Anthony Di Stasio, Adrien Mousty

**2016 - 2017**

**Forum actif**

**Projet JEE – Partie PL/SQL**

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.*

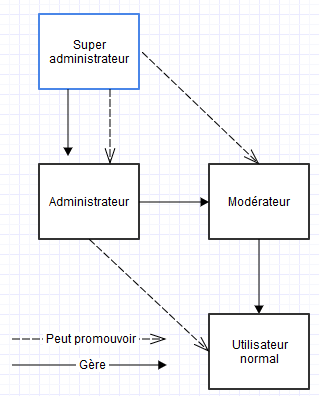
**

Figure 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

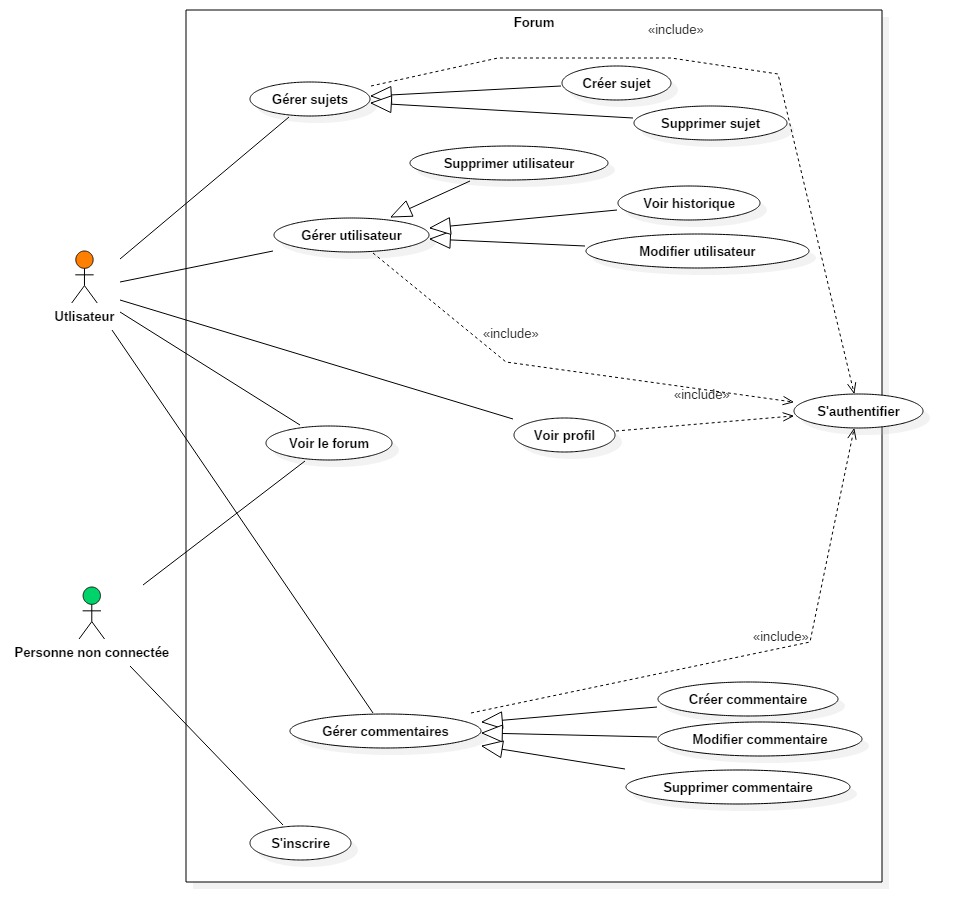


Figure Use Case

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.

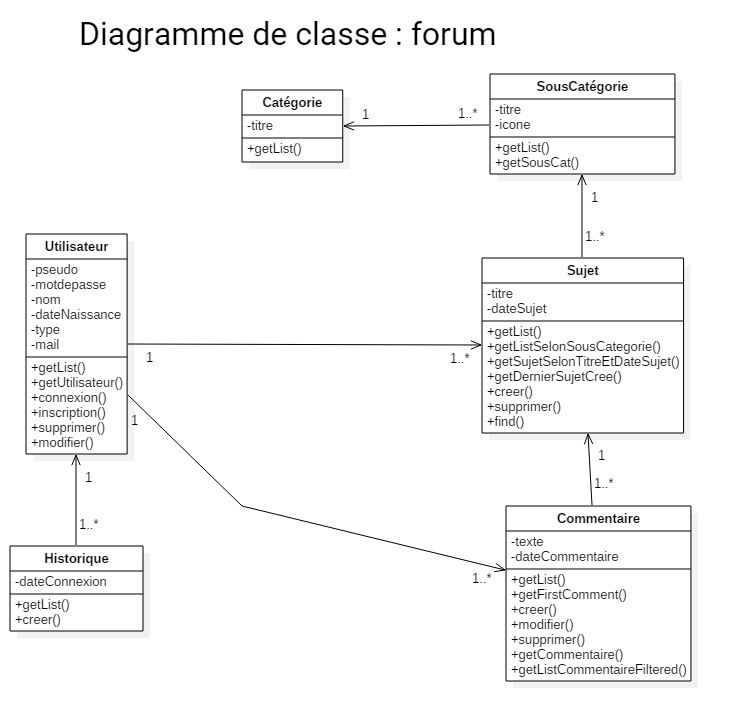


Figure Diagramme de classe

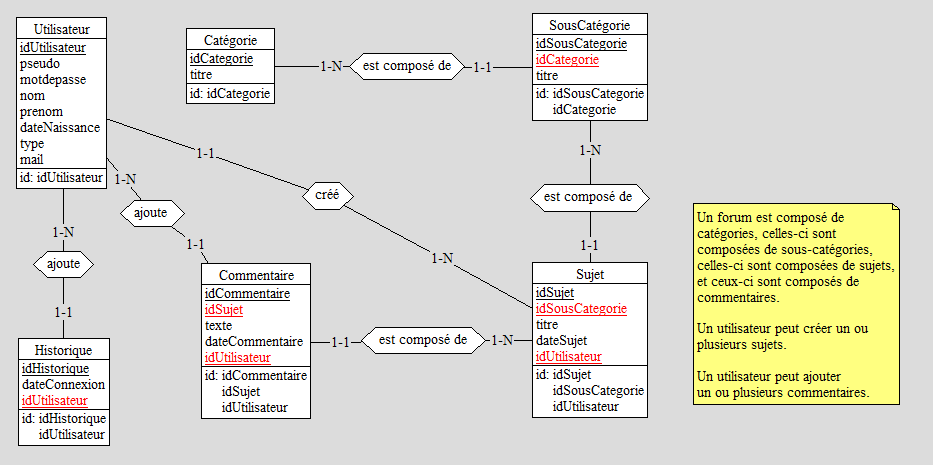
Le diagramme de classe est sensiblement équivalent au schéma conceptuel, seulement les clés primaires et étrangères sont absentes. Une autre différence est l’inversion des cardinalités.

# 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 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.

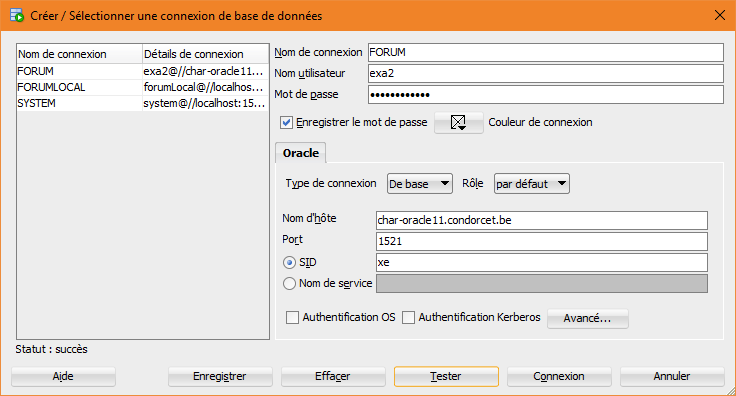


Figure Connexion Oracle

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.

## PL/SQL

### Les tables

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

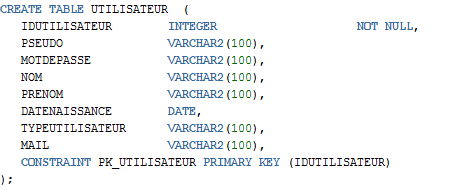


Figure table Utilisateur

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.

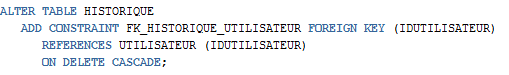


Figure Foreign key Historique Utilisateur

### 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.

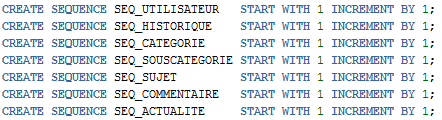


Figure Les séquences

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 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.

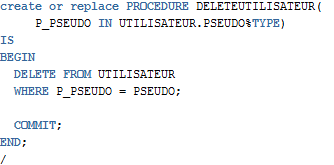


Figure Delete utilisateur

#### 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.

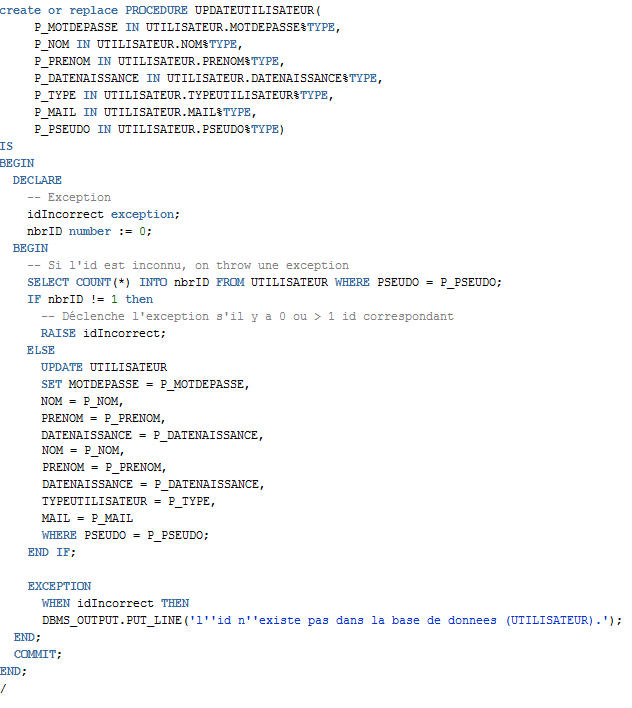


Figure Update utilisateur

#### 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.

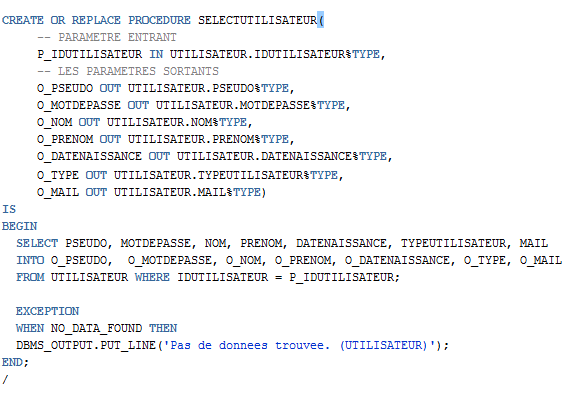


Figure Select Utilisateur

#### 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 ».

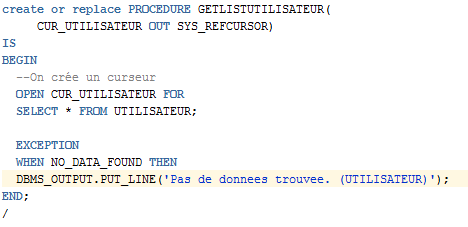


Figure Get List Utilisateur

### 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.

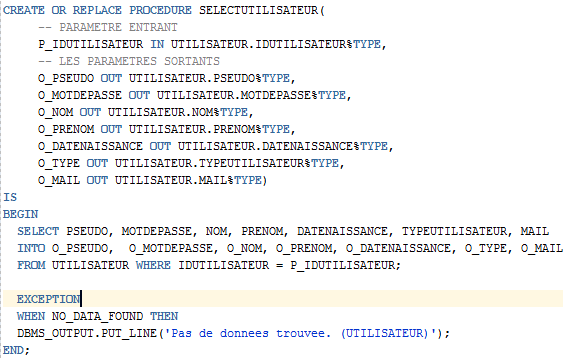


Figure NO\_DATA\_FOUND

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.

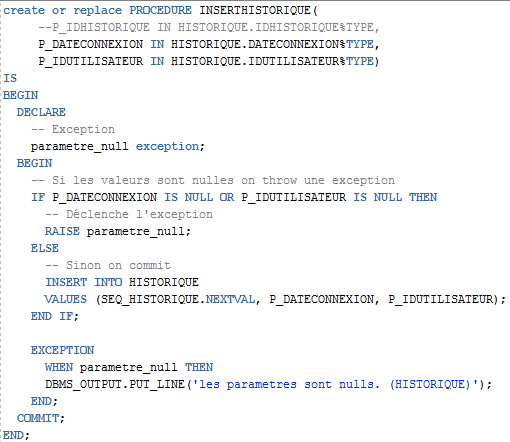


Figure Exception anonyme

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

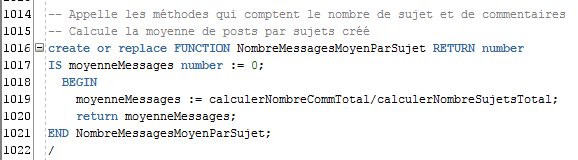


Figure exemple fonction stockée (1)

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.

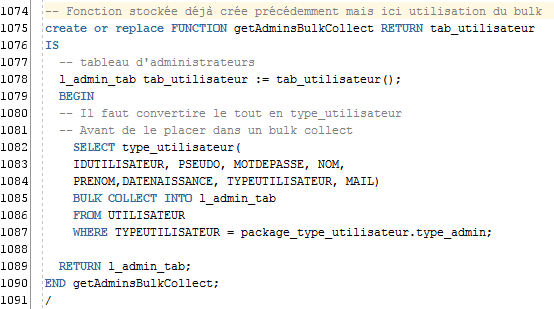


Figure Exemple fonction stockée (2)

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 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

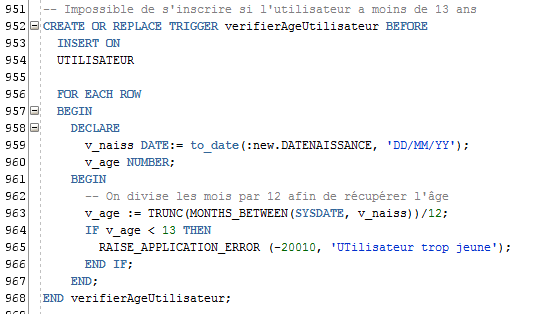


Figure Exemple déclencheur

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.

# Screenshots du site web

## Connexion

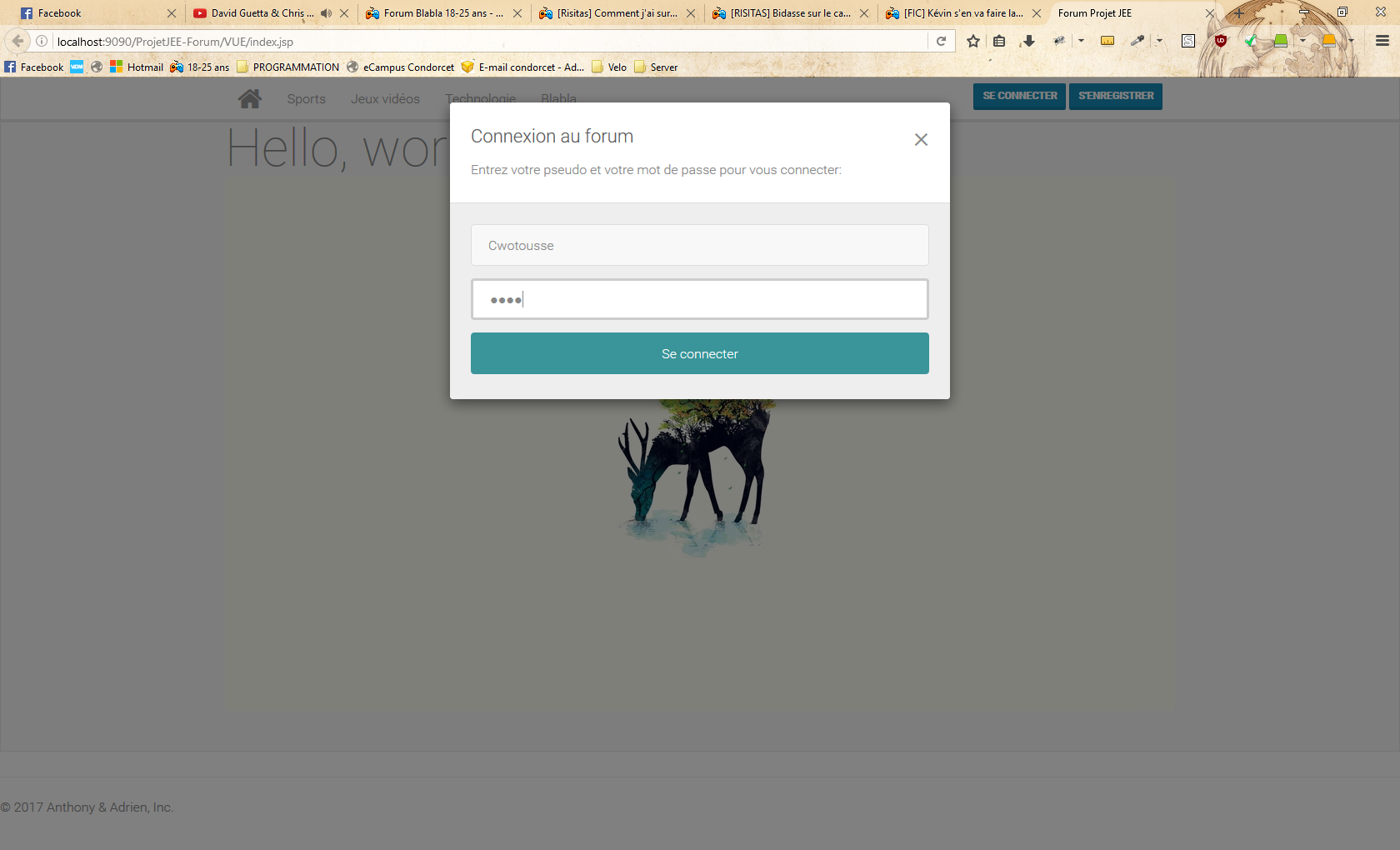


Figure Ecran : connexion

## Profil

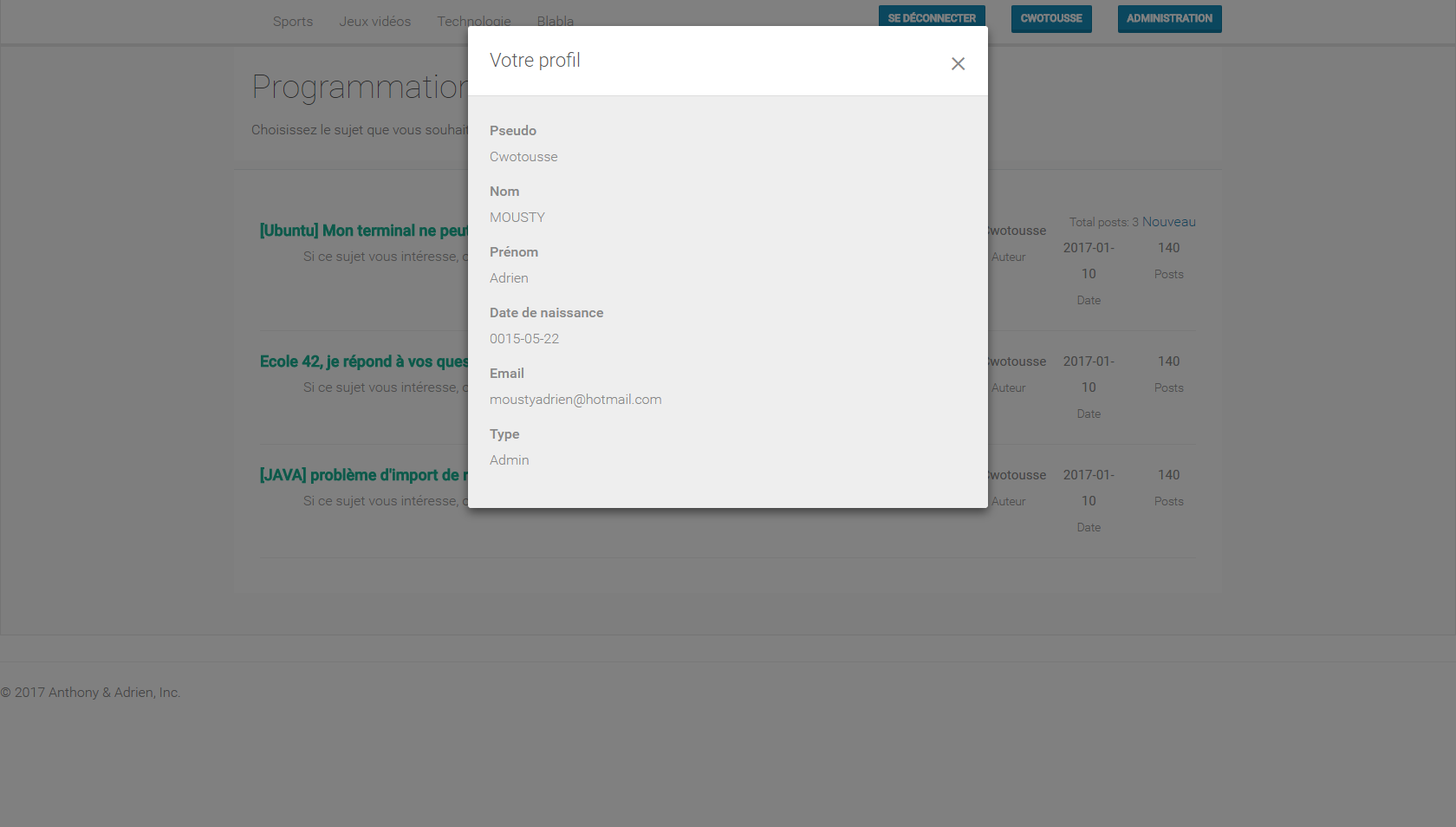


Figure Ecran : profil

## Vue générale du forum

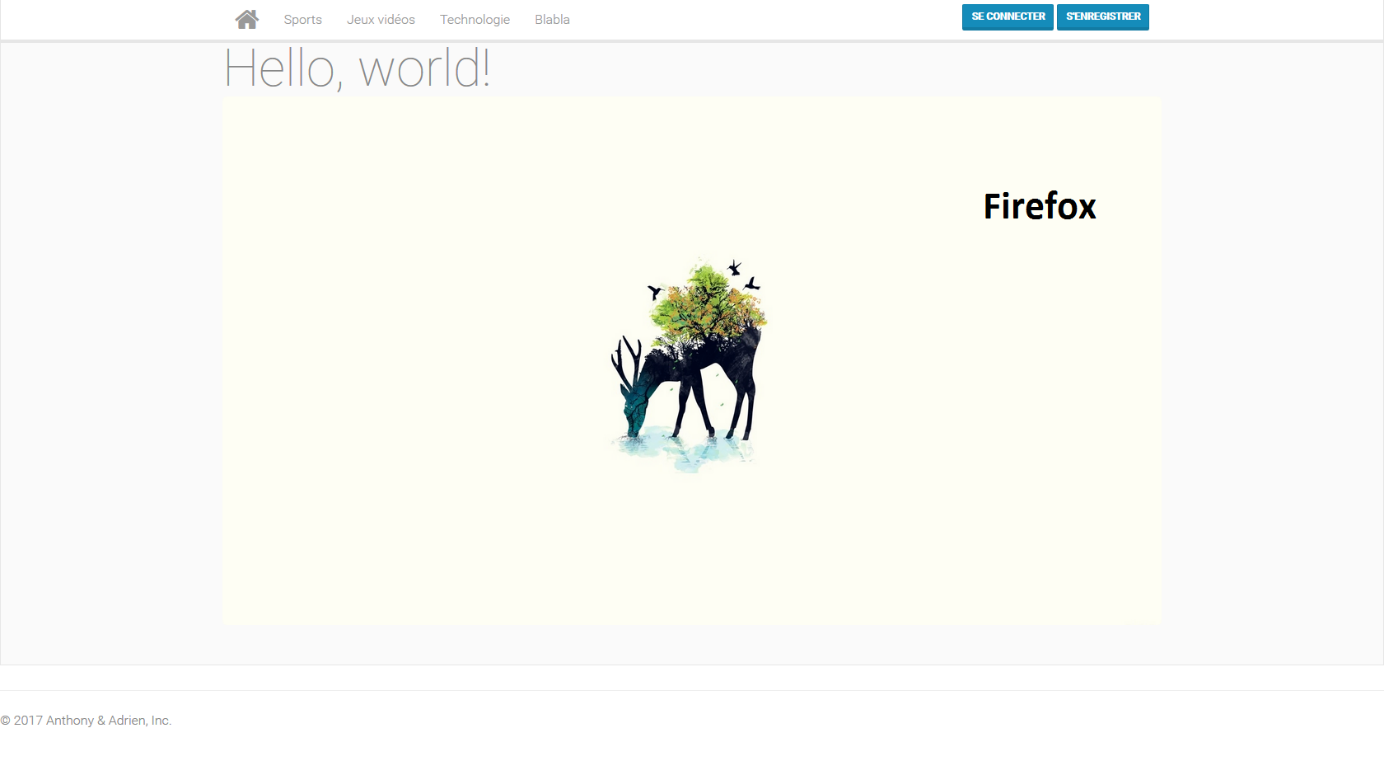


Figure Ecran : home

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

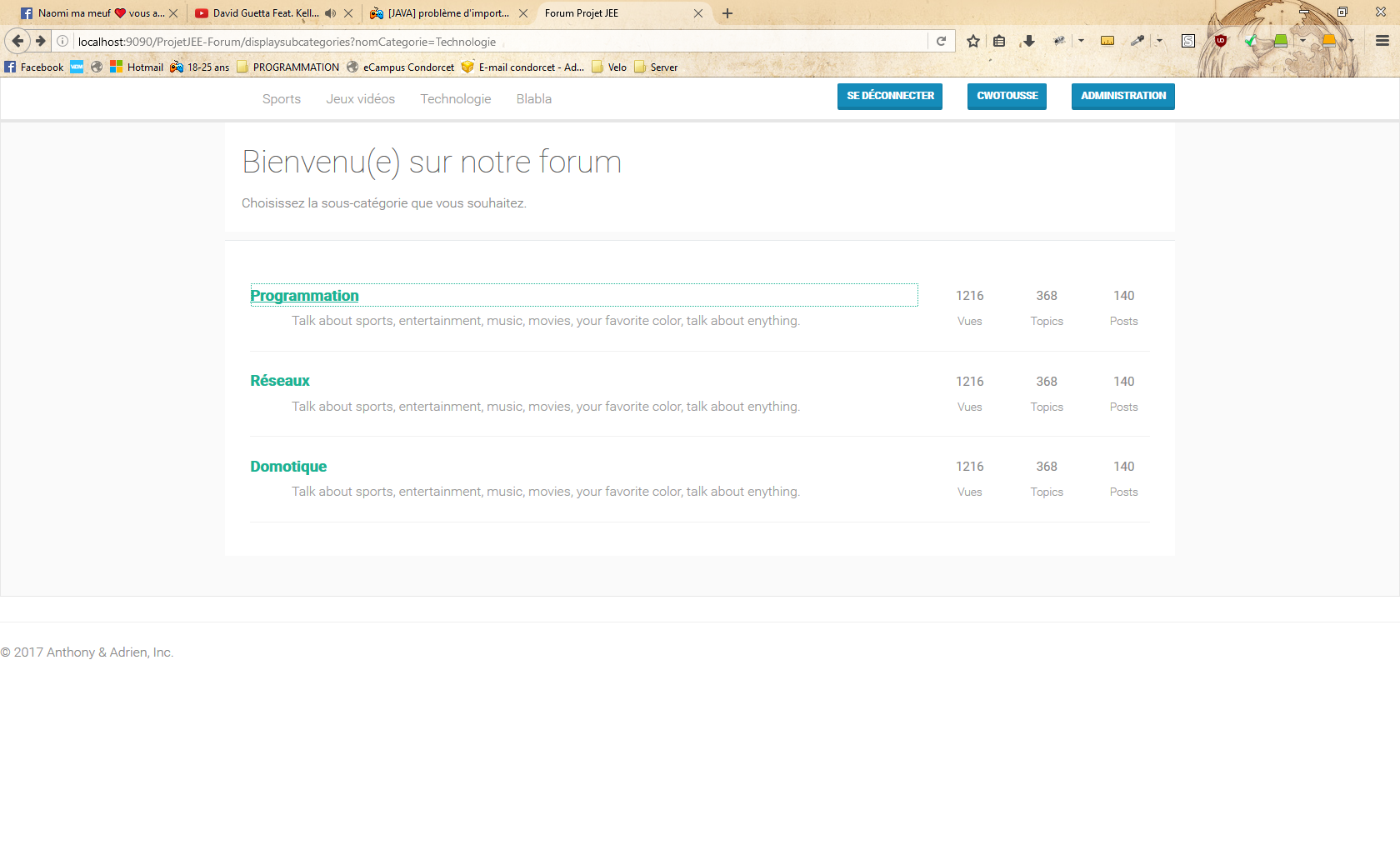


Figure Ecran : sous-catégorie

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

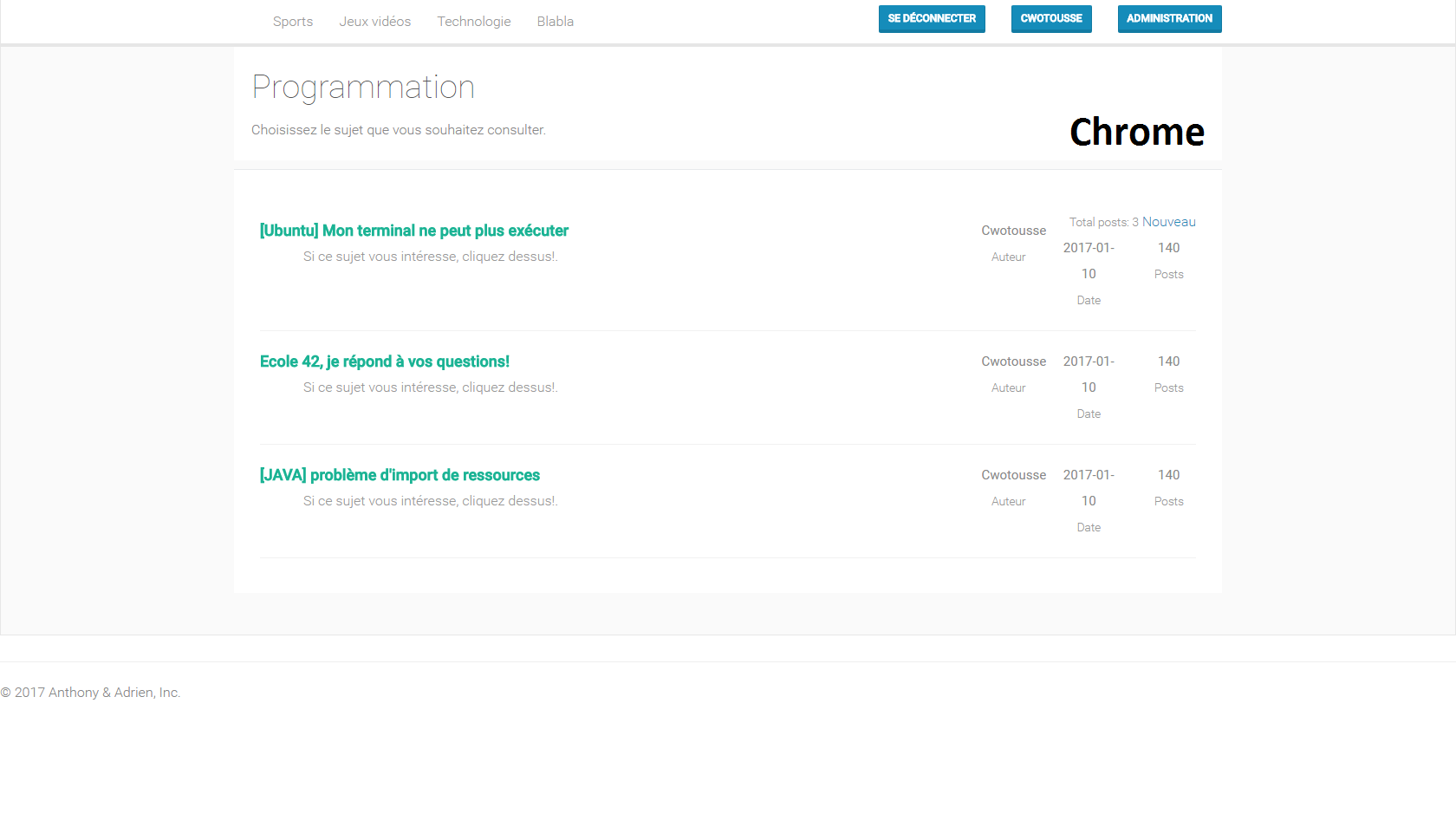


Figure Ecran : sujets

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 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

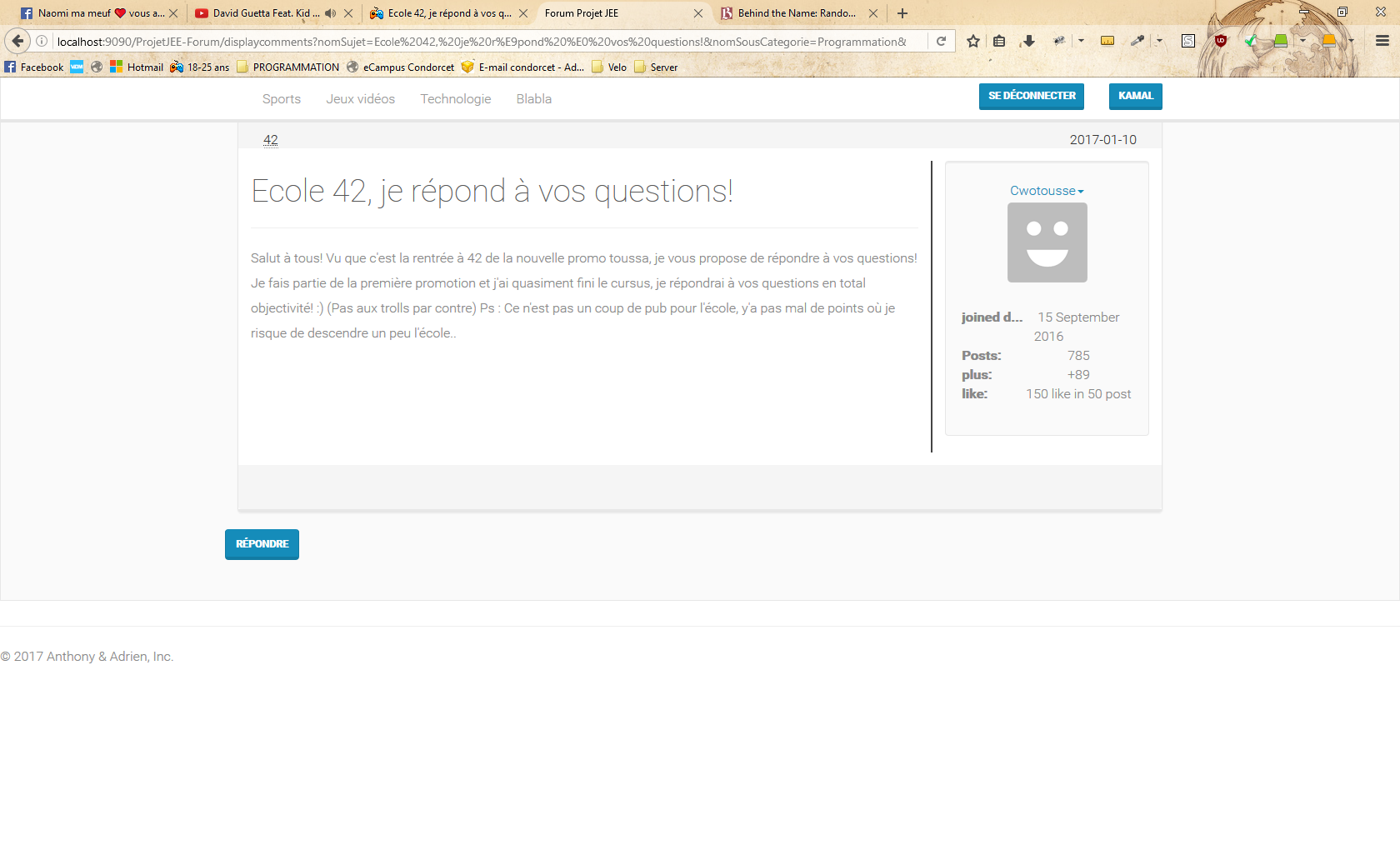


Figure Ecran : affichage commentaire

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

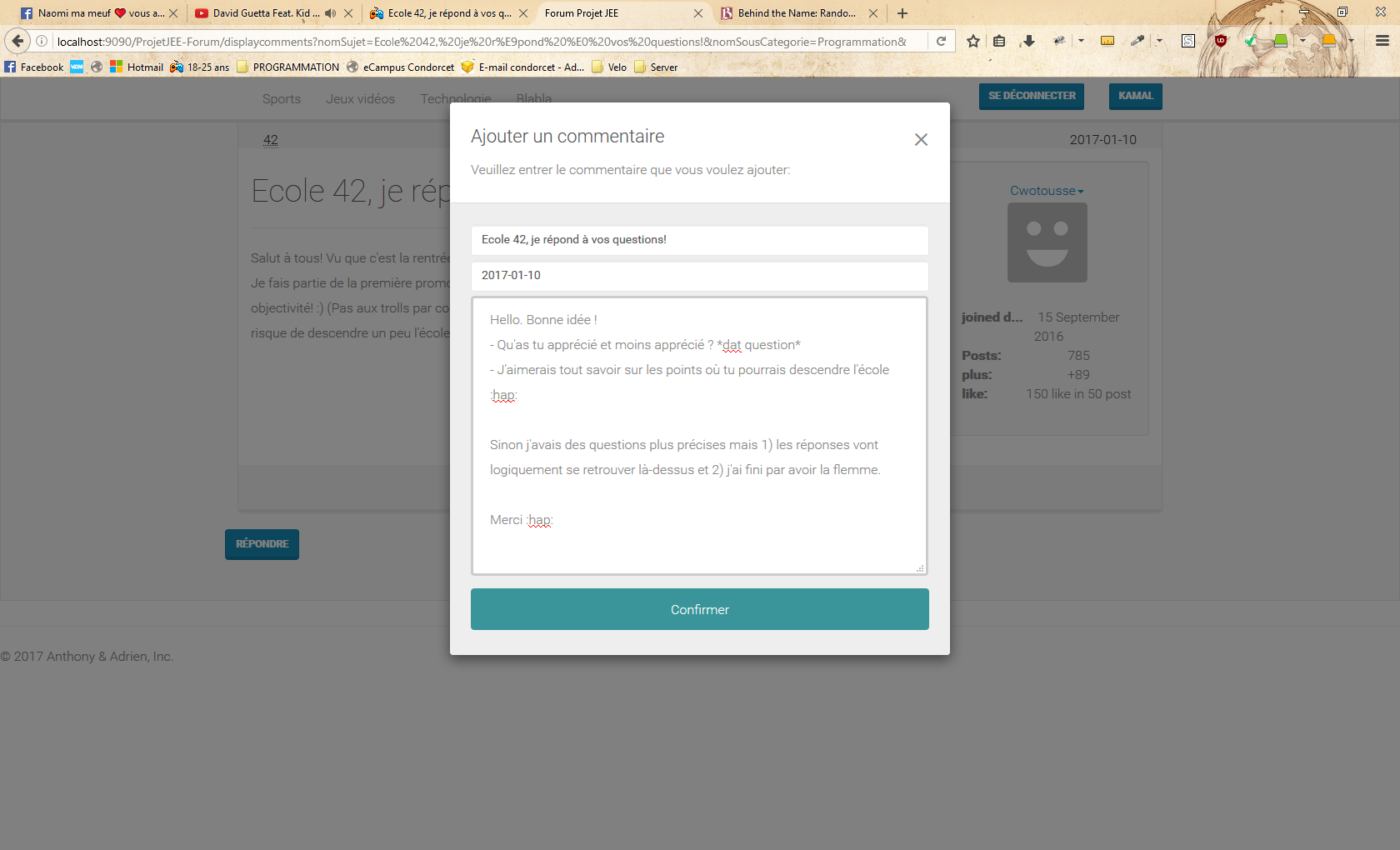


Figure Ecran : poster un commentaire

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

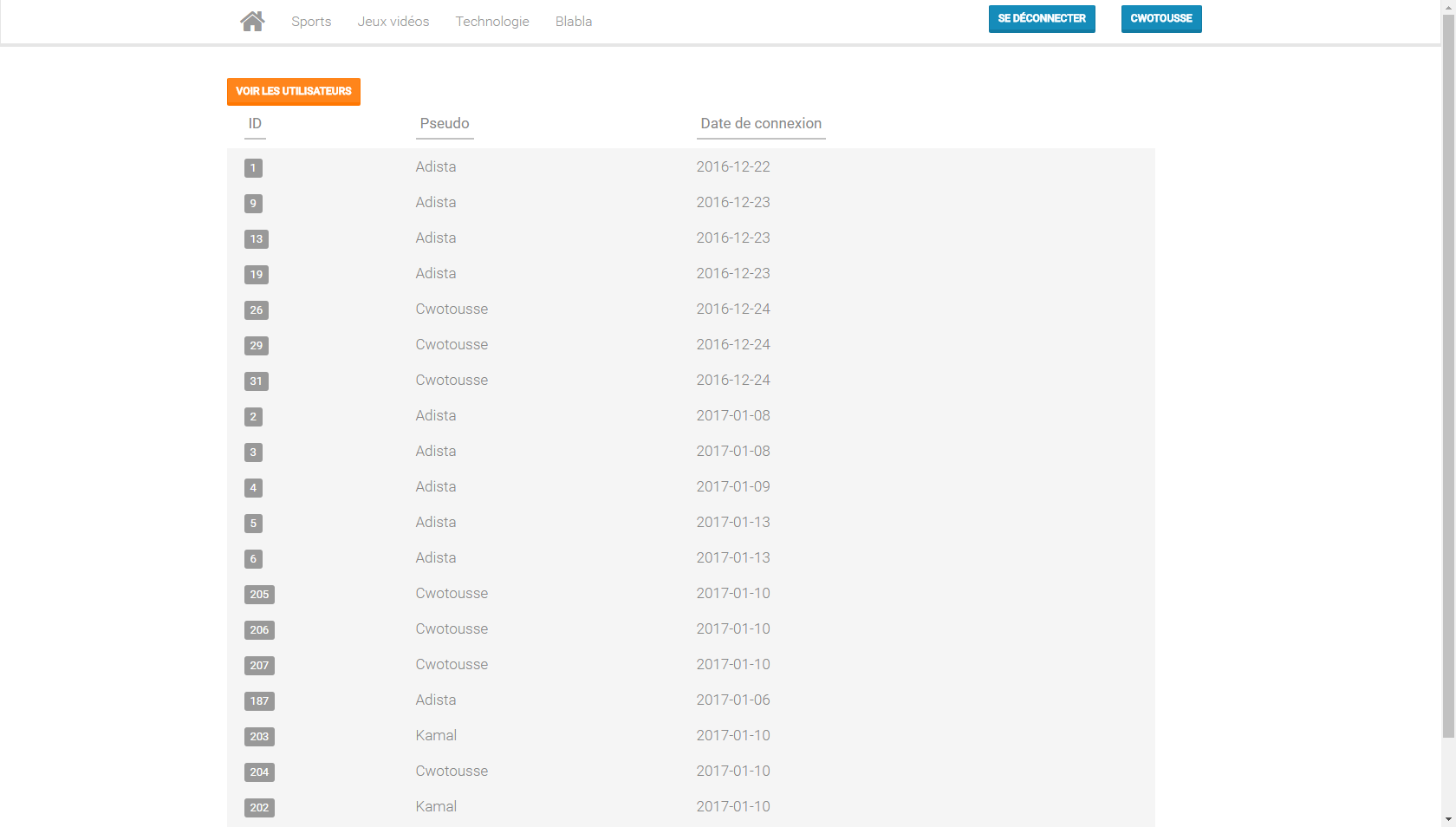


Figure Ecran : historique de connexion

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

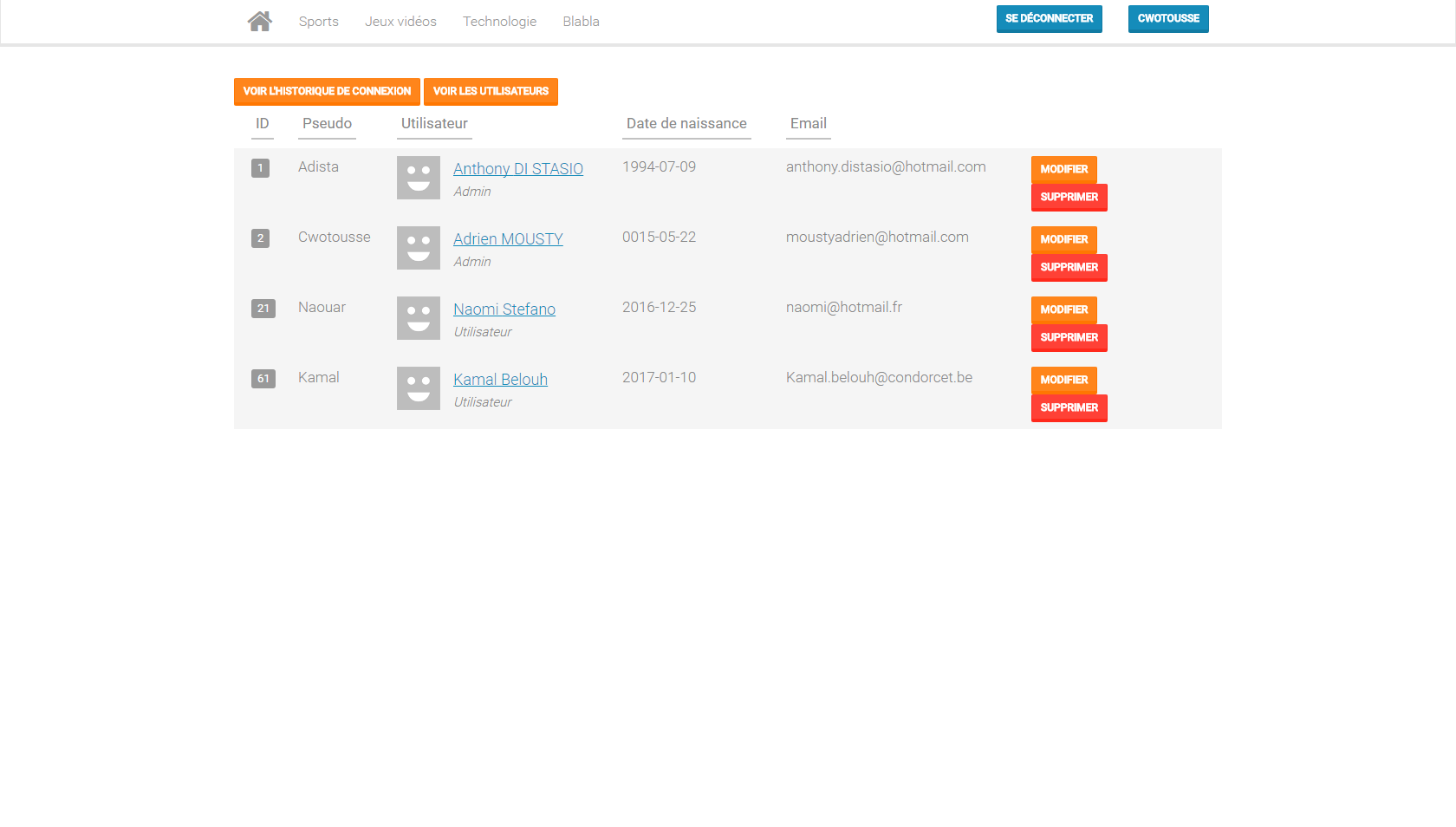


Figure Ecran : panneau de configuration

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.

# 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.

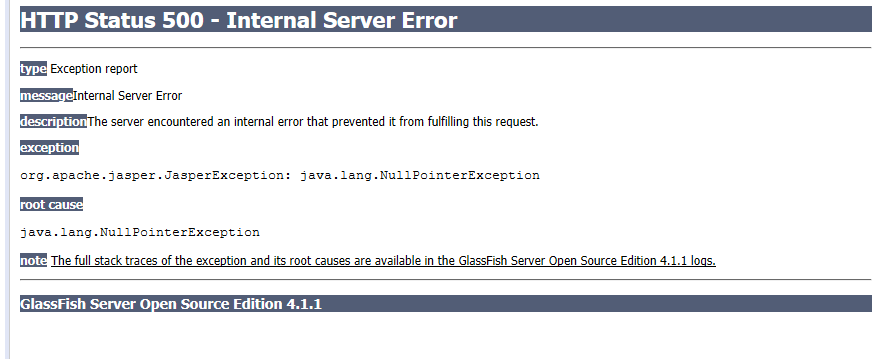


Figure Erreur 500

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 :

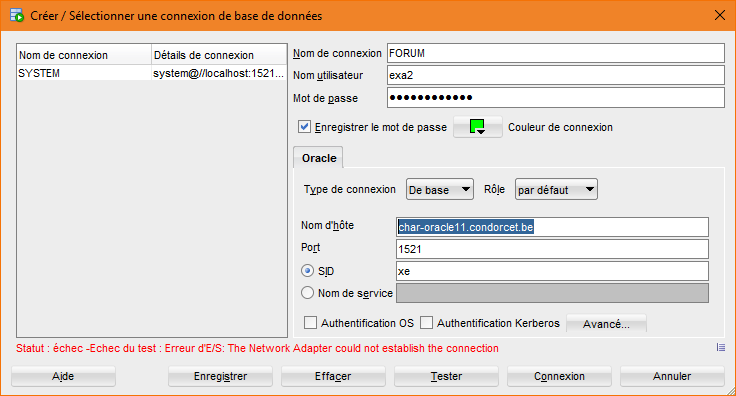


Figure Erreur ES

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.

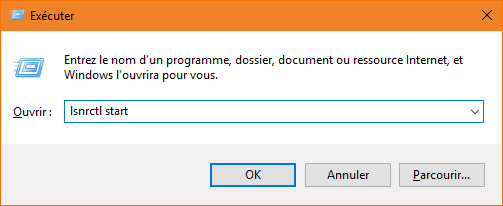


Figure 50 listener process (service) is not running

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 :

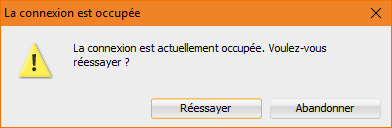


Figure Connexion occupée

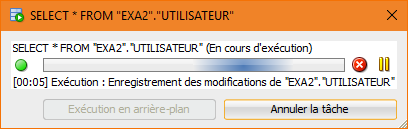


Figure Query en attente

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.

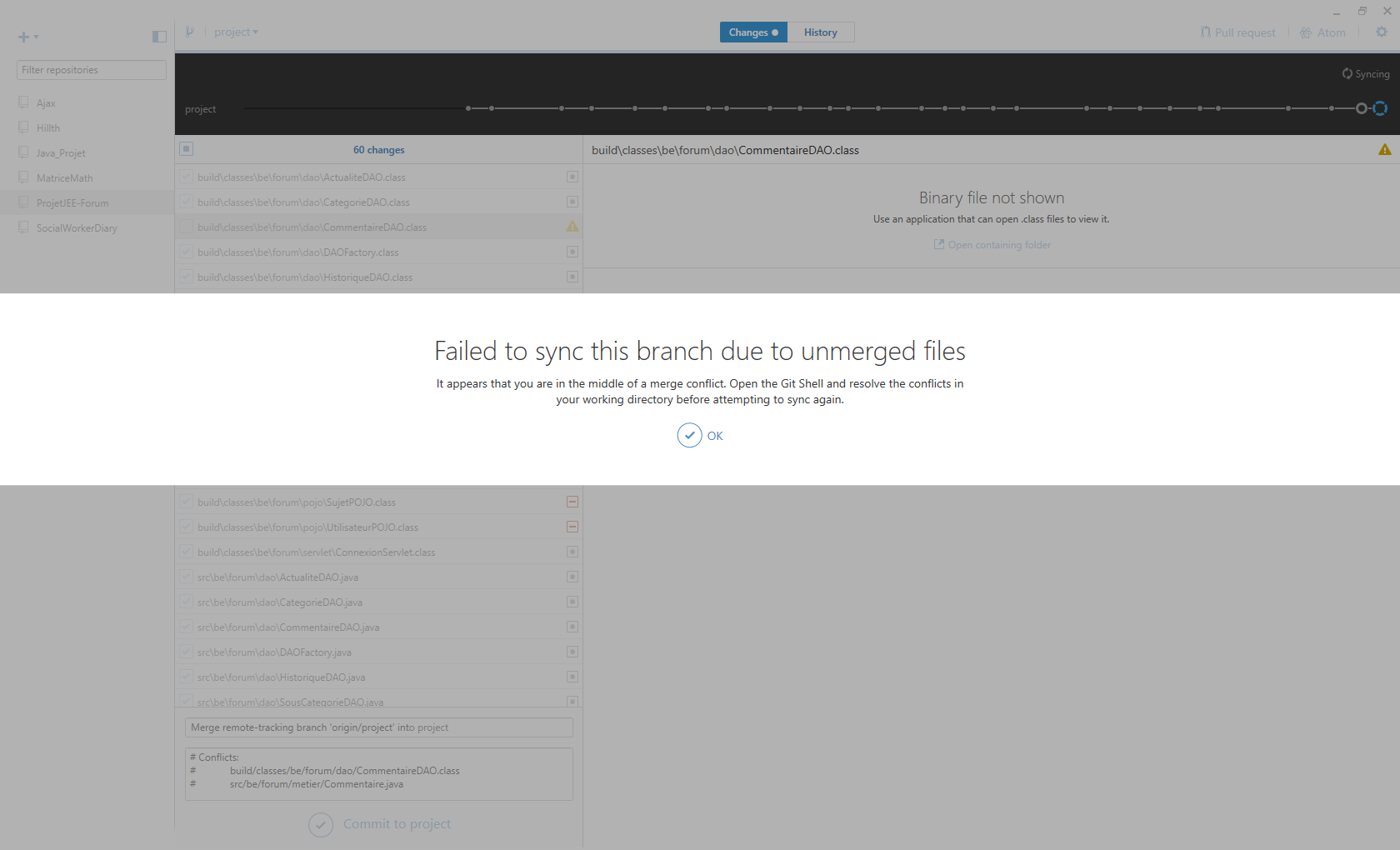


Figure Erreur GitHub

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](#_Toc471933020)

[Figure 2 Use Case 7](#_Toc471933021)

[Figure 3 Diagramme de classe 8](#_Toc471933022)

[Figure 4 Schéma de la base de données 9](file:///C:\Users\Anthony\workspaceJEE\ProjetJEE-Forum\ProjetJEE-Forum-PLSQL.docx#_Toc471933023)

[Figure 5 Connexion Oracle 10](#_Toc471933024)

[Figure 7 table Utilisateur 10](#_Toc471933025)

[Figure 8 Foreign key Historique Utilisateur 11](#_Toc471933026)

[Figure 9 Les séquences 12](#_Toc471933027)

[Figure 10 Insert utilisateur 13](#_Toc471933028)

[Figure 11 Delete utilisateur 14](#_Toc471933029)

[Figure 12 Update utilisateur 15](#_Toc471933030)

[Figure 13 Select Utilisateur 16](#_Toc471933031)

[Figure 14 Get List Utilisateur 17](#_Toc471933032)

[Figure 15 NO\_DATA\_FOUND 18](#_Toc471933033)

[Figure 16 Exception anonyme 19](#_Toc471933034)

[Figure 17 exemple fonction stockée (1) 20](#_Toc471933035)

[Figure 18 Exemple fonction stockée (2) 20](#_Toc471933036)

[Figure 19 Package 1 22](#_Toc471933037)

[Figure 20 Exemple déclencheur 23](#_Toc471933038)

[Figure 24 Ecran : connexion 24](#_Toc471933039)

[Figure 25 Ecran : profil 24](#_Toc471933040)

[Figure 26 Ecran : home 25](#_Toc471933041)

[Figure 27 Ecran : sous-catégorie 25](#_Toc471933042)

[Figure 28 Ecran : sujets 26](#_Toc471933043)

[Figure 29 Ecran : erreur 26](#_Toc471933044)

[Figure 30 Ecran : affichage commentaire 27](#_Toc471933045)

[Figure 31 Ecran : poster un commentaire 27](#_Toc471933046)

[Figure 32 Ecran : historique de connexion 28](#_Toc471933047)

[Figure 33 Ecran : panneau de configuration 28](#_Toc471933048)

[Figure 48 Erreur 500 29](#_Toc471933049)

[Figure 49 Erreur ES 29](#_Toc471933050)

[Figure 50 listener process (service) is not running 30](#_Toc471933051)

[Figure 51 Connexion occupée 30](#_Toc471933052)

[Figure 52 Query en attente 31](#_Toc471933053)

[Figure 53 Erreur GitHub 31](#_Toc471933054)

# 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)