



FORUM ACTIF

Projet JEE

Adrien MOUSTY

moustyadrien@hotmail.com

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

PL/SQL

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.

SELECT

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

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

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

```
CREATE OR REPLACE PROCEDURE SELECTUTILISATEUR(
  -- PARAMETRE ENTRANT
  P_IDUTILISATEUR IN UTILISATEUR.IDUTILISATEUR%TYPE,
  -- LES PARAMETRES SORTANTS
  O_PSEUDO OUT UTILISATEUR.PSEUDO%TYPE,
  O_MOTDEPASSE OUT UTILISATEUR.MOTDEPASSE%TYPE,
  O_NOM OUT UTILISATEUR.NOM%TYPE,
  O_PRENOM OUT UTILISATEUR.PRENOM%TYPE,
  O_DATENAISSANCE OUT UTILISATEUR.DATENAISSANCE%TYPE,
  O_TYPE OUT UTILISATEUR.TYPEUTILISATEUR%TYPE,
  O_MAIL OUT UTILISATEUR.MAIL%TYPE)
IS
BEGIN
  SELECT PSEUDO, MOTDEPASSE, NOM, PRENOM, DATENAISSANCE, TYPEUTILISATEUR, MAIL
  INTO O_PSEUDO, O_MOTDEPASSE, O_NOM, O_PRENOM, O_DATENAISSANCE, O_TYPE, O_MAIL
  FROM UTILISATEUR WHERE IDUTILISATEUR = P_IDUTILISATEUR;

  EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Pas de donnees trouvee. (UTILISATEUR)');
END;
/
```

Figure 1 Select Utilisateur

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.

```
CREATE OR REPLACE PROCEDURE SELECTUTILISATEUR(
  -- PARAMETRE ENTRANT
  P_IDUTILISATEUR IN UTILISATEUR.IDUTILISATEUR%TYPE,
  -- LES PARAMETRES SORTANTS
  O_PSEUDO OUT UTILISATEUR.PSEUDO%TYPE,
  O_MOTDEPASSE OUT UTILISATEUR.MOTDEPASSE%TYPE,
  O_NOM OUT UTILISATEUR.NOM%TYPE,
  O_PRENOM OUT UTILISATEUR.PRENOM%TYPE,
  O_DATENAISSANCE OUT UTILISATEUR.DATENAISSANCE%TYPE,
  O_TYPE OUT UTILISATEUR.TYPEUTILISATEUR%TYPE,
  O_MAIL OUT UTILISATEUR.MAIL%TYPE)
IS
BEGIN
  SELECT PSEUDO, MOTDEPASSE, NOM, PRENOM, DATENAISSANCE, TYPEUTILISATEUR, MAIL
  INTO O_PSEUDO, O_MOTDEPASSE, O_NOM, O_PRENOM, O_DATENAISSANCE, O_TYPE, O_MAIL
  FROM UTILISATEUR WHERE IDUTILISATEUR = P_IDUTILISATEUR;

  EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Pas de donnees trouvee. (UTILISATEUR)');
END;
```

Figure 2 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 ».

FONCTION STOCKÉE

EXEMPLE

```
-- Appelle les méthodes qui comptent le nombre de sujet et de commentaires
-- Calcule la moyenne de posts par sujets créé
create or replace FUNCTION NombreMessagesMoyenParSujet RETURN number
IS moyenneMessages number := 0;
BEGIN
    moyenneMessages := calculerNombreCommTotal/calculerNombreSujetsTotal;
    return moyenneMessages;
END NombreMessagesMoyenParSujet;
/
```

Figure 3 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.

```
-- Fonction stockée déjà créée précédemment mais ici utilisation du bulk
create or replace FUNCTION getAdminsBulkCollect RETURN tab_utilisateur
IS
    -- tableau d'administrateurs
    l_admin_tab tab_utilisateur := tab_utilisateur();
    BEGIN
        -- Il faut convertir le tout en type_utilisateur
        -- Avant de le placer dans un bulk collect
        SELECT type_utilisateur(
            IDUTILISATEUR, PSEUDO, MOTDEPASSE, NOM,
            PRENOM, DATENAISSANCE, TYPEUTILISATEUR, MAIL)
        BULK COLLECT INTO l_admin_tab
        FROM UTILISATEUR
        WHERE TYPEUTILISATEUR = package type utilisateur.type admin;

        RETURN l_admin_tab;
    END getAdminsBulkCollect;
/
```

Figure 4 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.

DÉTAILS DE CERTAINES PARTIE DU CODE

LAMBDA EXPRESSION

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.getEmail().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 « Catégorie » 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, MÉTHODE « CREATE() »

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

```
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();
            }
        }
    }
}
```

JSP ET SERVLETS

EXEMPLE DE 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.

```
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.

```

<error-page>
  <exception-type>java.lang.Throwable</exception-type>
  <location>/erreur.jsp</location>
</error-page>
<error-page>
  <exception-type>java.lang.NumberFormatException</exception-type>
  <location>/erreur.jsp</location>
</error-page>
<error-page>
  <error-code>404</error-code>
  <location>/erreur.jsp</location>
</error-page>

```

Figure 5 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.

```

1 <%@page isErrorPage="true"%>

```

Ensuite, il n'y a plus qu'à 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.

```

if (listCategorie.isEmpty()){
    request.setAttribute("error_message", "Il n'y a pas de post pour ce sujet.");
    RequestDispatcher dispatcher = request.getRequestDispatcher("/VUE/erreur.jsp");
    dispatcher.forward(request, response);
    response.setContentType("text/html");
} else {

```

Figure 6 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

SUJET

Sports Jeux vidéos Technologie Blabla

SE DÉCONNECTER CWTOUTOUSSE ADMINISTRATION

Programmation

Choisissez le sujet que vous souhaitez consulter.

Chrome

	Cwtoutousse	Total posts: 3 Nouveau
[Ubuntu] Mon terminal ne peut plus exécuter	Auteur	140
Si ce sujet vous intéresse, cliquez dessus!	10	Posts
	Date	
Ecole 42, je répond à vos questions!	Cwtoutousse	2017-01- 140
Si ce sujet vous intéresse, cliquez dessus!	Auteur	10 Posts
	Date	
[JAVA] problème d'import de ressources	Cwtoutousse	2017-01- 140
Si ce sujet vous intéresse, cliquez dessus!	Auteur	10 Posts
	Date	

Figure 7 Ecran : sujets

Lorsque l'on clique sur une sous-catégorie tous les sujets s'y référant sont affichés. Ici nous avons cliqué sur « programmation ». Il s'avère qu'il y a déjà 3 sujets créés à propos de la programmation.

ERREUR

Sports Jeux vidéos Technologie Blabla

SE DÉCONNECTER CWTOUTOUSSE

Une erreur est survenue

Un élément n'a pas été correctement rempli. [Revenir à l'accueil](#)

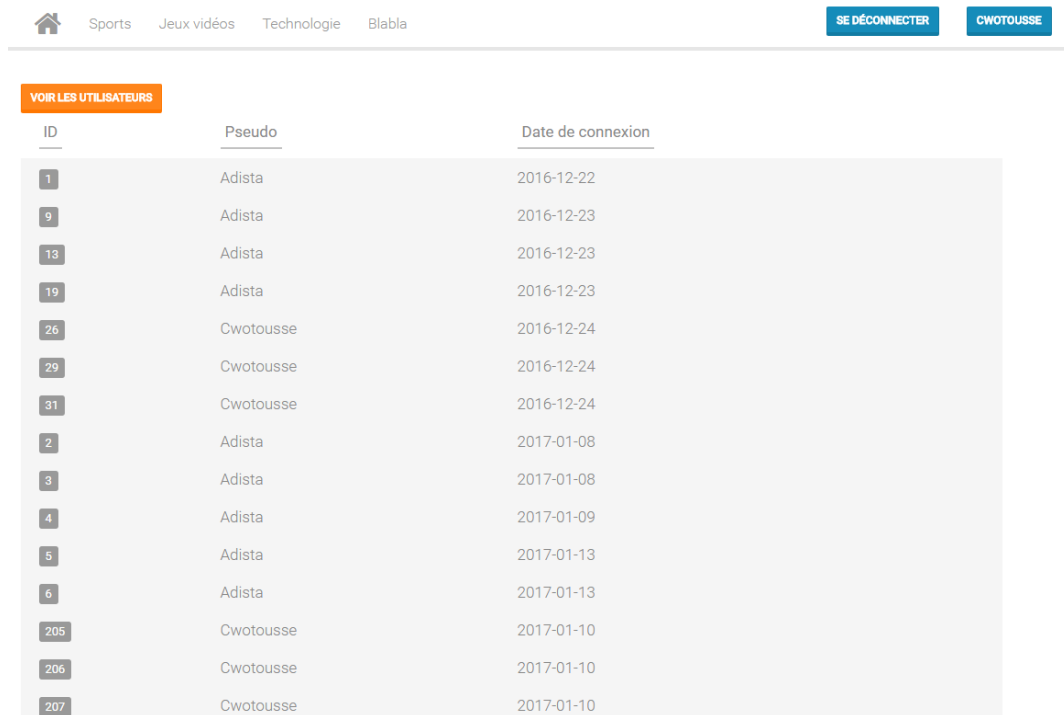


© 2017 Anthony & Adrien, Inc.

Figure 8 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.

HISTORIQUE DE CONNEXION

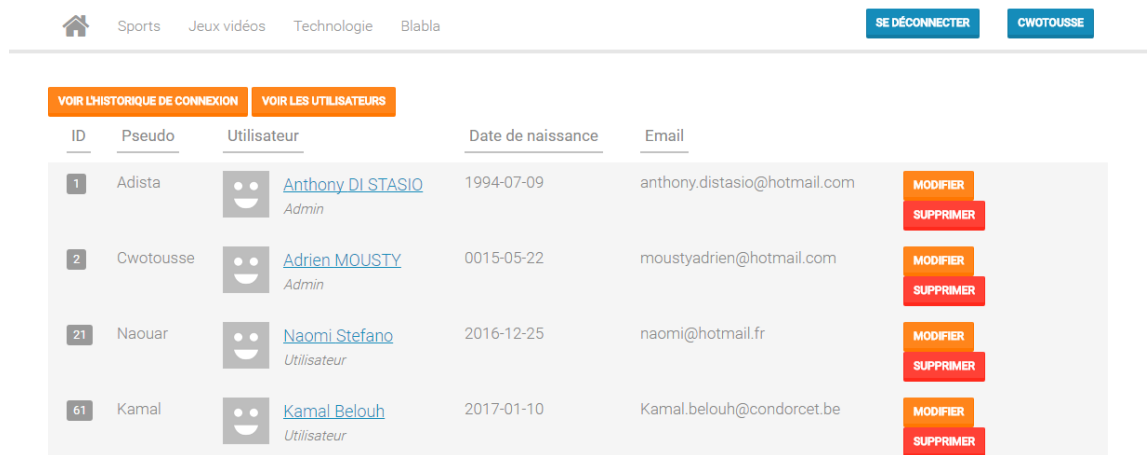


ID	Pseudo	Date de connexion
1	Adista	2016-12-22
9	Adista	2016-12-23
13	Adista	2016-12-23
19	Adista	2016-12-23
26	Cwotousse	2016-12-24
29	Cwotousse	2016-12-24
31	Cwotousse	2016-12-24
2	Adista	2017-01-08
3	Adista	2017-01-08
4	Adista	2017-01-09
5	Adista	2017-01-13
6	Adista	2017-01-13
205	Cwotousse	2017-01-10
206	Cwotousse	2017-01-10
207	Cwotousse	2017-01-10

Figure 9 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







ID	Pseudo	Utilisateur	Date de naissance	Email	
1	Adista	 Anthony DI STASIO Admin	1994-07-09	anthony.distasio@hotmail.com	MODIFIER SUPPRIMER
2	Cwotousse	 Adrien MOUSTY Admin	0015-05-22	moustyadrien@hotmail.com	MODIFIER SUPPRIMER
21	Naouar	 Naomi Stefano Utilisateur	2016-12-25	naomi@hotmail.fr	MODIFIER SUPPRIMER
61	Kamal	 Kamal Belouh Utilisateur	2017-01-10	Kamal.belouh@condorcet.be	MODIFIER SUPPRIMER

Figure 10 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.

JSTL

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 équivalent respectivement à un « foreach » et un « switch/case » en java.

FOREACH

```
<c:forEach items="${listeCommentaire}" var="commentaire">
  <section class="container" id="elem-comment"> <section
    class="row clearfix"> <section class="col-md-12 column">
  <div class="row clearfix">
    <div class="col-md-12 column">
      <div class="panel panel-default">
        <div class="panel-heading">
          <section class="panel-title"> <time class="pull-right">
          <i class="fa fa-calendar"></i>
          <div id="date-creation-sujet">
            ${commentaire.getSujet().getDateSujet()}</div>
```

Figure 11 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 11 Foreach que nous affichons la date du sujet issu de la liste à chaque fois.

CHOOSE ... WHEN

```
<!-- La personne connectée ne pourra modifier que ses posts -->
<!-- Si c'est un admin il peut tout modifier et supprimer -->
<c:choose>
  <c:when test="${commentaire.getUtilisateur().getPseudo() == utilisateur.getPseudo() || utilisateur.getType() == 'Admin'}">
    <section>
      <a href="#" class="btn btn-primary launch-modal" id="edit" href="#" data-modal-id="modal-modify-${commentaire.getID()}">
        Modifier
      <i class="fa fa-edit"></i>
    </a>
      <a href="#" class="btn btn-primary launch-modal" id="delete" href="#" data-modal-id="modal-delete-${commentaire.getID()}">
        Supprimer
      <i class="fa fa-trash-o"></i>
    </a>
    </section>
  </c:when>
</c:choose>
```

Figure 12 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.

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

RÉSUMÉ

- Projet effectué à deux
- Mise en place du modèle MVC
- Langages utilisés : Java, JEE, JSTL, PLSQL, JavaScript, HTML, CSS, jQuery
- Utilisation d'une base de données distante, Bootstrap, Glassfish 4.0 ainsi que Git.
- Ce travail fut effectué en collaboration complète, nous avons beaucoup travaillé ensemble, mais lorsque nous ne le faisons pas cela ne posait pas de problème grâce à l'outil GitHub.
- Note finale : 15/20