Anthony Di Stasio, Adrien Mousty

**2016 - 2017**

**Forum actif**

**Projet JEE**

Table des matières

[Enoncé 3](#_Toc471932636)

[Introduction 4](#_Toc471932637)

[Description du site web 5](#_Toc471932638)

[Fonctionnalités du forum 6](#_Toc471932639)

[Les différents diagrammes 7](#_Toc471932640)

[Diagramme des cas d’utilisations 7](#_Toc471932641)

[Diagramme de classes 8](#_Toc471932642)

[Base de données 9](#_Toc471932643)

[Schéma conceptuel 9](#_Toc471932644)

[Connexion oracle 10](#_Toc471932645)

[PL/SQL 11](#_Toc471932646)

[Les tables 11](#_Toc471932647)

[Les clés étrangères 11](#_Toc471932648)

[Les séquences 12](#_Toc471932649)

[Les procédures stockées 12](#_Toc471932650)

[Les exceptions 18](#_Toc471932651)

[Fonction stockée 20](#_Toc471932652)

[Package 21](#_Toc471932653)

[Déclencheurs 23](#_Toc471932654)

[Détails de certaines partie du code 24](#_Toc471932655)

[Java 24](#_Toc471932656)

[Pattern MVC 24](#_Toc471932657)

[Lambda expression 25](#_Toc471932658)

[Procédures stockées 26](#_Toc471932659)

[JSP et servlets 31](#_Toc471932660)

[Exemple de servlet « AfficherListUtilisateurServlet » 31](#_Toc471932661)

[Gestion des exceptions 32](#_Toc471932662)

[Screenshots du site web 33](#_Toc471932663)

[Connexion 33](#_Toc471932664)

[Profil 33](#_Toc471932665)

[Vue générale du forum 34](#_Toc471932666)

[Sous-catégorie 34](#_Toc471932667)

[Sujet 35](#_Toc471932668)

[Erreur 35](#_Toc471932669)

[Commentaire 36](#_Toc471932670)

[Historique de connexion 37](#_Toc471932671)

[Panneau d’administration 37](#_Toc471932672)

[Navigateurs 38](#_Toc471932673)

[Firefox 38](#_Toc471932674)

[Google Chrome 39](#_Toc471932675)

[Nouveautés apportées 40](#_Toc471932676)

[Bootstrap 40](#_Toc471932677)

[Description 40](#_Toc471932678)

[Utilisation 40](#_Toc471932679)

[jQuery 42](#_Toc471932680)

[Description 42](#_Toc471932681)

[Utilisation 43](#_Toc471932682)

[JSTL 44](#_Toc471932683)

[Description 44](#_Toc471932684)

[Liste des tags utilisés 44](#_Toc471932685)

[Utilisation 45](#_Toc471932686)

[GIT et GitHub 47](#_Toc471932687)

[GIT 47](#_Toc471932688)

[GitHub 48](#_Toc471932689)

[Problèmes rencontrés 49](#_Toc471932690)

[Conclusion 52](#_Toc471932691)

[Table des figures 53](#_Toc471932692)

[Bibliographie 55](#_Toc471932693)

# 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 2017 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.

Pour la partie SGBD, on vous demande :

1. Le schéma conceptuel de la base de données
2. Le script de création de la base de données
3. De gérer les accès à la base (interrogation et mise à jour) en PL/SQL :
   1. Utilisation des procédures, fonctions stockées (5)
   2. Utilisation des packages (1)
   3. Utilisation des triggers (1)
4. De gérer les exceptions
5. D’utiliser les curseurs, des variables de type record et des tableaux (1)
6. De documenter les différents scripts d’accès à la base de données

# Introduction

Le projet qui va suivre nous a particulièrement intéressés. En effet, contrairement au projet en Java, celui-ci se fait en groupe. C’est une excellente façon de confronter nos idées et de partager nos connaissances. Car bien que nous ayons suivi le même cursus, nous avons des façons différentes de coder.

Contrairement au projet traitant d’un agenda développé principalement pour les assistantes sociales et leurs patients, celui-ci sera codé en JEE/JAVA tout en ayant une base de données Oracle distante. C’est une bonne chose, car de cette façon nous avons la certitude que, qu’importe où nous nous trouvons, nous aurons toujours les mêmes données dans notre base de données.

Nous sommes partis dans l’idée d’implémenter un forum. Premièrement, car c’est quelque chose qui nous parle. Deuxièmement, car il y a peu de sujets différents à traiter. Un site de jeu, un réseau social ou un forum. Nous avons donc fait notre choix.

Avant de coder et même avant de commencer à élaborer nos schémas, nous nous sommes informés sur plusieurs choses ayant attrait au forum. Par exemple, comment les droits sont-ils gérés, qu’elle est la hiérarchie mise en place. Quelle est la différence entre un utilisateur connecté et non connecté. Est-ce que ces deux personnes ont accès aux mêmes informations ? Tout un tas de choses qui semblent logiques, voire basiques, mais qui nécessitent tout de même qu’on y réfléchisse.

Nous voulons commencer notre programme en ayant des fondations solides, afin de ne pas devoir modifier notre schéma ultérieurement dû à un élément mal pensé à la naissance de notre projet.

# 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 décrits 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 pourront la promouvoir.

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

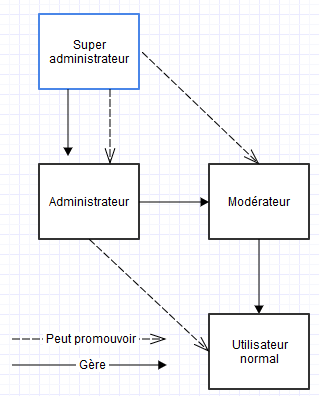
**

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
* 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
  + Voir l’historique de connexion

# Les différents diagrammes

## Diagramme des cas d’utilisations

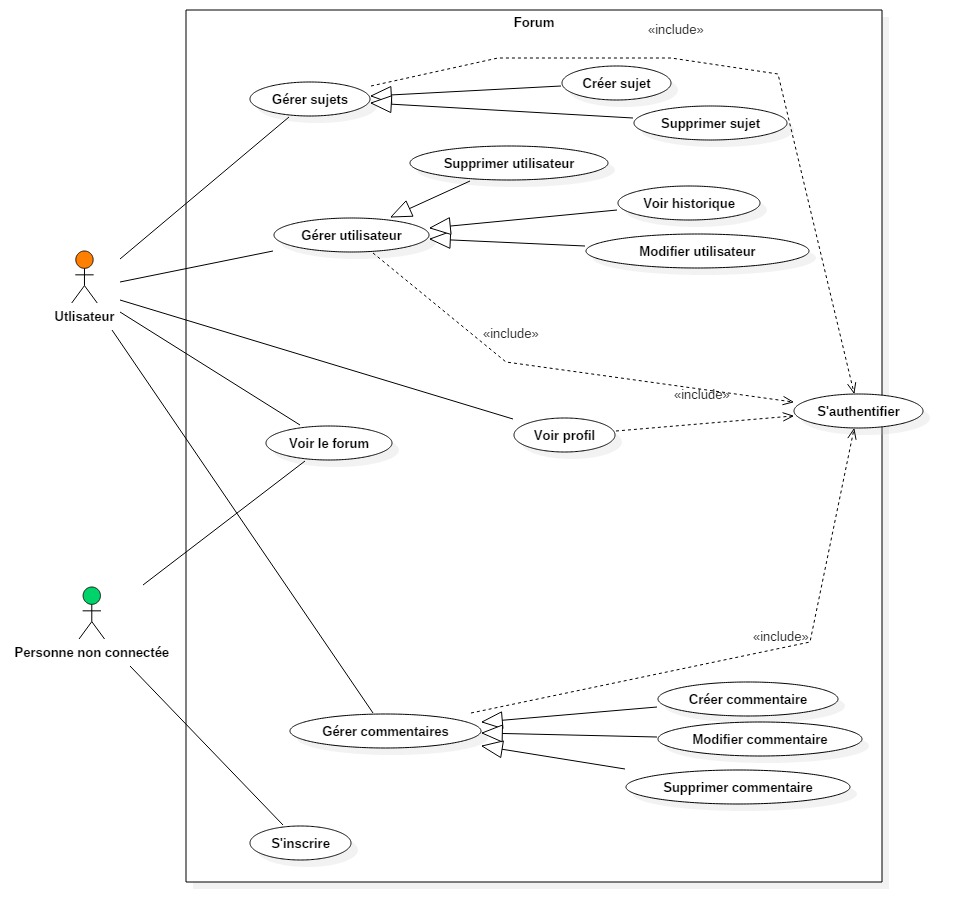


Figure 2 Use Case

Nous avons un acteur qui n’apparaît nulle part ailleurs, la personne non connectée. En effet, il est difficile de représenter une personne qui n’existe pas dans la base de données.

Néanmoins, cette personne peut interagir avec le forum. Elle peut s’inscrire et voir les différents sujets.

Ce diagramme nous montre les actions possibles (et à fortiori à implémenter) d’un utilisateur.

## Diagramme de classes

Le diagramme de classe permet de voir toutes les classes, leurs attributs ainsi que leurs méthodes existantes dans le programme. Nous pouvons aussi voir les différents liens entre les classes.

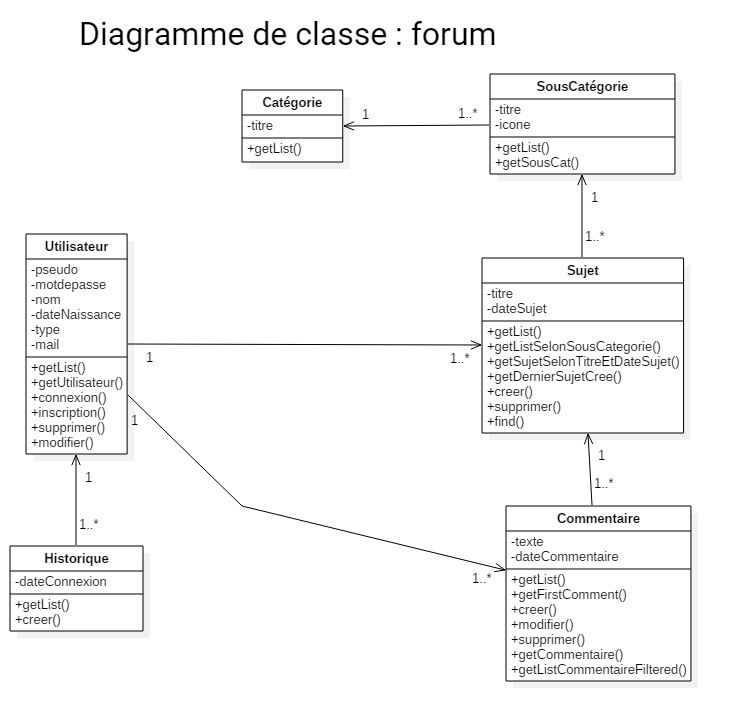


Figure 3 Diagramme de classe

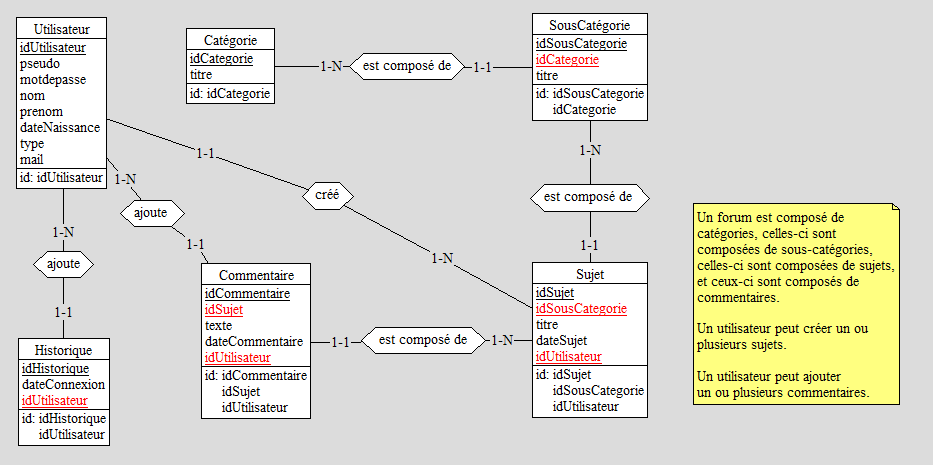
Le diagramme de classe est sensiblement équivalent au schéma conceptuel, seulement les clés primaires et étrangères sont absentes. Une autre différence est l’inversion des cardinalités.

# Base de données

## Schéma conceptuel

Le schéma conceptuel représente la base de données de notre application. Celui-ci les tables nécessaires, ainsi que les différents liens entre-elles.

Figure 4 Schéma de la base de données



## Connexion oracle

Nous avons utilisé une base de données distante et non locale pour cet exercice. Mme Copin nous a donné nos logs et il suffisait de créer une nouvelle connexion avec ceux-ci. Nous la testons ensuite, le statut affiche « succès ». Il ne reste plus qu’à l’enregistrer.

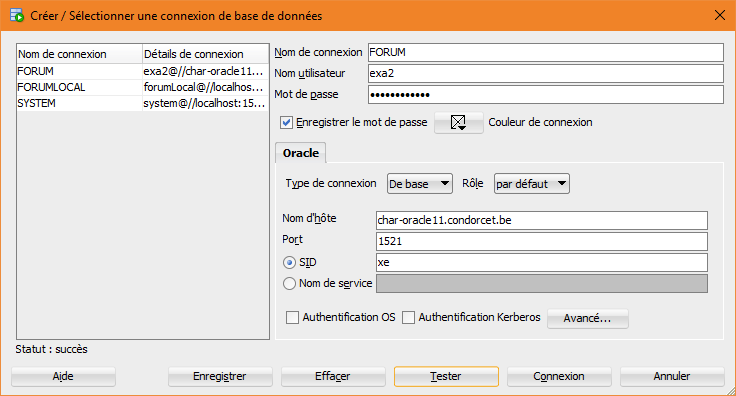


Figure 5 Connexion Oracle

Comme nous sommes des gens prévoyants, nous avons déjà fait un script afin de créer nos tables et toutes les autres choses dont nous avions besoin.

Pour finir, nous avons dû appeler cette base de données de notre programme. Cela se fait dans la classe « SingletonConnexion ». Cela s’apparente à ce que nous avions fait précédemment pour le projet en Java.

Pour utiliser Oracle comme serveur, il faut bien sur ajouter une bibliothèque à notre projet. Celle-ci se nomme « ojdbc6.jar ».

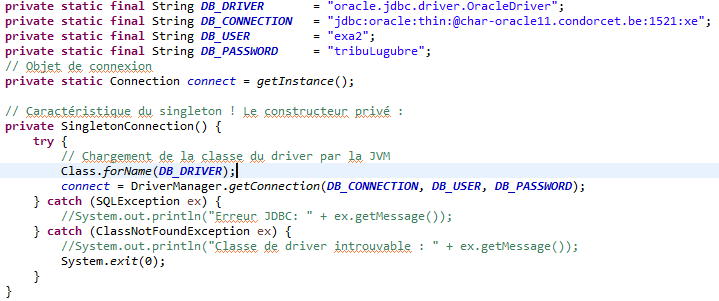


Figure 6 Connexion Oracle – Java

Dans « DriverManger.getConnection », nous entrons toutes les données qui étaient nécessaires à la création de la connexion. Il n’y a rien d’autre à faire de plus, la base de données est liée à notre forum.

## PL/SQL

### Les tables

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

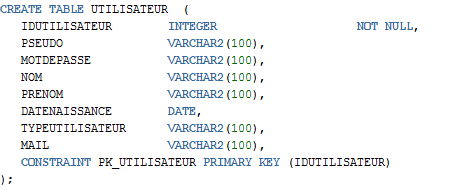


Figure 7 table Utilisateur

Le reste des tables conservent cette même syntaxe.

### 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.

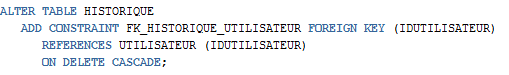


Figure 8 Foreign key Historique Utilisateur

### 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.

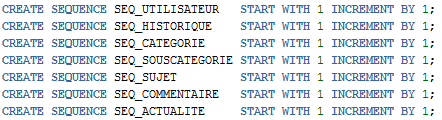


Figure 9 Les séquences

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 similaires, 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 :

* Le pseudo, le mot de passe, le nom, le prénom, la date de naissance, le type et le 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.

Nous avons aussi géré les exceptions, celles-ci sont expliquées juste après les procédures stockées.



Figure 10 Insert utilisateur

#### 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.

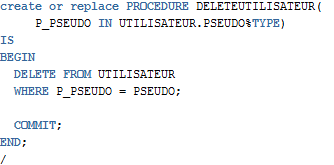


Figure 11 Delete utilisateur

#### 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.

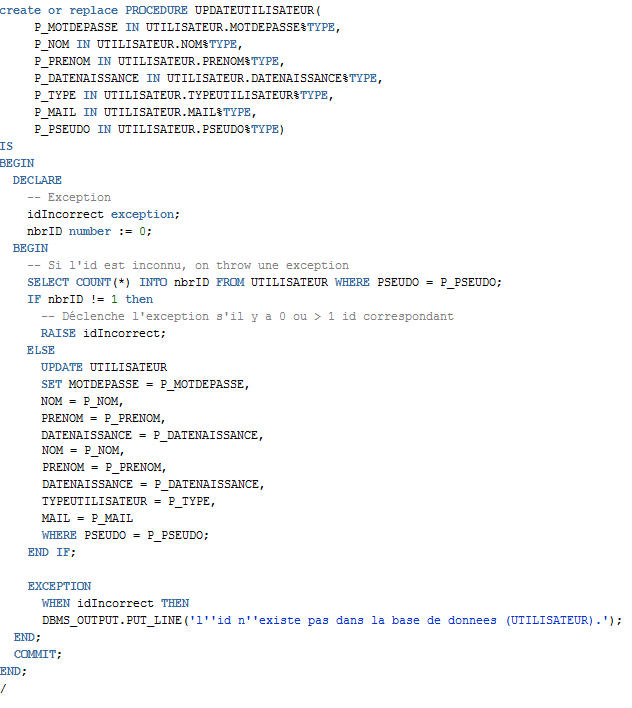


Figure 12 Update utilisateur

#### 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.

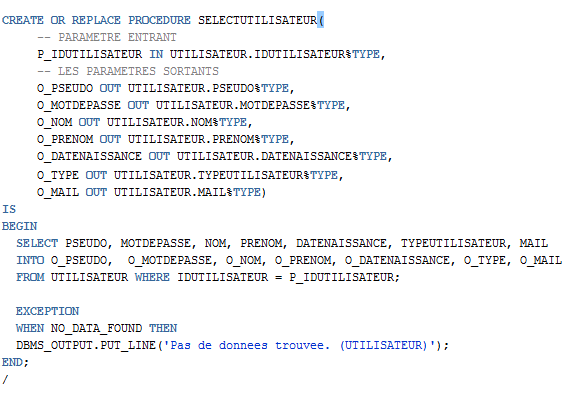


Figure 13 Select Utilisateur

#### 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 ».

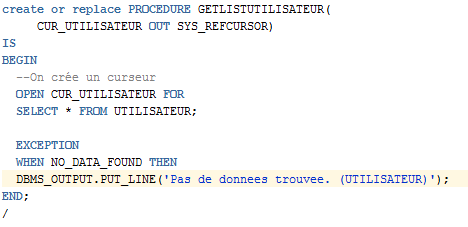


Figure 14 Get List Utilisateur

### Les exceptions

Dans ce projet nous avons utilisé les deux types d’exceptions possibles, à savoir : les **anonymes** et les **nommées**. D’ailleurs, la grande majorité de nos procédures stockées comprennent des exceptions. Seules les procédures traitant de la suppression n’en sont pas pourvues. Nous avons jugé que ce type de procédure ne nécessitait pas d’exception éventuelle.

#### Exception nommée

Ce type d’exception possède deux avantages certains :

* On crée une exception ayant un nom.
* On peut la réutiliser dans le code
* Le code est accessible et modifiable (il se trouve dans SQLErr)
* Il y a déjà une longue liste d’exception potentiellement existante.

Nous l’avons utilisée pour les procédures SELECT et GETLIST.

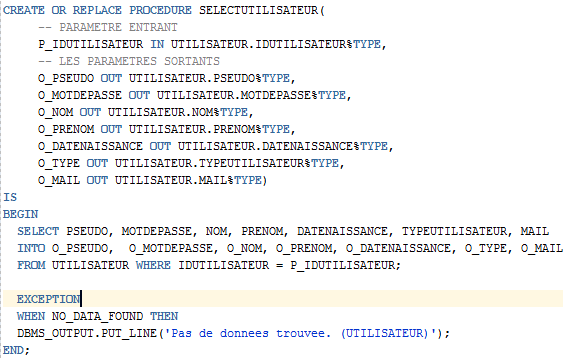


Figure 15 NO\_DATA\_FOUND

Pour ce genre d’exception, il ne faut pas créer de nouveau bloc PL SQL. C’est aussi un type d’exception moins « verbeux ».

Lors de la sélection, si aucune donnée ne peut être renvoyée, l’exception « NO\_DATA\_FOUND » (code erreur = +100) est déclenchée. Car ce genre d’exception se déclenche à la suite d’une erreur PL/SQL. Cela nous affiche qu’aucune donnée n’a pu être trouvée.

#### Exception anonyme

Ce genre d’exception est défini et déclenché par le développeur. Leur code est compris entre -20000 et -20999. Malheureusement, l’intérêt est restreint, car sans nom on ne peut pas les utiliser.

Il faut donc les créer, les nommer et es appeler explicitement et les lever à un certain moment du programme. Ces exceptions ont une portée qui leur est propre : elles quittent le bloc dans lequel elles sont pour aller rechercher l’exception dans un bloc supérieur. Elles s’arrêtent lorsqu’il trouve le mot « exception ».

Nous les avons utilisées pour les procédures d’INSERT.

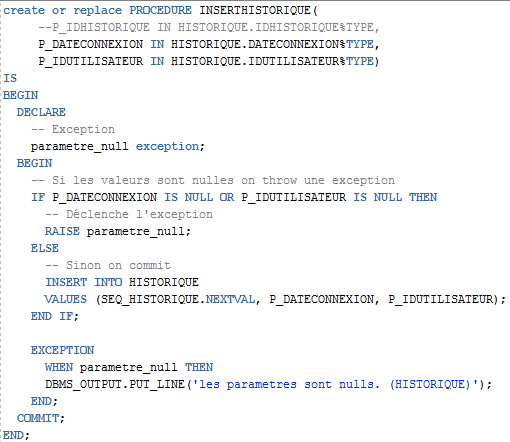


Figure 16 Exception anonyme

Avant toute chose, nous devons créer un nouveau bloc PL SQL, puis déclarer et nommer notre exception. Ensuite, nous voulons savoir si les valeurs à insérer ne sont pas initialisées. Si c’est le cas, alors nous levons l’exception. Celle-ci affichera un texte en rapport avec le type d’exception en question et l’ajout dans la table ne sera pas effectué.

### Fonction stockée

#### Description

Nous avons créé quelques fonctions stockées, car nous jugions qu’elles pouvaient être utilisé dans futur éventuel. En effet, elles remplissent toutes un rôle qui va dans la même direction. À savoir récupérer des données sur le type d’utilisateurs ou alors sur les sujets et messages créés.

Pour ce qui est de leur création ou de l’endroit où elles se trouvent, nous n’allons pas plus nous étendre sur le sujet, car cela a déjà été abordé précédemment avec les procédures stockées.

#### Exemple

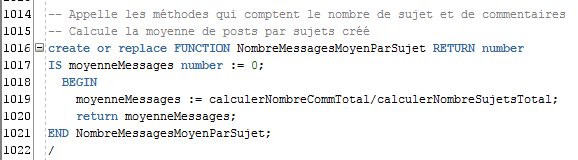


Figure 17 exemple fonction stockée (1)

Par exemple, avec cette fonction stockée nous appelons deux autres fonctions. Ces deux fonctions ressortent respectivement le nombre de commentaires et au total et le nombre de sujets. Nous divisons ensuite ces deux valeurs et la fonction stockée « NombreMessagesMoyenParSujet » nous donnera la moyenne des messages laissés sur un sujet. Cela pourrait être utile lorsque nous voulons analyser par exemple le taux de réaction à chaque sujet et savoir si un sujet fait un « flop » ou non.

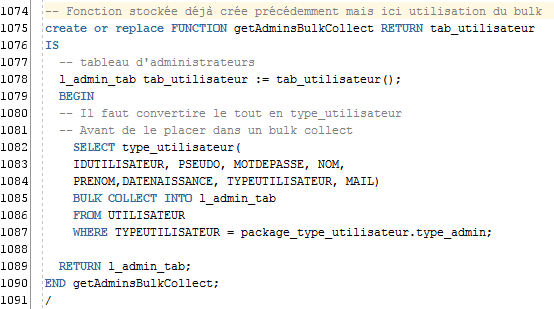


Figure 18 Exemple fonction stockée (2)

Dans cet exemple un peu plus complexe, nous cherchons simplement à créer un tableau de type « UTILISATEUR » ne contenant que les personnes ayant le type « Admin ». Pour se fait, nous avons stockés le type correspond dans notre base de données. Nous utilisons aussi un package pour récupérer le type à filtrer.

### Package

#### Description

Un package est un objet de schéma qui regroupe les types, les éléments et les sous-programmes PL / SQL liés logiquement. Les « packages » ont généralement deux parties, une déclaration et un corps. La déclaration agit comme l'interface de l’application ; elle déclare les types, les variables, les constantes, les exceptions, les curseurs et les sous-programmes disponibles pour l'utilisation.

Le corps, lui, définit entièrement les curseurs et les sous-programmes, et implémente ainsi la déclaration.

#### Exemple

Ce package contient deux fonctions, et une procédure, la première fonction permet d’afficher les utilisateurs ayant plus de 10 commentaires. La deuxième permet d’afficher les sujets les plus commentés (plus de 10 commentaires).

La procédure, quant à elle, permet de donner un accès plus élevé que le rôle d’utilisateur, et donc devenir modérateur. Elle opère lorsqu’un utilisateur dépasse 100 connexions sur le forum.



Figure 19 Package 1

### Déclencheurs

#### Description

Un déclencheur est une procédure stockée spéciale qui s'exécute automatiquement lorsqu'un événement se produit dans le serveur de base de données. Les déclencheurs DML peuvent s'exécuter lorsqu'un utilisateur essaie de modifier des données via un événement de type DML[[1]](#footnote-1). Les événements DML sont des instructions INSERT, UPDATE ou DELETE. Le déclencheur peut s’activer AVANT ou APRES l’événement. Cela se configure dans son code. Ceux-ci n’ont pas d’argument.

Attention néanmoins, ils sont déconseillés si nous pouvons les éviter, car ils ralentissent considérablement la base de données déjà potentiellement surchargée.

Il faut aussi prendre en compte le « commit » ou le « rollback » est strictement interdit dans un déclencheur.

#### Avantages

1. Ajouter des contraintes sur les valeurs d'une colonne d'une table non gérées par le SGBDR.
2. Réaliser un audit des changements sur les données (ex : lister l’ensemble des évènements qui vont intervenir sur une table)
3. Ajout des règles de gestion

* Ne pas inscrire une personne ayant une adresse e-mail bannie

1. Publier des informations concernant divers évènements (postgreSql)

#### Exemple

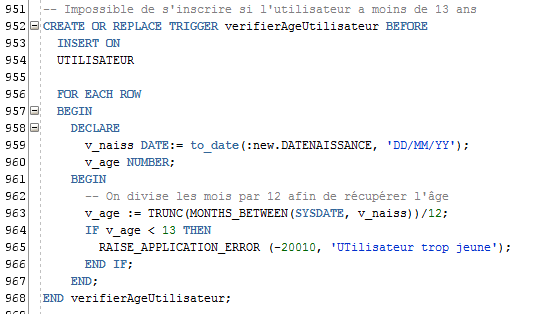


Figure 20 Exemple déclencheur

Dans cet exemple, nous calculons l’âge de l’utilisateur voulant s’inscrire. S’il a moins de 13 ans, alors celui-ci déclenche une exception et ne pourra pas s’inscrire. Nous ne voulons pas d’enfants trop jeunes sur notre forum, car cela pourrait heurter leur sensibilité.

# Détails de certaines partie du code

## Java

### Pattern MVC

Nous avons implémenté le pattern MVC dans notre forum dans le but de mieux structurer notre code. Celui-ci découpe notre code en 3 parties : modèle, vue et contrôleur.

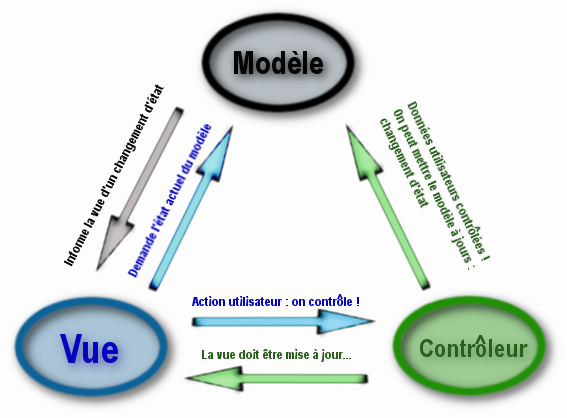


Figure 21 Pattern MVC

Dans notre projet, nous avons donc séparés, les modèles, les vues, et les contrôleurs.

Nos vues correspondent aux différentes pages « .jsp ». Nos modèles correspondent à des classes possédant les méthodes effectuant des méthodes DAO. Et enfin, nos contrôleurs qui sont les servlets.

### Lambda expression

Comme pour l’ancien projet JAVA, nous avons continué d’utiliser des « filters » sur nos listes. Voici quelques exemples :

Ce filter effectue une vérification dans la base de données sur la liste des utilisateurs, il vérifie si aucun pseudo ou mail n’existerait pas déjà dans cette base de données.

Si ceux-ci n’existent pas alors je peux créer l’utilisateur.

|  |
| --- |
| **if**(**this**.getList()  .stream()  .anyMatch(x -> x.getPseudo().equals(pseudo)  || x.getMail().equals(mail))) {  **return** **false**;  } **else** {  **new** DAOFactory().getUtilisateurDAO().create(utilisateur);  **return** **true**;  } |

Voici un autre exemple. Celui-ci retourne une liste de sous-catégorie correspondante à une catégorie.

Nous récupérons l’objet « Categorie » grâce au lambda x qui correspond à une sous-catégorie, et nous vérifions son titre par rapport à celui reçu en paramètre de la méthode

|  |
| --- |
| **public** ArrayList<SousCategorie> getList(String titreCat){  **return** **this**.getList()  .stream()  .filter(x -> x.getCategorie().getTitre().equals(titreCat))  .collect(Collectors.*toCollection*(ArrayList::**new**));  } |

### Procédures stockées

#### Dans le DAO

Pour bien comprendre comment nous avons implémenté 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 méthodes, la seule différence dépend du nombre de paramètres de la procédure stockée.

##### Méthode « create() »

Pour implémenter les procédures stockées que nous avons créées 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’exécutons 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;  } |

## JSP et servlets

### Exemple de servlet « AfficherListUtilisateurServlet »

Pour illustrer une servlet, nous avons décidé de vous montrer la servlet « AfficherListUtilisateurServlet ».

Tout d’abord, nous récupérons la liste des utilisateurs grâce au modèle utilisateur. Ensuite, nous vérifions si celle-ci n’est pas vide, si oui alors nous renvoyons une erreur sur notre page prévue pour cela. Et nous redirigeons vers cette page d’erreur.

Sinon, nous vérifions la session, si elle existe déjà, alors nous pouvons vérifier si le type d’utilisateur est bien administrateur, car cette liste d’utilisateur n’est visible que pour l’administrateur. Si le type est bien celui voulu, alors nous pouvons « setAttribute » cette liste, et rediriger vers la page « restrained\_access.jsp ». Si la condition est fausse, nous renvoyons de nouveau vers la page d’erreur avec le message adéquat, etc.

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  UtilisateurModele utilisateurModele = **new** UtilisateurModele();  ArrayList<Utilisateur> listUtilisateur = utilisateurModele.getList();  **if** (listUtilisateur.isEmpty()) {  request.setAttribute("error\_message", "Pas d'utilisateurs enregistré.");  RequestDispatcher dispatcher = request.getRequestDispatcher("/VUE/erreur.jsp");  dispatcher.forward(request, response);  response.setContentType("text/html");  } **else** {  HttpSession session = request.getSession();  **if** (!session.isNew()) {  Utilisateur utilisateurConnecté = (Utilisateur) session.getAttribute("utilisateur");  **if** (utilisateurConnecté.getType().equals("Admin")) {  request.setAttribute("listUtilisateur", listUtilisateur);  RequestDispatcher dispatcher = request  .getRequestDispatcher(request.getContextPath() + "/restrained\_access.jsp");  dispatcher.forward(request, response);  } **else** {  request.setAttribute("error\_message", "Vous n'êtes pas autorisé à visionner cette page.");  RequestDispatcher dispatcher = request.getRequestDispatcher("/VUE/erreur.jsp");  dispatcher.forward(request, response);  response.setContentType("text/html");  }  } **else** {  request.setAttribute("error\_message", "Pas de session en cours.");  RequestDispatcher dispatcher = request.getRequestDispatcher("/VUE/erreur.jsp");  dispatcher.forward(request, response);  response.setContentType("text/html");  }  }  response.setContentType("text/html");  } |

### Gestion des exceptions

À chaque fois qu’une exception levée par le programme ou une erreur levée par nous-mêmes est déclenchée, nous devons afficher un écran qui est spécialement dédié aux erreurs.

Pour se faire, cela est relativement simple.

Premièrement, il faut modifier le XML, indiquer la page que nous voulons définir comme celle recevant les exceptions en tout genre.

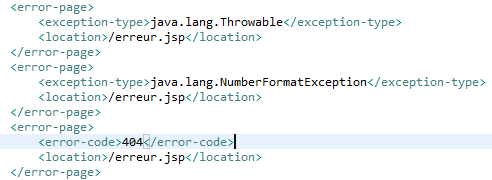


Figure 22 Gestion erreur XML

Nous pouvons voir que les erreurs de type 404 et deux exceptions provenant du code Java sont gérées. Nous pouvons en gérer bien entendu bien plus.

Ensuite, dans notre page dédiée nous lui indiquons que c’est bien elle qui se chargera de tout.

2016-12-30_19h51_25

Ensuite, il n’y a plus qu’a rediriger les erreurs sur cette page lorsque nous les levons nous-mêmes. Dans le cas de l’exception 404 il n’y a rien d’autre à faire si ce n’est de faire un écran décent expliquant l’exception à l’utilisateur et en lui proposant une redirection.

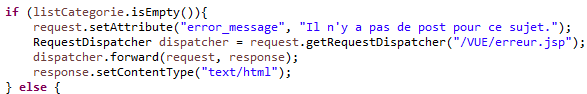


Figure 23 Redirection erreur

Dans ce servlet, nous tentons de récupérer la liste de toutes les catégories. Si cette liste est vide, nous ne pouvons rien afficher. Alors plutôt que de ne rien afficher, nous redirigeons l’utilisateur vers la page d’erreur. Celui-ci sera donc averti qu’il n’existe pas de sous-catégorie.

# Screenshots du site web

## Connexion

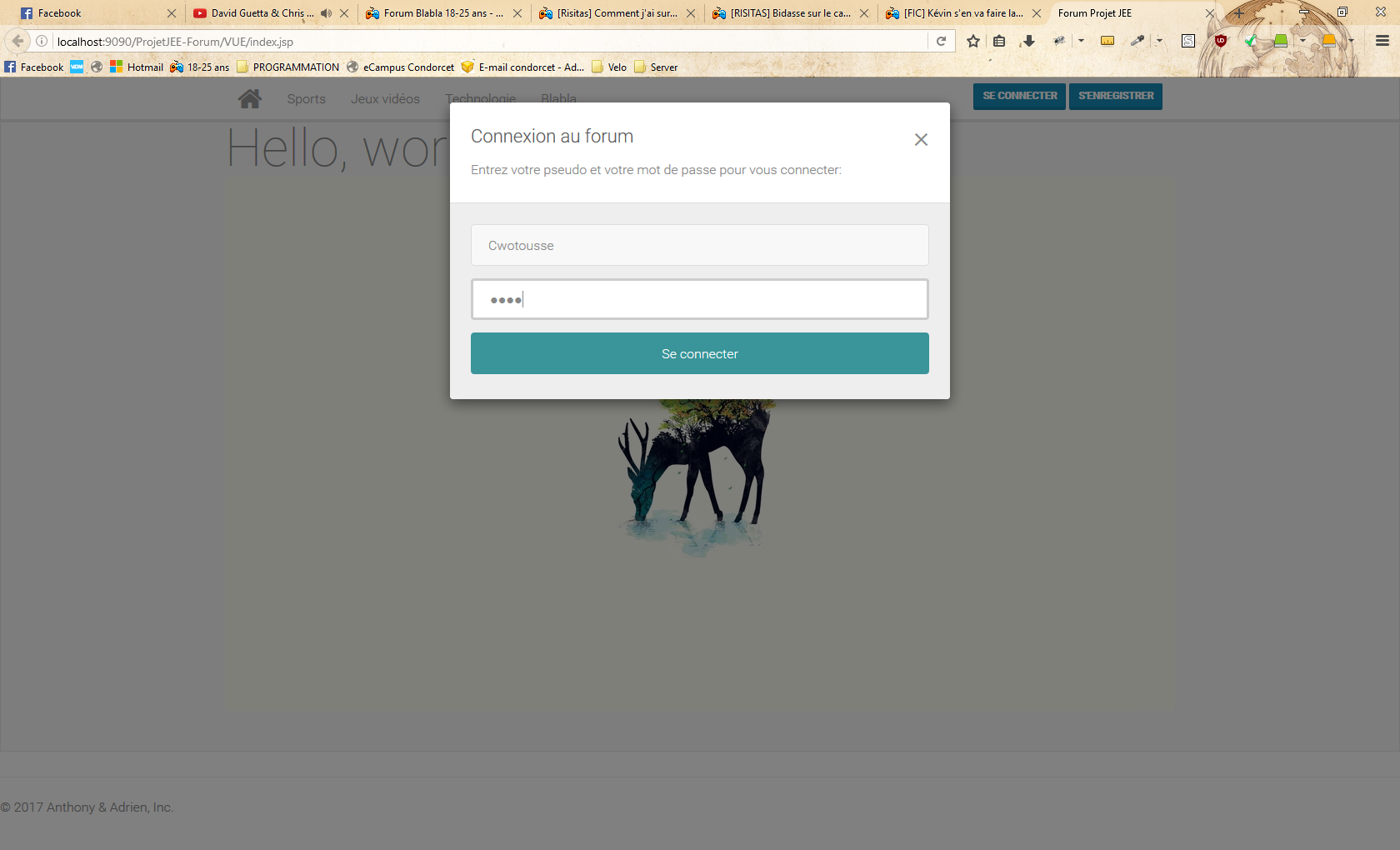


Figure 24 Ecran : connexion

## Profil

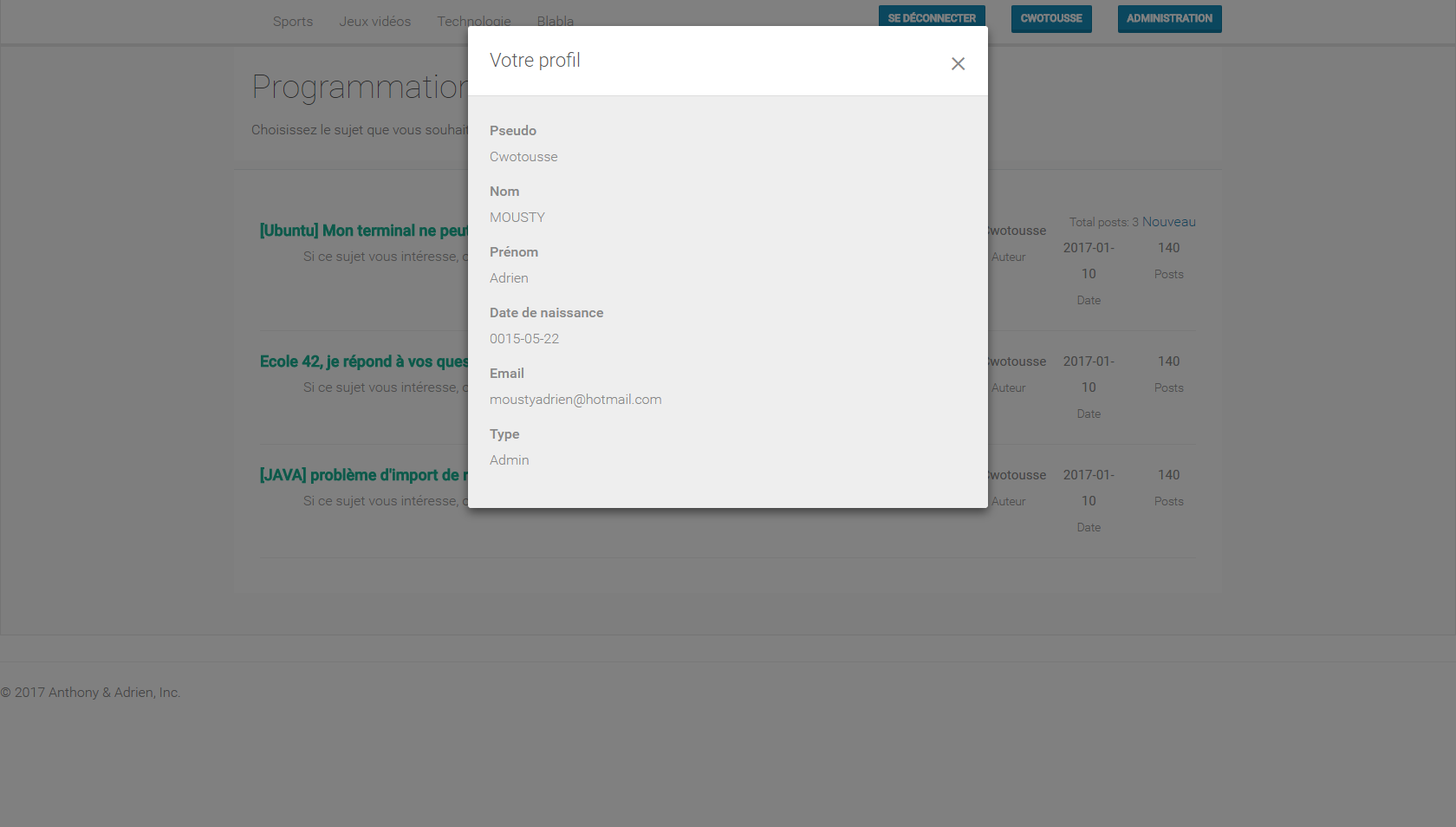


Figure 25 Ecran : profil

## Vue générale du forum

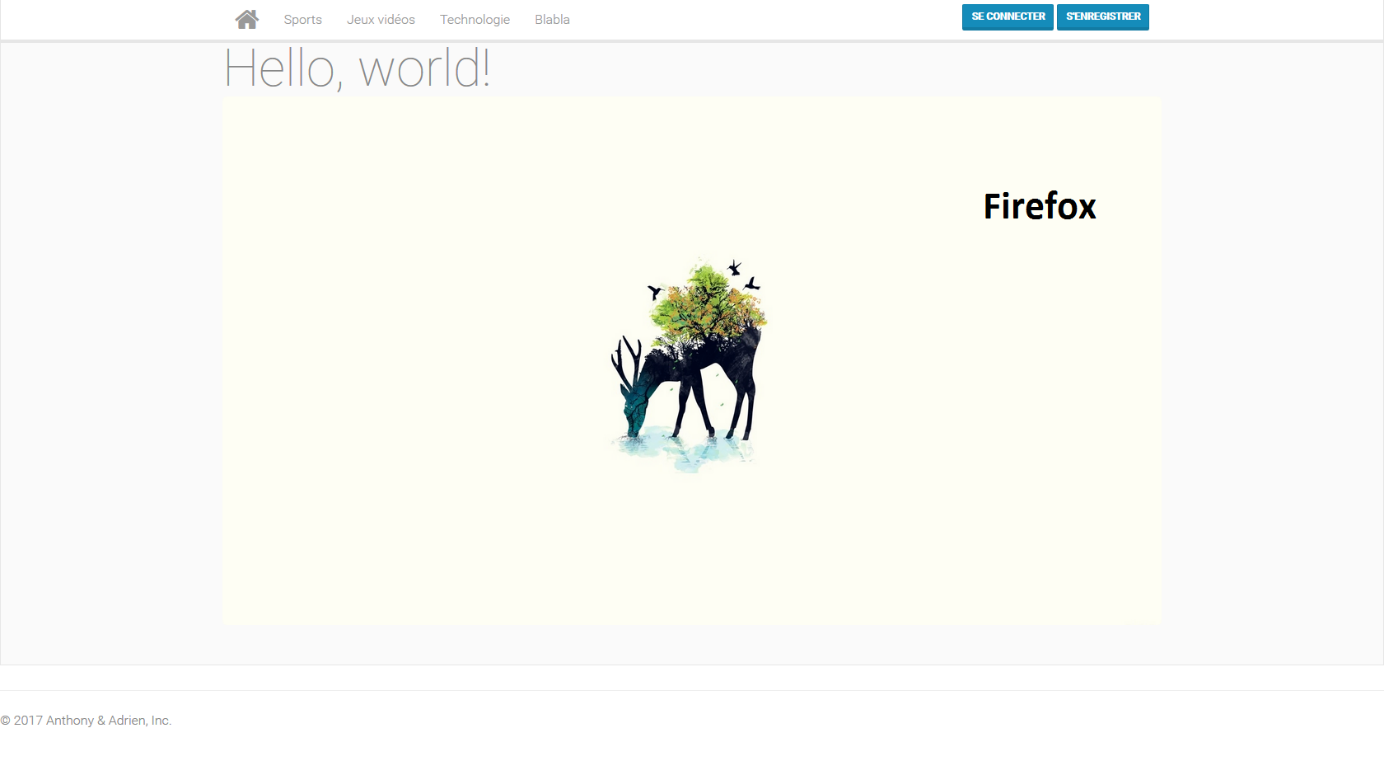


Figure 26 Ecran : home

C’est le premier écran que l’utilisateur peut voir lorsqu’il arrive sur notre site. Il y a relativement peu d’informations affichées. Les catégories ainsi que les boutons pour se connecter ou s‘inscrire.

## Sous-catégorie

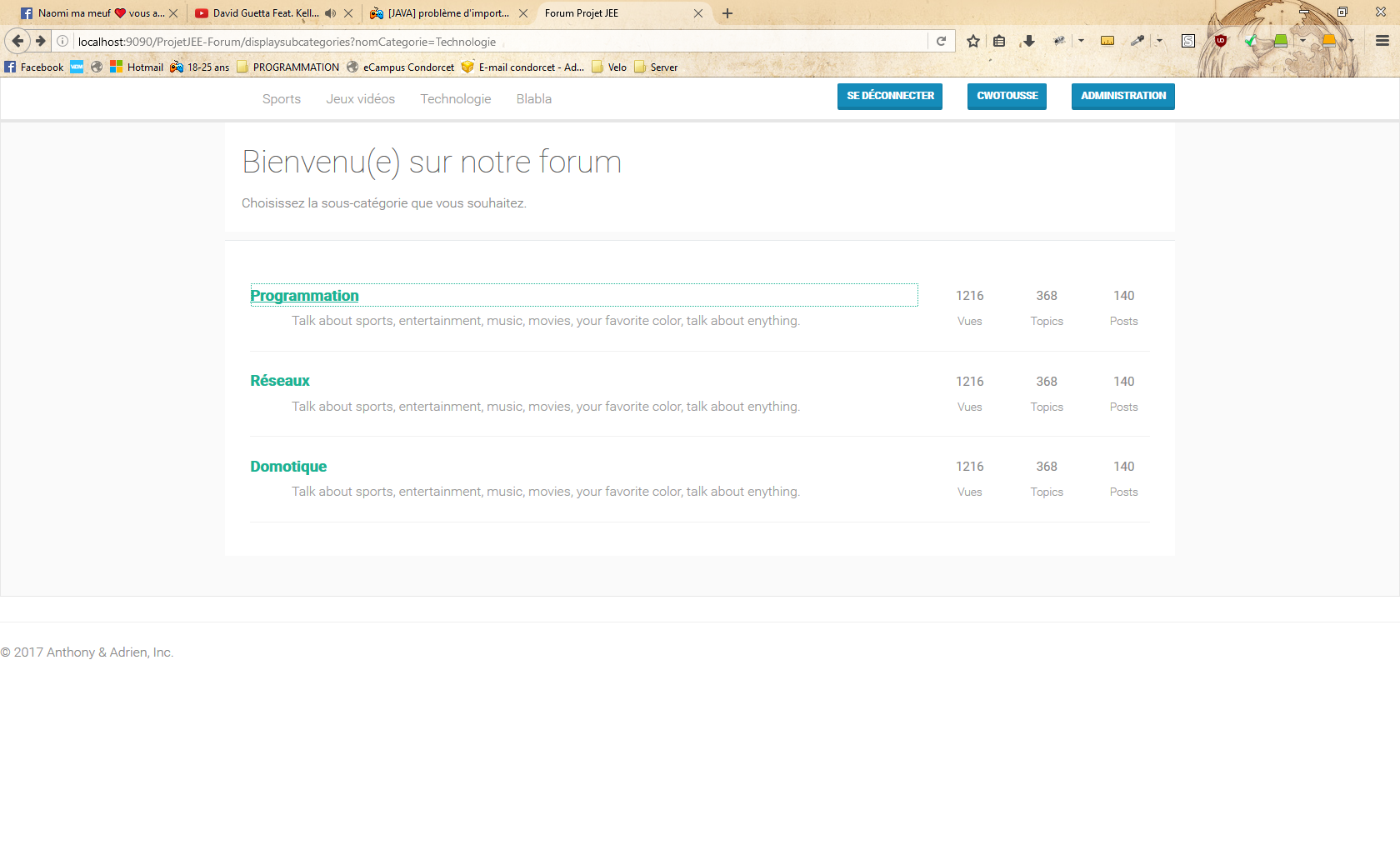


Figure 27 Ecran : sous-catégorie

Cet écran apparaît lorsque l’on clique sur un onglet de la « navbar ». Il nous redirige vers les sous-catégories correspondantes. Pour cet exemple, nous avons cliqué sur l’onglet « technologies ».

## Sujet

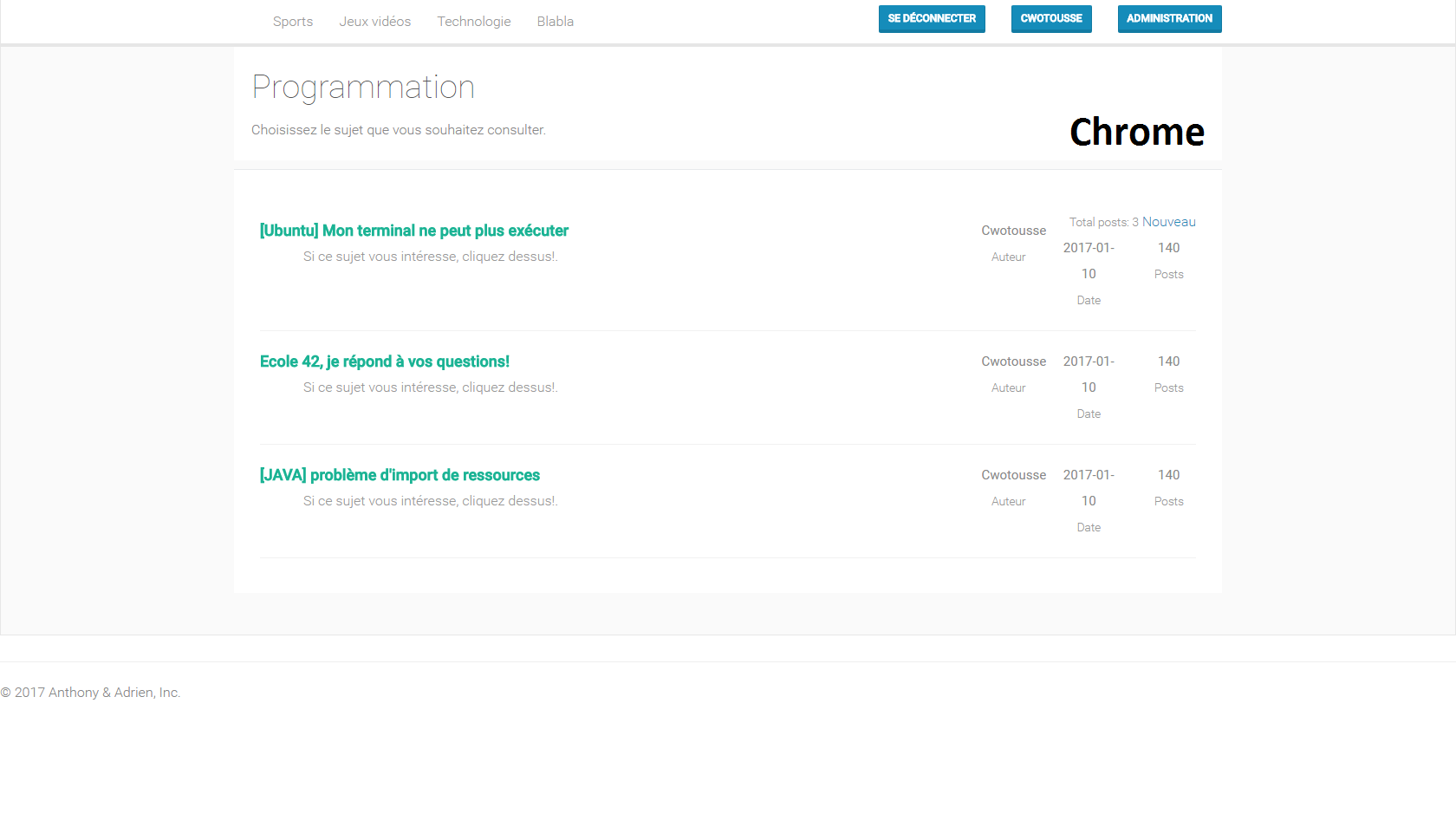


Figure 28 Ecran : sujets

Cet écran est la suite logique de Figure 18 Ecran : sous-catégorie. En effet, lorsque l’on clique sur une sous-catégorie tous les sujets s’y référant sont affichés. Ici nous avons clique sur « programmation ». Il s’avère qu’il y a déjà 3 sujets créés à propos de la programmation.

## Erreur



Figure 29 Ecran : erreur

Cet écran apparaît lorsqu’une erreur survient, comme il l’indique si bien. Il offre aussi la possibilité de connaître la raison de l’erreur. Il permet en outre de retourner au menu principal. Toutes les erreurs passent par lui.

## Commentaire

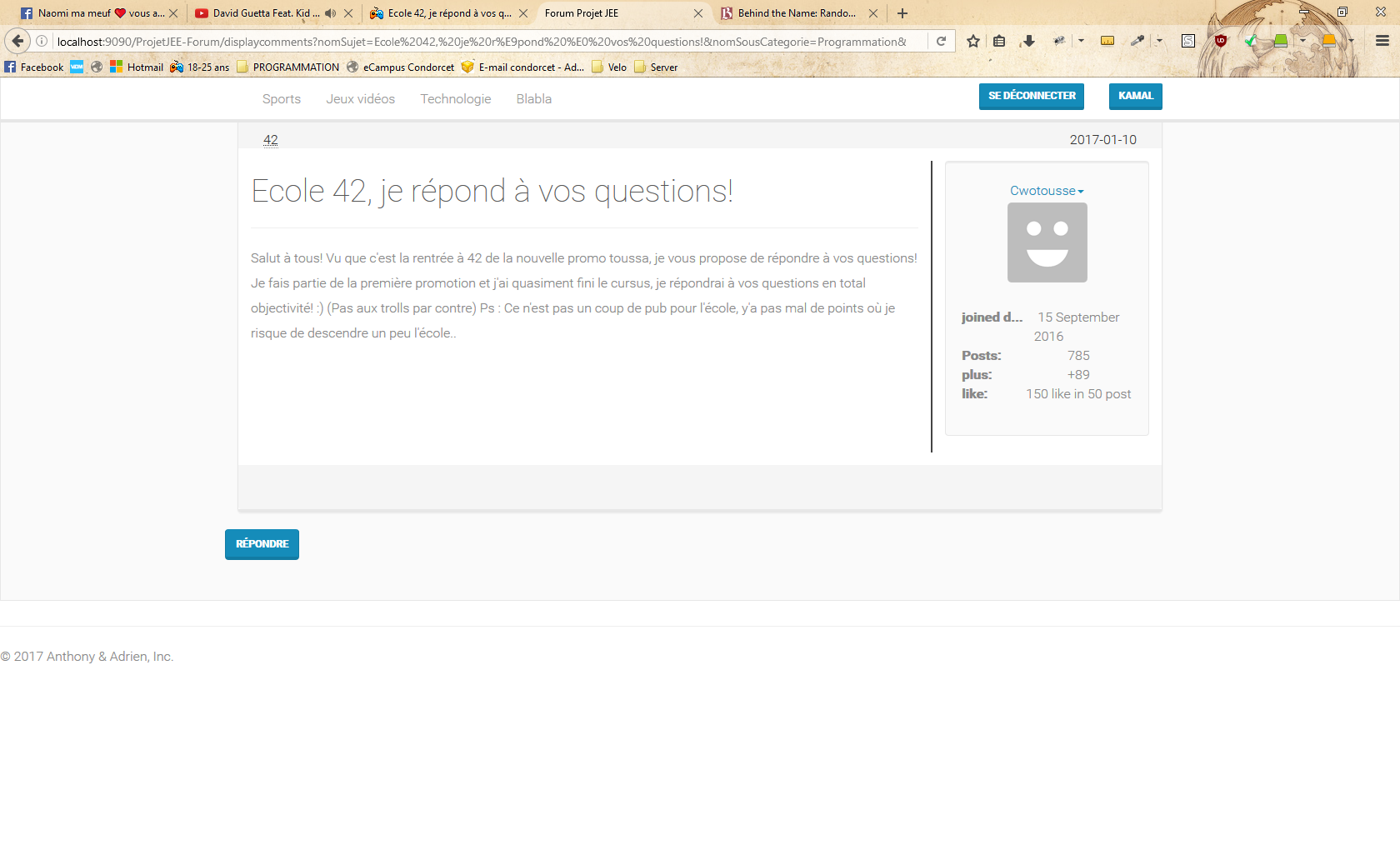


Figure 30 Ecran : affichage commentaire

Lorsque nous cliquons sur un sujet, les commentaires relatifs à celui-ci sont affichés.

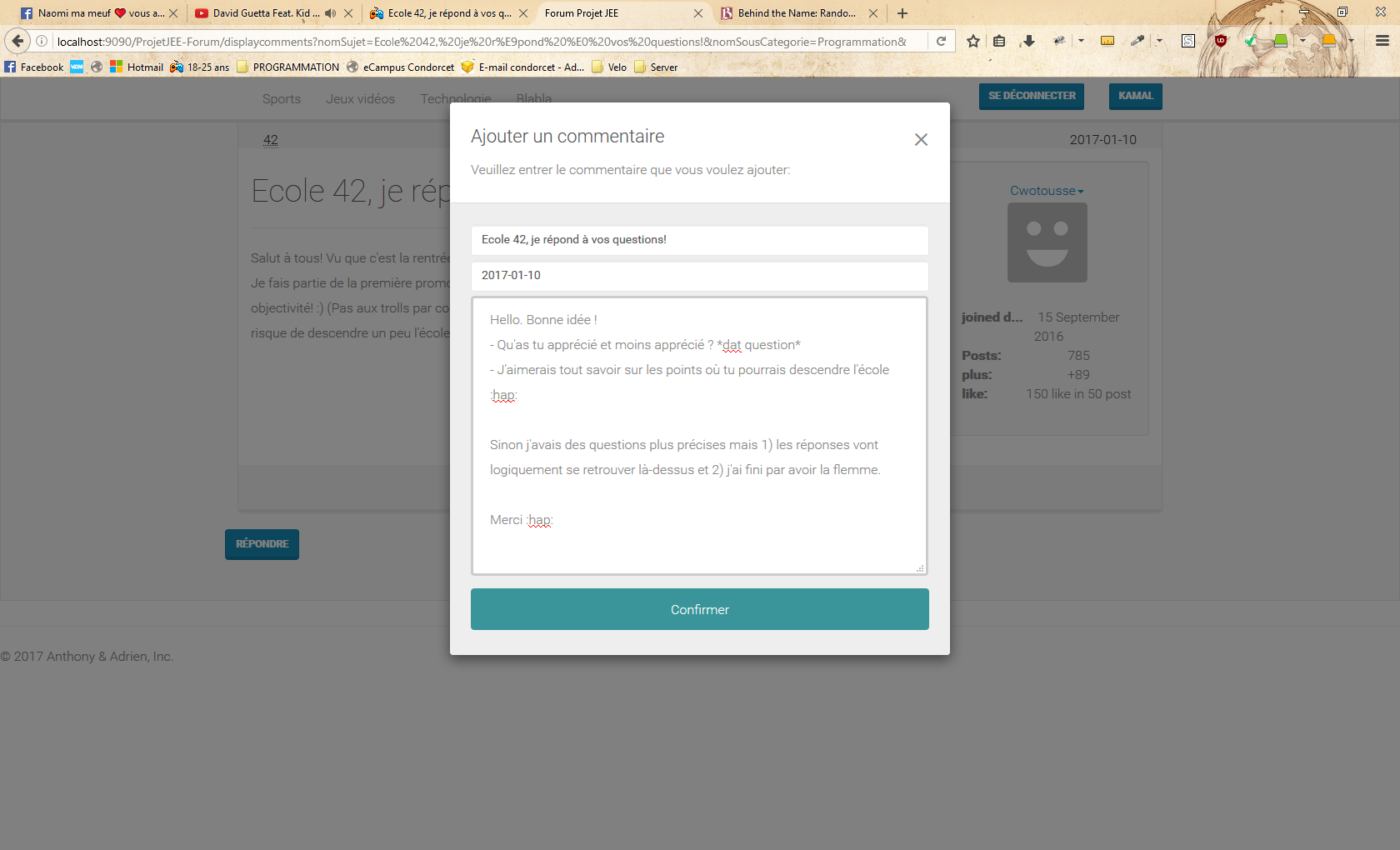


Figure 31 Ecran : poster un commentaire

Une fenêtre modale apparaît dynamiquement sur l’écran lorsqu’une personne souhaite réagir à un sujet. Le titre et la date du sujet sont bien entendu en « read-only ». Le commentaire sera ajouté à la suite des autres déjà existants pour ce sujet.

## Historique de connexion

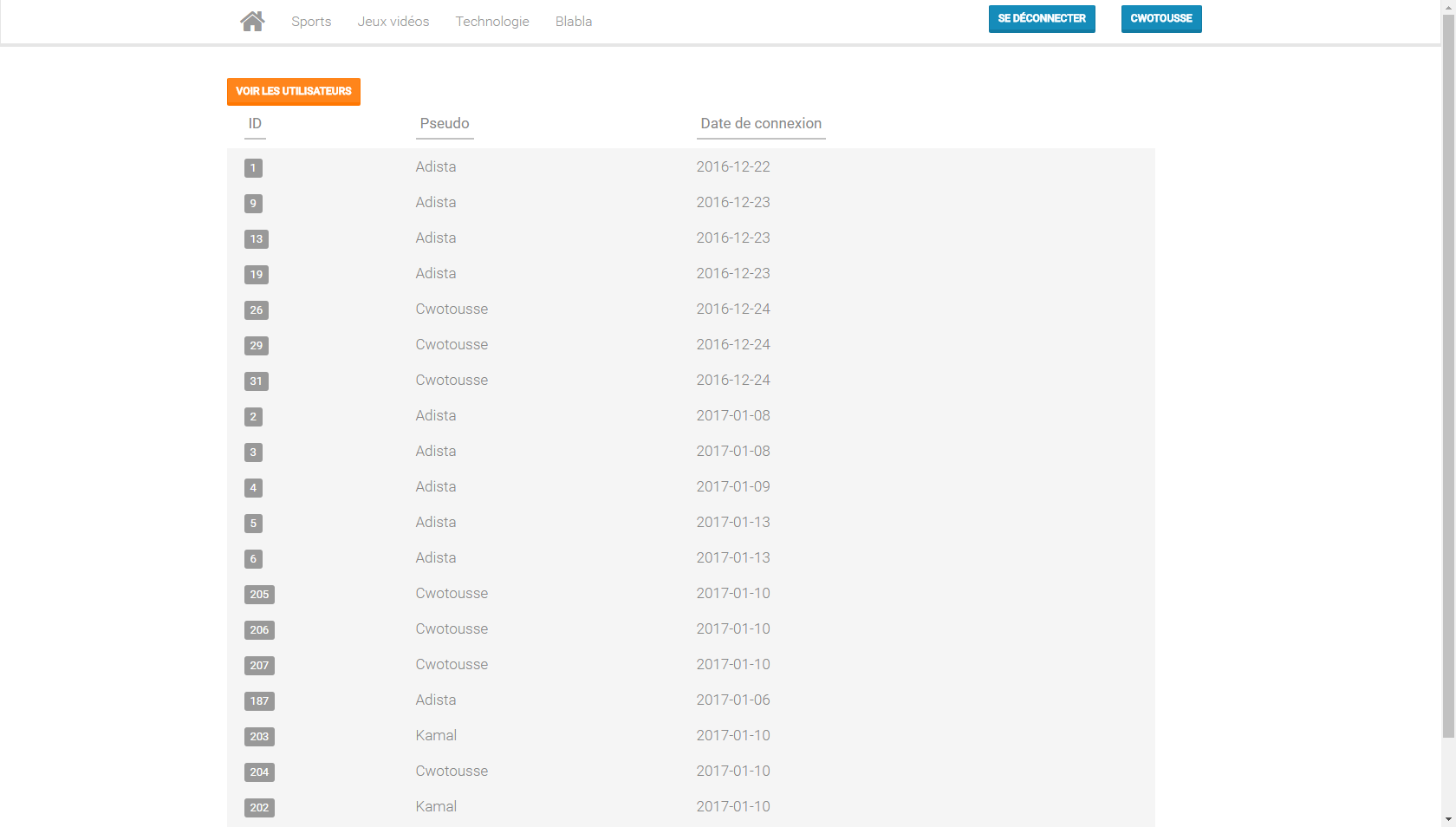


Figure 32 Ecran : historique de connexion

Ce panneau affiche l’id ainsi que le pseudo de l’utilisateur qui s’est connecté. La date de cette connexion est aussi affichée.

## Panneau d’administration

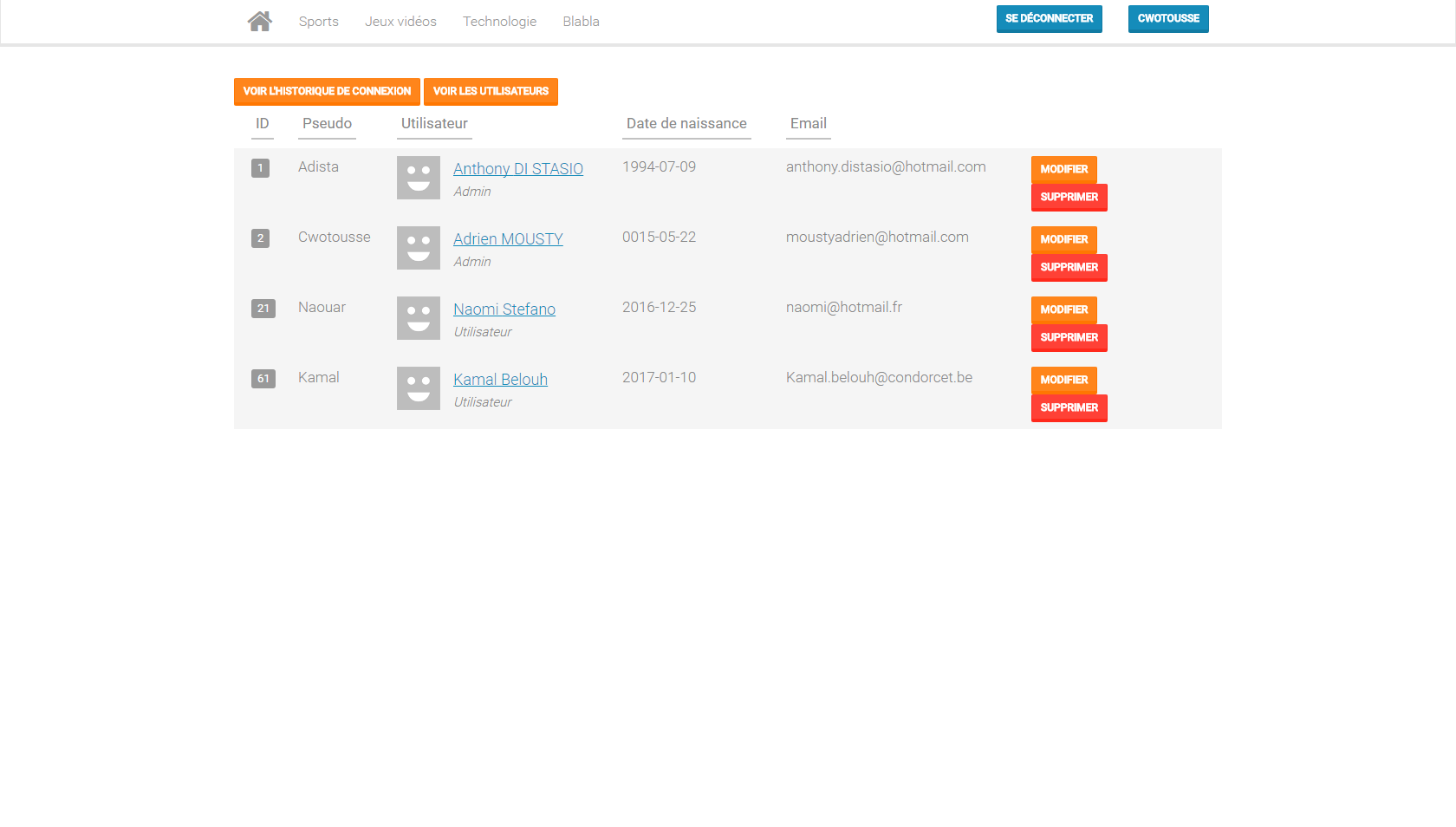


Figure 33 Ecran : panneau de configuration

Celui-ci n’est visible que pour les administrateurs. Il recense tous les utilisateurs. De plus, il permet de promouvoir un utilisateur. Enfin, il permet aussi de modifier ses informations et de les supprimer.

# Navigateurs

Afin d’être sûrs que notre application soit compatible avec plusieurs navigateurs, nous l’avons testée sur Firefox et sur Google Chrome. Le résultat correspond à nos attentes. Les pages affichées sont presque similaires en tout point. À titre de comparaison, nous allons afficher l’accueil ainsi que l’affichage des sujets sur chacun de ces deux navigateurs.

## Firefox

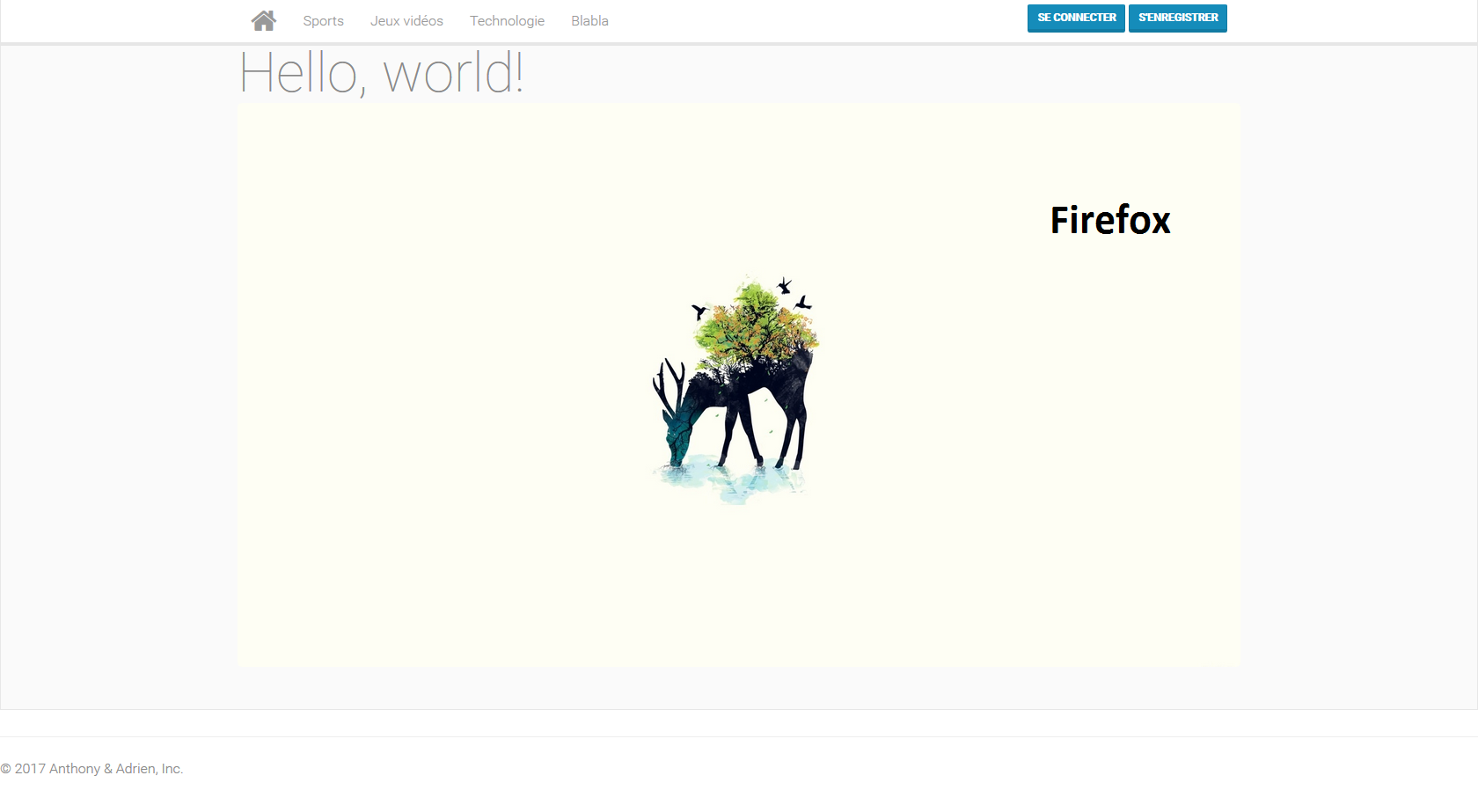


Figure 34 Firefox : accueil

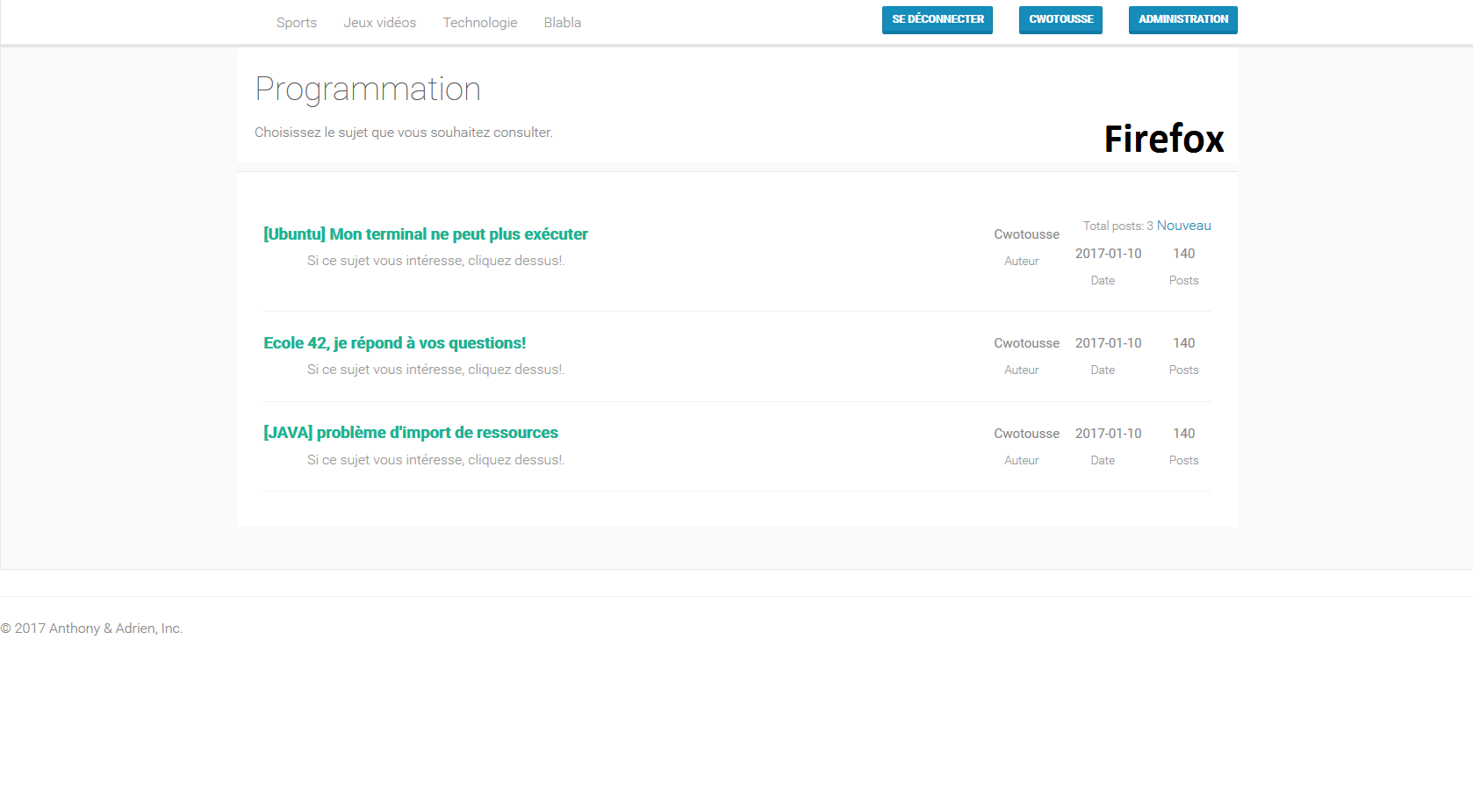


Figure 35 Firefox : sujets

## Google Chrome

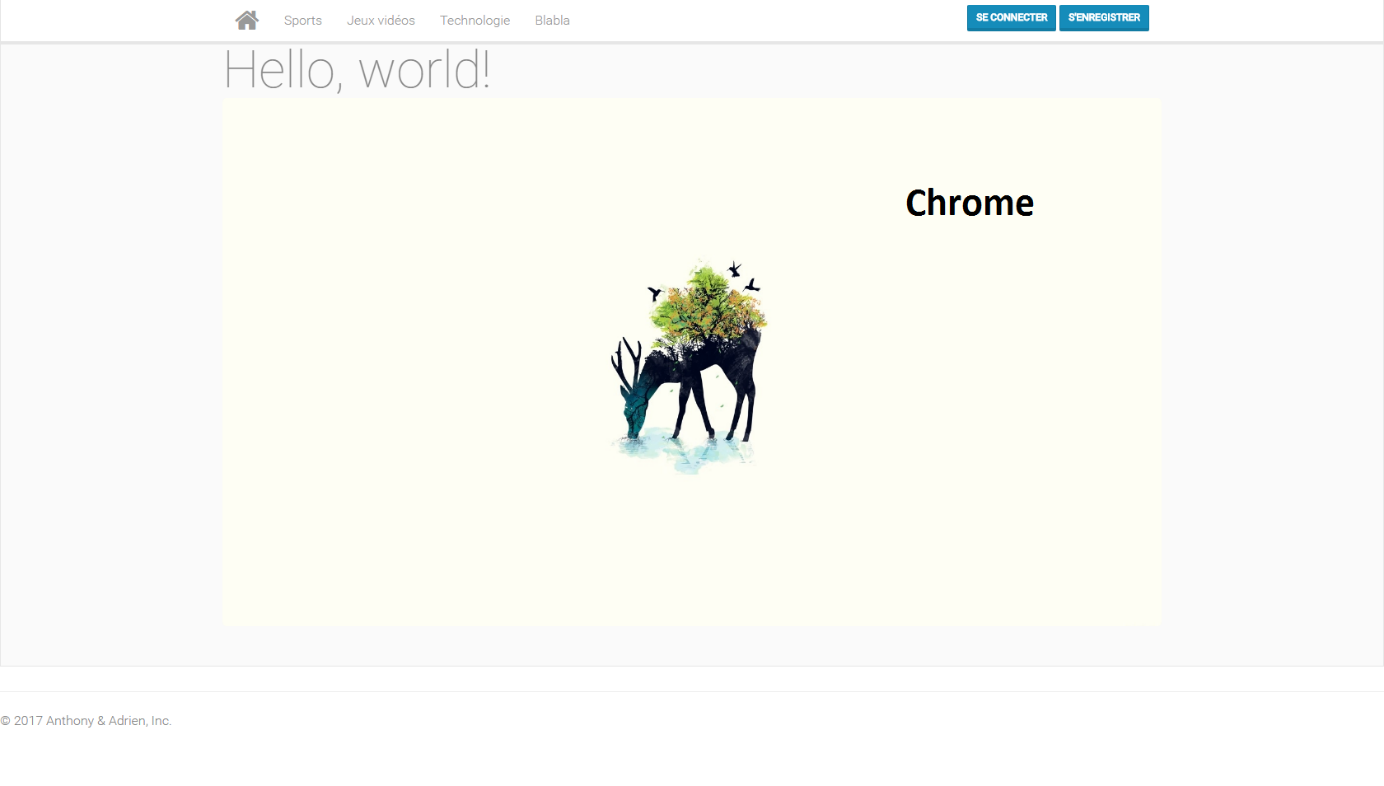


Figure 36 Chrome : accueil

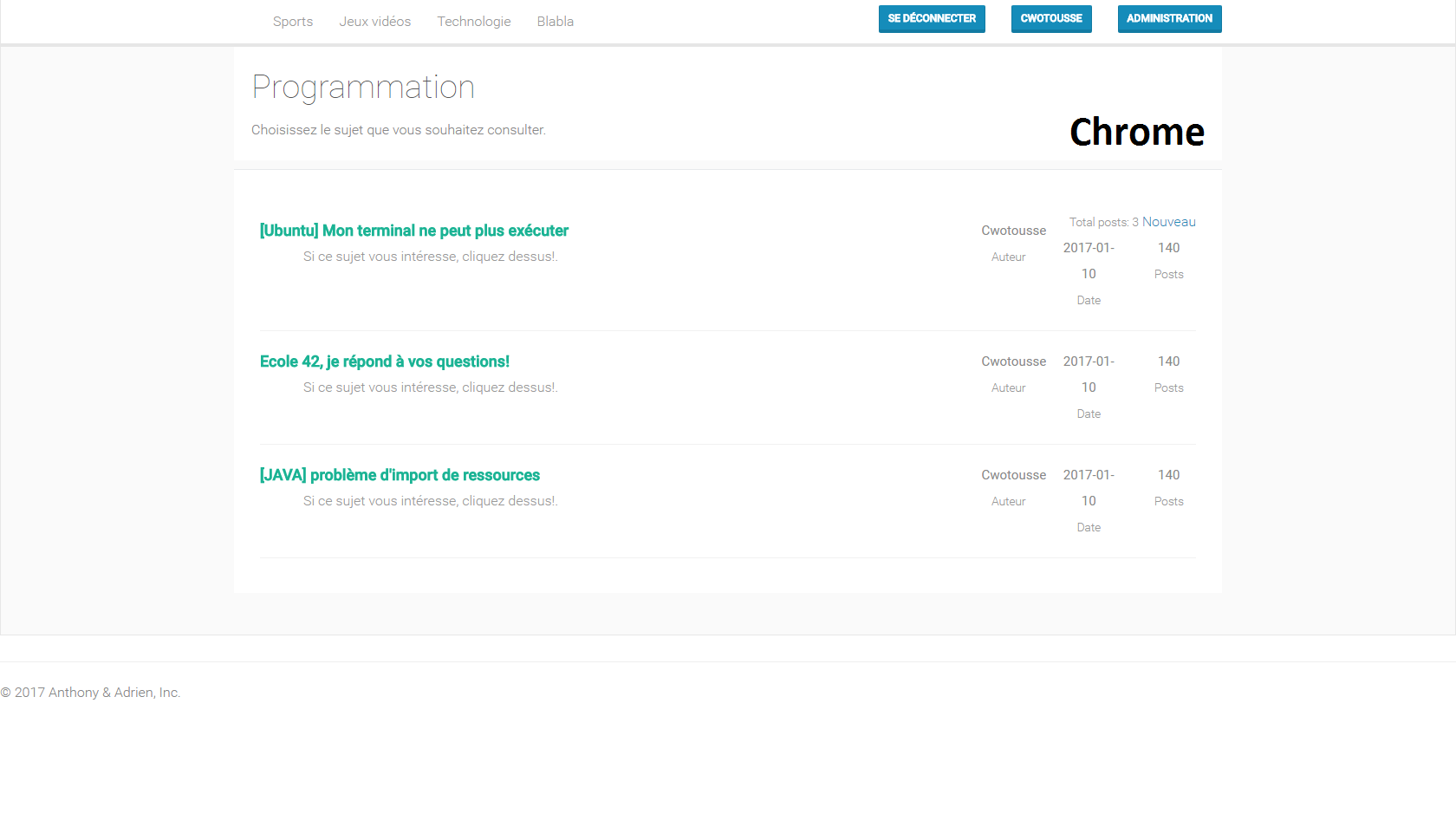


Figure 37 Chrome : sujets

Comme vous pouvez le voir, il n’y a aucune différence. Notre application JEE fonctionne aussi bien sur Firefox que sur Google Chrome. Nous n’avons pas testé sur les autres, car ils sont relativement peu utilisés.

# Nouveautés apportées

## Bootstrap

### Description

Bootstrap est le Framework HTML, CSS et JS le plus populaires pour développer des projets mobiles réactifs et des projets sur le Web. C'est l'un des projets les plus populaires sur la plate-forme de gestion de développement GitHub.



Figure 38 Bootstrap

Nous avons décidé de mettre en place Bootstrap sur notre forum. Cela nous assure un résultat « cross-platform » de qualité. De plus, il gère la compatibilité avec les différentes versions de navigateurs existants. Il faut noter que Bootstrap offre une multitude de templates tous plus beaux les uns que les autres. Cela nous permet de gagner énormément de temps vis-à-vis du design du site. Et puis, cela ne sert à rien de réinventer la roue.

### Utilisation

Premièrement, il faut télécharger la dernière version de Bootstrap (nous avons utilisé la version 3.3.7).

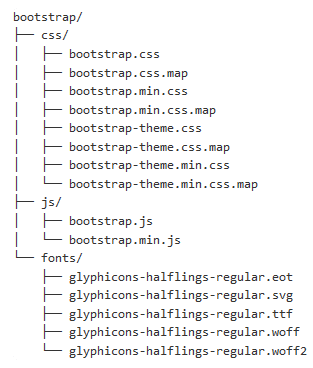


Figure 39 Contenu Boostrap

Une version basique est embarquée avec Bootstrap. Mais nous avons préféré aller télécharger un autre template qui correspondait plus à nos attentes. Suite à cela, nous intégrons Bootstrap à notre projet et par conséquent à Éclipse. La suite se fait automatiquement.

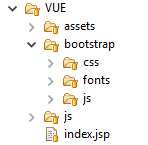
.

Figure 40 Eclipse Bootstrap

## jQuery

### Description



Figure 41 jQuery

Comment ne pas utiliser jQuery à partir du moment où nous développons un site web, qui plus est en utilisant Boostrap qui inclus déjà ce Framework. Comme le résume sa description sur son site web, « *The Write Less, Do More, JavaScript Library*. » C’est pourquoi nous avons décidé de l’utiliser. Bien que nous en sommes restés aux bases qui sont les « hide/show » ainsi que la modification dynamique de valeur et de texte. Mais il y a une multitude de choses à faire grâce à jQuery que nous n’avons pas vraiment eu le temps d’exploiter pour ce projet, malheureusement.

### Utilisation

Nous avons regroupé toutes nos fonctions dans un seul fichier externe, « script.js ». Je dirais même qu’elles sont toutes regroupées dans la fonction **jQuery(document).ready(function() { … }) ;**

#### Document.ready

Une page ne peut pas être manipulée en toute sécurité tant que le document n'est pas prêt. JQuery détecte cet état de préparation. Le code inclus à l'intérieur de $ (document) .ready () ne sera exécuté qu'une fois que la page Document Object Model (DOM) est prête pour le code JavaScript à exécuter. Le code inclus dans $ (fenêtre) .load (function () {...}) s'exécute une fois que la page entière (images ou iframes), et pas seulement le DOM, est prête.

#### Exemple

Ici, lorsque l’utilisateur clique sur l’élément ayant l’id « reply », il va déclencher cette fonction.

Premièrement il va stocker dans une variable la date de création du sujet, en vérifiant bien qu’elle ne contient pas d’espace superflus.

Ensuite, elle va transmettre cette date dans le champ « date-sujet-label ». Elle va aussi transmettre à « nom-sujet-label » le nom du sujet.

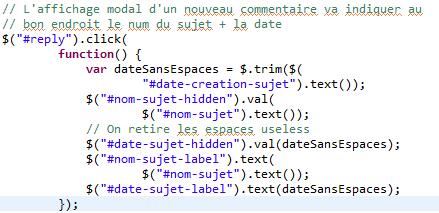


Figure 42 Utilisation jQuery

Cela va permettre d’afficher dynamiquement deux champs à l’affichage modal permettant l’ajout d’un commentaire.



Figure 43 Ajout commentaire

## JSTL

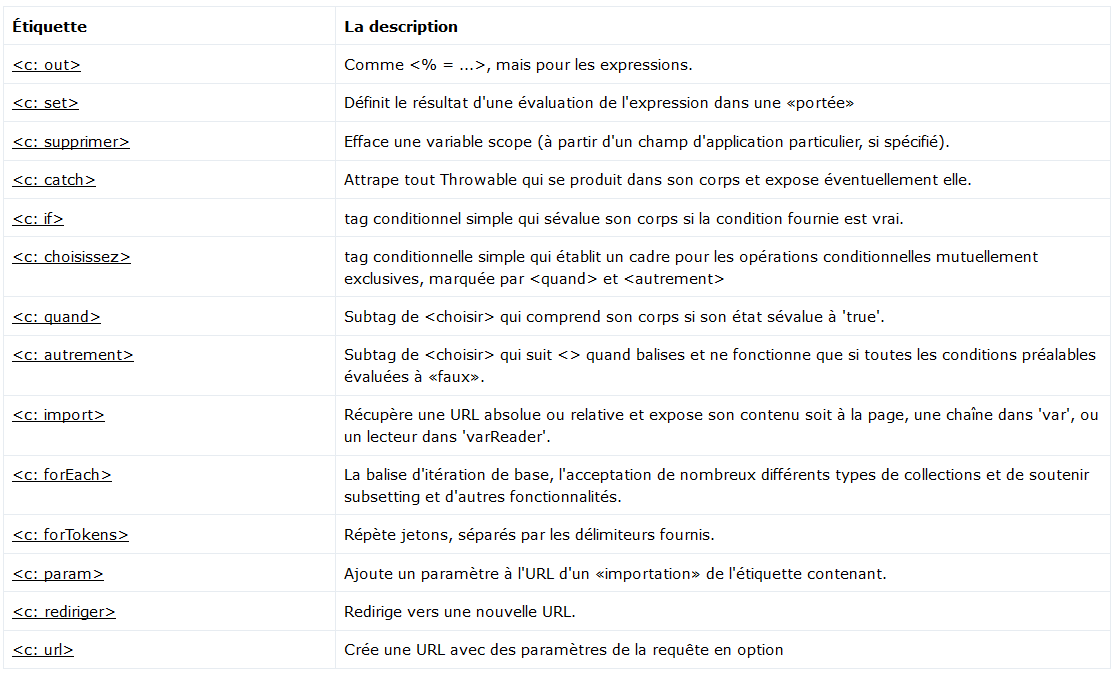
### Description

JSTL est l'acronyme de Java server page Standard Tag Library. Elle a été développée par la Java Community Process (JCP). Nous l’avons découvert, car de très nombreuses applications JEE pour le Web (à base de servlets et JSP) et [Framework](https://fr.wikipedia.org/wiki/Frameworks) Java incluent la JSTL. Elle propose aussi de développer une page JSP sans avoir besoin d’implémenter directement du code Java.  
Elle utilise ainsi une syntaxe proche des langages utilisés par les web designers (principalement HTML) ce qui leur permet de concevoir des pages dynamiques complexes sans connaissances préalables du langage Java. Pour ce projet, nous avons principalement utilisé des tags de structure (itération, conditionnement ...).

Afin de pouvoir l’utiliser dans notre projet, il suffit de fait l’inclue suivant :



### Liste des tags utilisés



### Utilisation

Dans ce projet, nous avons principalement utilisé la balise « foreach » ainsi que la balise « choose / when », mais aussi « otherwise » qui agit comme un « else » d’une alternative « if ». Ces balises equivalent respectivement à un « foreach » et un « switch/case »  en java.

#### Foreach

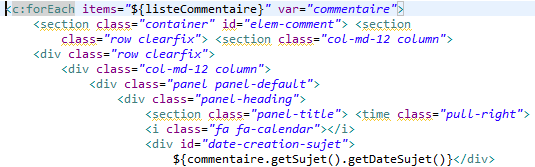


Figure 44 Foreach

Dans cet exemple, nous recevons la liste des commentaires existant pour un sujet choisi. Pour chaque élément nous allons exécuter le code HTML compris entre la balise d’ouverture et de fermeture du « foreach ».

Nous pouvons voir aussi qu’à la fin de la Figure 10 Foreach que nous affichons la date du sujet issu de la liste à chaque fois.

#### Choose … when

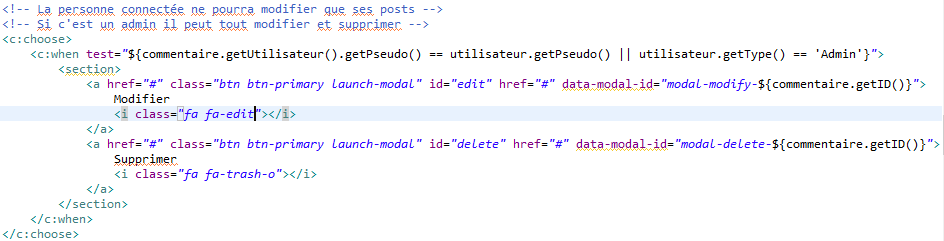


Figure 45 Choose

Ce bout de code affiche le bouton « modifier » et « supprimer » pour chaque commentaire correspondant aux conditions situées dans la balise « when ». Si la personne connectée est la même que celle qui a laissé ce commentaire, alors elle voit les boutons.

Si la personne connectée est un administrateur, alors il aura accès à tous les boutons.

#### Utilisation des balises « when » et « otherwise » en JSTL pour éviter les filtres

Pour éviter d’effectuer des filtres pour rediriger vers la page d’accueil en cas de notation d’une autre page « .jsp » dans l’URL du navigateur, nous avons utilisé les balises « when » et « otherwise ».

Comme vous avez pu remarquer, nous avons une page « restrained\_access.jsp » qui possède la partie administration, cette page n’est donc pas accessible à un utilisateur lambda. Nous devons donc interdire l’accès à cette personne. Comme nous savons, il existe plusieurs façons pour faire cela. Mais nous avons choisi la facilité, cela ne nécessite que 3 lignes de code.

Ce que nous faisons est très simple, au début de la page, nous ouvrons alors la balise « choose » qui permet d’accéder aux balises « when » et « otherwise ». Ensuite nous ouvrons la balise « when » ou nous posons nos conditions.

|  |
| --- |
| <c:when test=*"*${sessionScope.utilisateur.getType().equals('Admin')}*"*> |

Nous récupérons alors le type de l’utilisateur connecté, grâce à la session. Si ce type est égal à « Admin », alors, il peut accéder à cette page. Et voir la partie administration.

Nous fermons bien sur la balise lorsque nous voulons placer la balise « otherwise », qui comme dit plus tôt, agit comme un « else » d’une alternative « if ».

|  |
| --- |
| </c:when>  <c:otherwise>  <c:redirect url=*"/VUE/index.jsp"* />  </c:otherwise>  </c:choose> |

Quand la condition n’est pas vraie, alors je redirige vers la page d’accueil. Ensuite je ferme les balises « otherwise » et « choose ».

# GIT et GitHub

Git est un outil, vu 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 fichiers : 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’à « push » notre projet sur le serveur est un gain de temps et d’énergie considérable. Étant donné que nous possédions nos comptes respectifs, nous n’avions pas du tout envie de recréer un compte commun, ce qui serait allé 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.

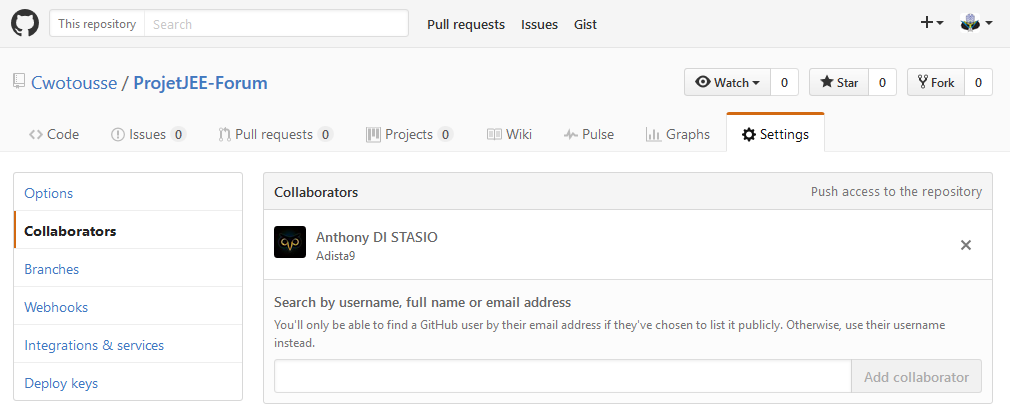


Figure 46 Collaboration GitHub

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.

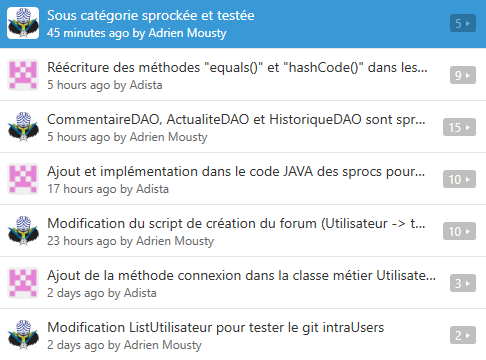


Figure 47 Auteurs GitHub

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 prises en compte.

# Problèmes rencontrés

Lors du développement de ce projet, nous avons rencontré quelques soucis. Ceux-ci nous ont coûté pas mal de temps.

Premièrement, suite à un push sur le serveur (d’une version bien entendu stable et sans erreurs), il fut impossible de relancer cette version. Une mauvaise nouvelle ne venant jamais seule, le message d’erreur demeure extrêmement vague. Il fut donc très difficile de déceler sa provenance.

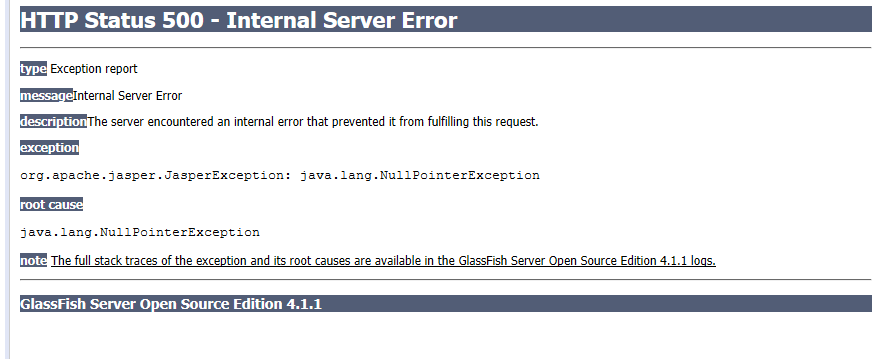


Figure 48 Erreur 500

Ensuite, madame Copin nous a donné jeudi 15 décembre nos logs afin que nous puissions nous connecter sur la base de données distante. Nous la testons à l’école et changeons notre mot de passe. Tout se passe bien, nous faisons tourner notre script et toutes les données apparaissent correctement. Le week-end arrivant, nous tentons de nous connecter chez nous afin d’avancer sur ce projet. Mais quelle ne fut pas notre surprise lorsque ce doux message apparut :

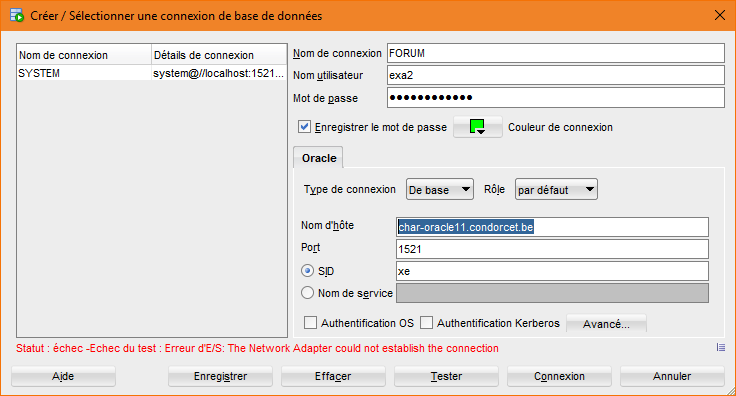


Figure 49 Erreur ES

Suite à cela, nous recherchons des informations sur Internet afin de trouver la provenance de cette erreur.

Un site spécialisé nous proposé de relancer le  « Listener process » (service). Ce que nous faisons. Cela ne fonctionne pas.

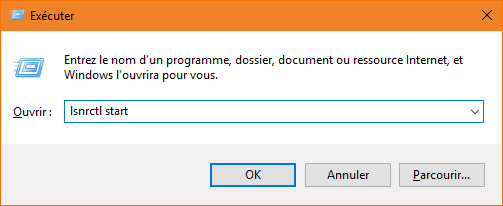


Figure 50 listener process (service) is not running

Après un certain temps passé, en vain, à trouver l’origine de l’erreur, nous demandons à nos collègues afin de savoir si cela se produisait aussi chez eux. Ils répondent par l’affirmative. Nous en déduisons donc que nous ne pouvons nous connecter que si nous sommes sur le réseau de l’école.

Quelques jours plus tard, le problème s’est réglé de lui-même. Néanmoins, nous avons continué à avoir des problèmes avec la base de données jusqu’à la fin de notre projet. Il arrivait que parfois le site s’exécutât sans récupérer d’informations de la base de données.

En continuant sur la longue lancée des problèmes liés à la base de données, il était aussi parfois très difficile de pouvoir tester notre code ou de compter sur notre base de données dans SQLDevelopper.

En effet, la base de données était souvent mystérieusement « occupée ». Pendant ce temps nous ne pouvions rien faire si ce n’est admirer ces jolis messages :

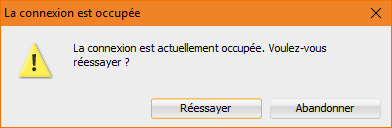


Figure 51 Connexion occupée

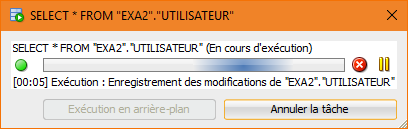


Figure 52 Query en attente

Ce message s’affichait indéfiniment. Bien entendu, le problème n’était pas dû au fait que nous étions à deux personnes voulant accéder à la base de données. Car nous avons travaillé pour la plupart du temps ensemble, l’un à côté de l’autre.

Nous avons eu aussi des problèmes avec Git. En effet, dès que nous modifions le même fichier, sans pour autant modifier les mêmes lignes de codes, un de nous devait « discard » sa mise à jour du fichier. Car GitHub n’acceptait pas de synchroniser. Ce fut encore plus fastidieux lorsque nous avons tentés de chacun travailler sur une branche différente, puis en voulant les fusionner à la branche « Master ». Ce fut impossible.

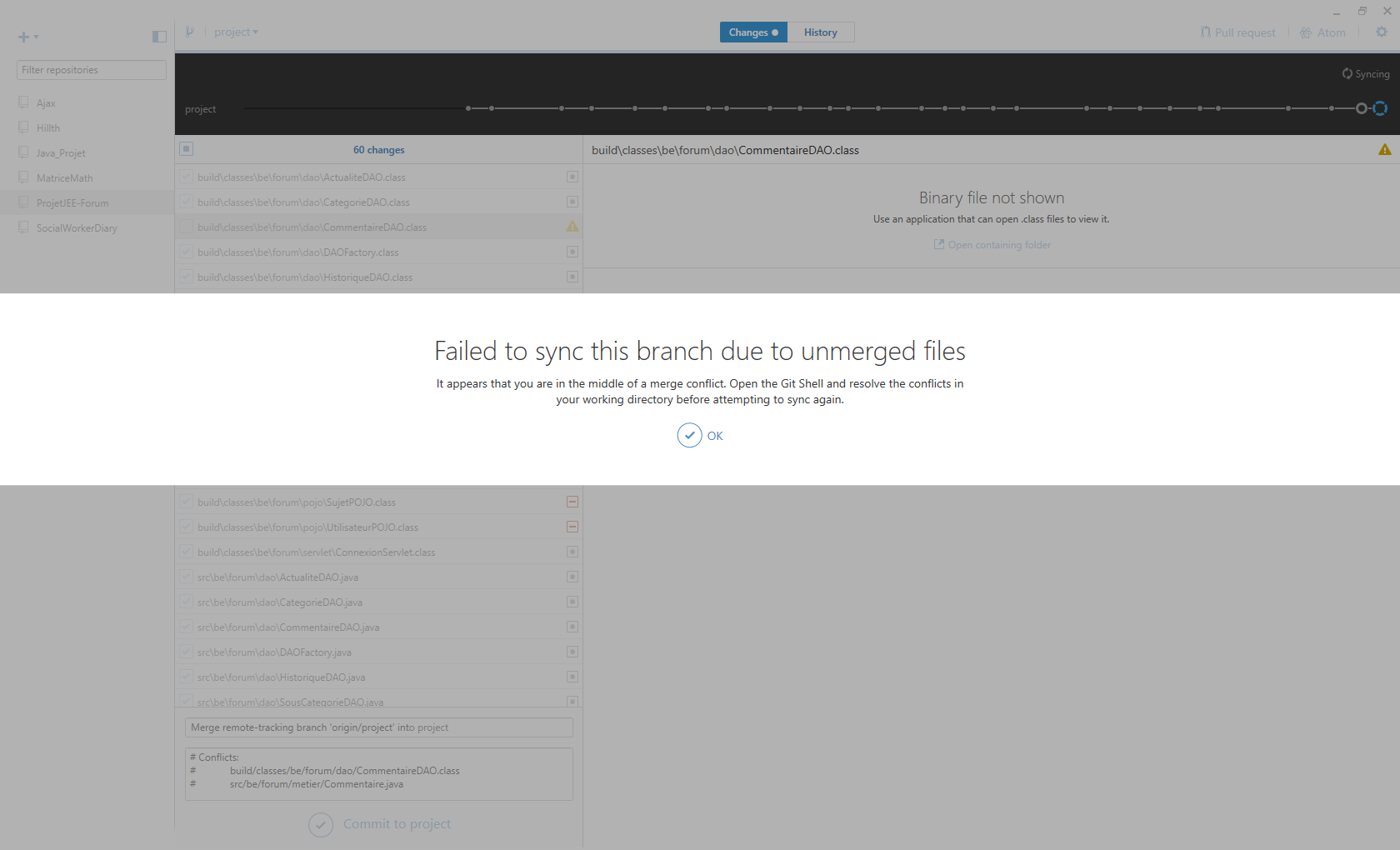


Figure 53 Erreur GitHub

Il était donc relativement difficile pour nous de travailler en même temps, à moins que nous travaillions sur des parties diamétralement opposées.

Il nous est aussi arrivé de modifier ce dossier, ou les autres documents et de tout perdre en synchronisant notre projet avec la nouvelle version de l’autre, qui effaçait nos précédentes modifications.

# Conclusion

Pour conclure, nous pouvons dire sans hésiter que ce projet fut extrêmement intéressant à développer. Celui-ci fut plus simple à implémenter que la gestion d’une station de ski. Et ce, pour plusieurs raisons.

Premièrement, nous ne sommes pas seuls, nous avons donc 2 fois plus de connaissances et de temps.

Deuxièmement, car nous avons déjà acquis une certaine connaissance en matière de DAO.

Troisièmement, car l’utilisation de Bootstrap nous permet de nous focaliser sur le code et d’allouer relativement peu de temps aux choses futiles tel le design du forum.

Quatrièmement, le sujet étant libre, cela nous permet d’implémenter « ce que l’on veut ». Si nous jugeons un élément intéressant, nous pouvons l’implémenter en toute liberté.

De plus, grâce à l’utilisation de Git nous n’avons plus à nous envoyer l’un à l’autre les nouvelles versions du projet. Les mises à jour du code sont explicitées clairement, nous savons qui a modifié quelle partie du code et surtout nous avons l’historique des mises à jour.

Le seul bémol fut peut-être le fait de devoir coder ce projet pendant notre blocus. Cela nous réduit donc notre temps d’étude, qui fut déjà réduit dû à un autre projet en parallèle, celui d’Android. Nous pouvons ajouter aussi les nombreux problèmes rencontrés qui nous ont fait perdre pas mal de temps.

# Table des figures

[Figure 1 Hiérarchie 6](#_Toc471932694)

[Figure 2 Use Case 7](#_Toc471932695)

[Figure 3 Diagramme de classe 8](#_Toc471932696)

[Figure 4 Schéma de la base de données 9](file:///C:\Users\Anthony\Desktop\ProjetJEE-Forum.docx#_Toc471932697)

[Figure 5 Connexion Oracle 10](#_Toc471932698)

[Figure 6 Connexion Oracle – Java 10](#_Toc471932699)

[Figure 7 table Utilisateur 11](#_Toc471932700)

[Figure 8 Foreign key Historique Utilisateur 11](#_Toc471932701)

[Figure 9 Les séquences 12](#_Toc471932702)

[Figure 10 Insert utilisateur 13](#_Toc471932703)

[Figure 11 Delete utilisateur 14](#_Toc471932704)

[Figure 12 Update utilisateur 15](#_Toc471932705)

[Figure 13 Select Utilisateur 16](#_Toc471932706)

[Figure 14 Get List Utilisateur 17](#_Toc471932707)

[Figure 15 NO\_DATA\_FOUND 18](#_Toc471932708)

[Figure 16 Exception anonyme 19](#_Toc471932709)

[Figure 17 exemple fonction stockée (1) 20](#_Toc471932710)

[Figure 18 Exemple fonction stockée (2) 20](#_Toc471932711)

[Figure 19 Package 1 22](#_Toc471932712)

[Figure 20 Exemple déclencheur 23](#_Toc471932713)

[Figure 21 Pattern MVC 24](#_Toc471932714)

[Figure 22 Gestion erreur XML 32](#_Toc471932715)

[Figure 23 Redirection erreur 32](#_Toc471932716)

[Figure 24 Ecran : connexion 33](#_Toc471932717)

[Figure 25 Ecran : profil 33](#_Toc471932718)

[Figure 26 Ecran : home 34](#_Toc471932719)

[Figure 27 Ecran : sous-catégorie 34](#_Toc471932720)

[Figure 28 Ecran : sujets 35](#_Toc471932721)

[Figure 29 Ecran : erreur 35](#_Toc471932722)

[Figure 30 Ecran : affichage commentaire 36](#_Toc471932723)

[Figure 31 Ecran : poster un commentaire 36](#_Toc471932724)

[Figure 32 Ecran : historique de connexion 37](#_Toc471932725)

[Figure 33 Ecran : panneau de configuration 37](#_Toc471932726)

[Figure 34 Firefox : accueil 38](#_Toc471932727)

[Figure 35 Firefox : sujets 38](#_Toc471932728)

[Figure 36 Chrome : accueil 39](#_Toc471932729)

[Figure 37 Chrome : sujets 39](#_Toc471932730)

[Figure 38 Bootstrap 40](#_Toc471932731)

[Figure 39 Contenu Boostrap 40](#_Toc471932732)

[Figure 40 Eclipse Bootstrap 41](#_Toc471932733)

[Figure 41 jQuery 42](#_Toc471932734)

[Figure 42 Utilisation jQuery 43](#_Toc471932735)

[Figure 43 Ajout commentaire 43](#_Toc471932736)

[Figure 44 Foreach 45](#_Toc471932737)

[Figure 45 Choose 45](#_Toc471932738)

[Figure 46 Collaboration GitHub 47](#_Toc471932739)

[Figure 47 Auteurs GitHub 48](#_Toc471932740)

[Figure 48 Erreur 500 49](#_Toc471932741)

[Figure 49 Erreur ES 49](#_Toc471932742)

[Figure 50 listener process (service) is not running 50](#_Toc471932743)

[Figure 51 Connexion occupée 50](#_Toc471932744)

[Figure 52 Query en attente 51](#_Toc471932745)

[Figure 53 Erreur GitHub 51](#_Toc471932746)

# Bibliographie

*BootStrap*. (s.d.). Consulté le 12 3, 2016, sur getbootstrap: http://getbootstrap.com/

Copin, B. (2016). *PL/SQL.*

JQuery Foundation. (s.d.). *JQuery*. Consulté le 12 22, 2016, sur jquery: https://jquery.com/

Microsoft. (s.d.). *Microsoft*. Consulté le 12 29, 2016, sur CREATE TRIGGER (Transact-SQL): https://msdn.microsoft.com/fr-fr/library/ms189799.aspx

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/

OpenClassRooms. (2016, 10 31). *Créez votre application web avec Java EE*. Consulté le 12 4, 2016, sur OpenClassRooms: https://openclassrooms.com/courses/creez-votre-application-web-avec-java-ee/la-technologie-jsp-1-2

TutorialsPoint. (s.d.). *JSP - Standard Tag Library (JSTL) Tutorial*. Consulté le 12 23, 2016, sur tutorialspoint: https://www.tutorialspoint.com/jsp/jsp\_standard\_tag\_library.htm

TutorialsPoint. (s.d.). *PL/SQL - Triggers*. Consulté le 12 29, 2016, sur TutorialsPoint: https://www.tutorialspoint.com/plsql/plsql\_triggers.htm

Vandevorst, A. (2016). *JEE.*

1. Data Manipulation Language [↑](#footnote-ref-1)