, il nous est possible de voir l’historique des « commits » ainsi que son auteur, les fichiers modifiés et les commentaires résumant les modifications.

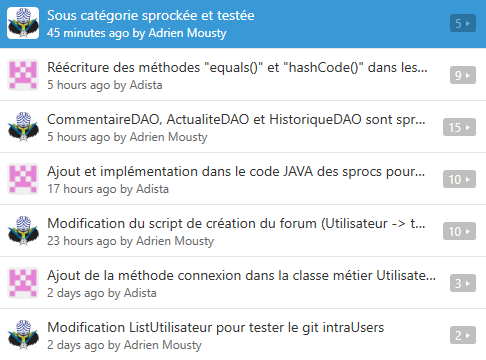


Figure 3 Auteurs GitHub

Une fois avoir synchronisé le projet afin de récupérer la dernière version, il n’y a plus qu’a lancer le projet sur Eclipse et les modifications sont prise en compte.

# Problèmes rencontrés

# Conclusion

# Table des figures

[Figure 1 Hiérarchie 5](#_Toc469332164)

[Figure 2 Collaboration GitHub 17](#_Toc469332165)

[Figure 3 Auteurs GitHub 18](#_Toc469332166)

# Bibliographie

Mkyong. (2011, Avril 10). *JDBC CallableStatement – Stored Procedure CURSOR example*. Consulté le Décembre 13, 2016, sur mkyong: https://www.mkyong.com/jdbc/jdbc-callablestatement-stored-procedure-cursor-example/

Site bootstrap 3 12 16