

Projet Réseaux Middlewares : I.K.Y.A

Equipe : Groupe 5

Jean-Damien THEVENOUX

Abdelkrim AHMED-BACHA

Nacer BOUHERROU

Thibaut BESACIER

Olivier MARIN

Professeur : Mr. Jules Satin-Chevalier

I. Introduction.

a. Enjeux pédagogiques.

Ce projet a été très instructif pour l'équipe puisqu'il visait à la manipulation de plusieurs structures propres à Java. Nous avons donc utilisé « [Maven](#) » pour l'inclusion automatique de bibliothèque au sein du notre projet. Le « [framework openJPA](#) » afin de persister les données de notre application et les « [EJB 3.0](#) » pour une gestion de l'application entre un client et un serveur. Le projet est bien documenté grâce à la « [JavaDoc](#) ».

b. Notre réalisation.

Au sein de ce compte rendu, nous allons expliquer notre démarche afin de remplir le cahier des charges et structurer notre application en y incluant les outils conseillés (openJPA, Maven et les EJB 3). Nous commencerons par un petit tutoriel pour déployer l'application dans les meilleures conditions.

II. Settings pour le déploiement de l'application.

Notre projet a été versionné en utilisant l'outil « [Git](#) » et se trouve à l'adresse suivante :

<https://github.com/telecom-se/ikya-grp5>

a. Configuration de la base de données.

Pour faire des tests sur votre ordinateur, il faut disposer d'une application (PHPMyAdmin ou encore MySQL Launcher) dispensant [MySQL](#). Dans un premier temps, créer une base de données « [ikya](#) ». Puis dans un second temps, veuillez peupler votre base de données avec le fichier nommé « [ikya.sql](#) » présent dans le répertoire Git.

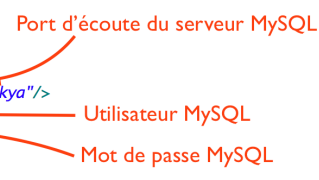
Après avoir téléchargé l'application, veuillez l'importer en tant que [projet maven](#) sous eclipse. Avant toute opération de déploiement de l'application, soyez certain de disposer d'une [JRE1.8.0_20](#) sur [Eclipse](#) et d'un serveur [GlassFish 4](#).

b. Configuration du fichier « persistence.xml ».

Vient ensuite la modification du fichier de persistance. Le fichier « persistence.xml » se trouve sous « META-INF » dans le répertoire « src/main/ressources ». Veuillez renseigner le pseudonyme, le mot de passe de votre propre application dispensant MySQL et surtout le port sur lequel votre MySQL écoute les requêtes SQL (ici, c'est le port 8889)

Exemple de fichier « persistence.xml » :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <persistence xmlns="http://java.sun.com/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
6     http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
7   version="2.0">
8   <persistence-unit name="ikya_project">
9     <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
10
11     <class>org.ikya.entities.User</class>
12     <class>org.ikya.entities.Contact</class>
13     <class>org.ikya.entities.Challenge</class>
14     <class>org.ikya.entities.Messagerie</class>
15     <class>org.ikya.entities.Message</class>
16
17     <validation-mode>NONE</validation-mode>
18
19     <properties>
20
21       <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
22       <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:8889/ikya"/>
23       <property name="javax.persistence.jdbc.user" value="root"/>
24       <property name="javax.persistence.jdbc.password" value="root"/>
25       <property name="jdbc.SynchronizeMappings" value="buildSchema(ForeignKeys=true)"/>
26       <property name="openjpa.RuntimeUnenhancedClasses" value="supported"/>
27
28     </properties>
29   </persistence-unit>
30 </persistence>
```



c. Déploiement de l'application.

Vous pouvez maintenant ajouter votre projet au serveur « GlassFish 4 » avant de le déployer. Si vous rencontrez des problèmes. Assurez-vous de n'avoir raté aucune des étapes de la démarche présentée ci dessus ou sinon rafraichissez votre projet.

d. Point d'entrée de l'application.

Une fois le projet correctement déployé, rendez-vous à l'url suivante :

<http://localhost:8080/ikya/login.jsp>

Remarque :

Suivant la gamme de votre PC, il arrive qu'il faille installer les jars liés à « openJPA » et au connecteur « JDBC » dans le répertoire « lib » sous « domain1 » de votre serveur « GlassFish ».

IV. Architecture globale de l'application.

a. Choix d'implémentation.

Concernant l'architecture globale de l'application, nous avons choisi d'implémenter un client web en utilisant la norme JEE (servlet, jsp) et « [bootstrap](#) ». L'application a pris plus de temps à implémenter mais le rendu et l'utilisation sont largement plus satisfaisant surtout lorsqu'il s'agit de constater le bon fonctionnement d'un module.

b. Persistance des entités avec openJPA.

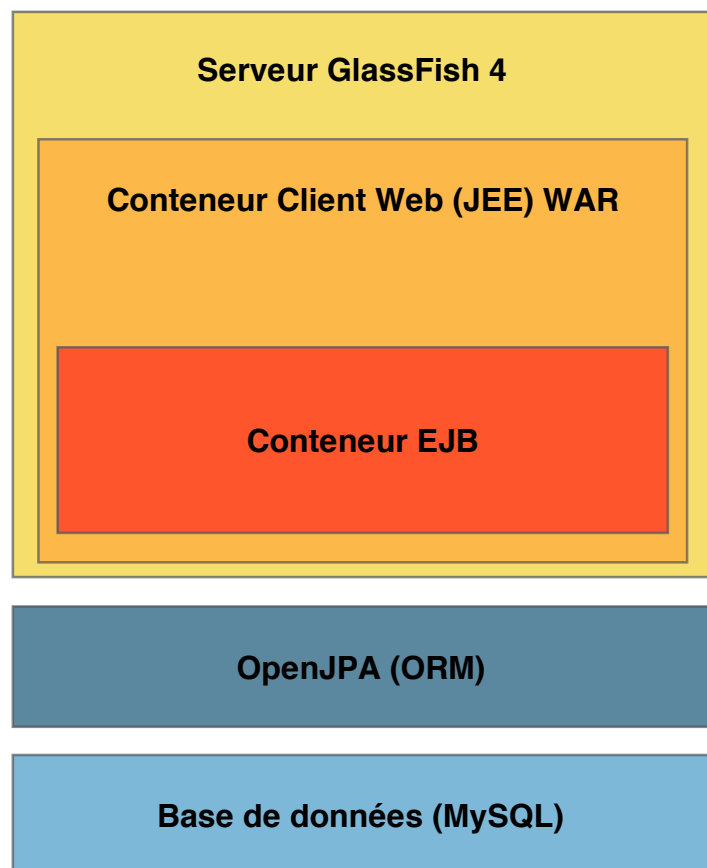
Concernant la persistance, nous nous sommes attachés à utiliser l'implémentation « [openJPA](#) » de la norme « [JPA](#) » fournie par JAVA EE. Par ailleurs, toutes les « [transactions openJPA](#) », entre le programme et la base de données, sont implémentées dans des DAO (EJB de type session) incluses dans le package « [org.ikay.dao](#) ». Enfin, les accès à la base de données sont fournis par la classe « [OpenJPAUtils.java](#) » présente dans le package « [org.ikya.utils](#) ».

c. Structure globale de l'application.

Les EJB sont instanciés au sein d'un conteneur qui leur est propre. De cette façon, ils peuvent être injectés dans n'importe quelle classe ou page jsp de l'application. Cette dernière est organisée sous forme de WAR. C'est sous cette forme qu'elle peut être déployée sur le server GlassFish autant qu'application.

Les DAO, implémentées sous forme d'EJB sessions stateless, se servent des méthodes de persistance fournies par openJPA pour sauvegarder les données.

De cette façon, c'est le code métier qui impose sa logique de sauvegarde à la base de données et non l'inverse comme c'est le cas en l'absence d'ORM de persistance.



d. Structure globale de la base de données.

Remarque préliminaire :

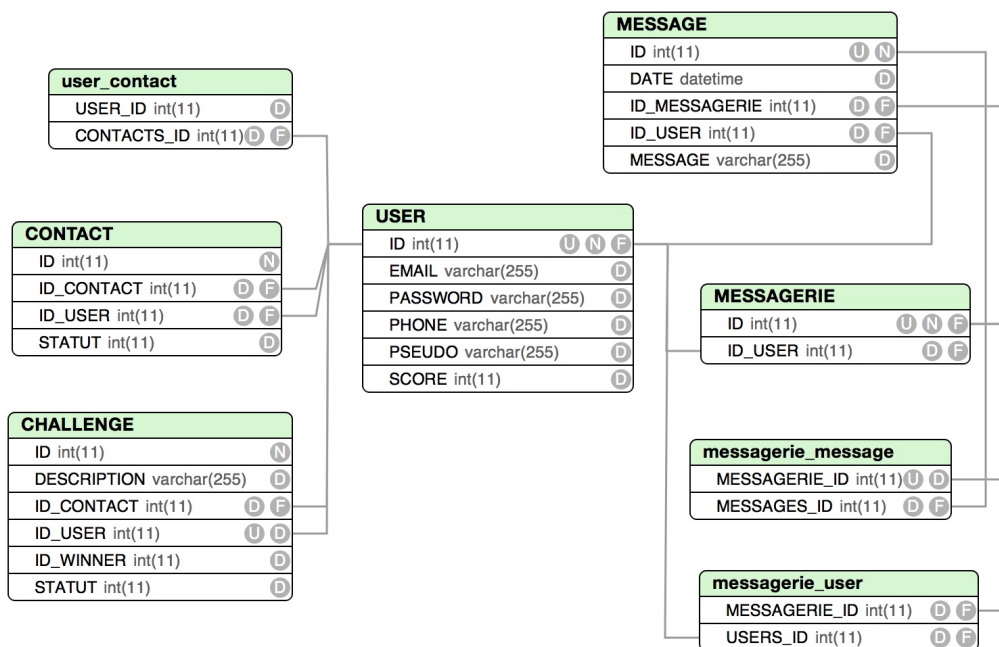
Le type des champs d'une table est annoté par un unique caractère. Veuillez prendre connaissance du tableau suivant avant de prendre connaissance de la structure de la base de données :

Annotation	Signification	Type
F	Foreign Key	Clé primaire d'une table.
N	Not Null	Champs ne pouvant pas prendre de valeur nulle.
D	Decimal	Type d'un champ de type numérique.
U	Unique	La valeur de ce champ est unique pour toutes les entrées de la table.

Introduction à la structure de la base de données :

La table des utilisateurs « **USER** » est la table centrale de la structure. Les autres tables disposent également d'un identifiant unique mais sont rattachées à un utilisateur par le biais d'une clé étrangère.

Vue globale de la structure de la base de données :



Observation concernant la base de données :

On peut regrouper les tables par modules. On s'aperçoit que les tables « **USER** », « **CONTACT** » et « **USER_CONTACT** » permettent de gérer la création d'utilisateurs et la gestion de leurs contacts. On notera que l'ajout d'un contact est bien géré en base de données grâce à la présence d'un champ « **STATUT** » dans la table des contacts « **CONTACT** ».

La gestion des challenges est assurée par les tables « **USER** », « **CONTACT** », « **USER_CONTACT** » et « **CHALLENGE** ». L'utilisation des 3 premières tables permet de s'assurer que la demande est bien envoyée à un contact et non à un utilisateur quelconque de l'application. Le cycle de vie d'un challenge (vue, accepté, relevé et partage du score) est assuré par un champ « **STATUT** » présent dans la table des challenges « **CHALLENGES** ».

La messagerie concerne les classes « **MESSAGERIE** », « **MESSAGERIE_MESSAGE** », « **MESSAGERIE_USER** », « **MESSAGE** », « **CONTACT** » et « **USER** ». Ce module permet le partage d'une conversation entre plusieurs utilisateurs (**bonus**). La structure de la messagerie est détaillée dans la partie qui lui est consacrée.

Remarque finale concernant l'architecture :

Toutes les constantes liées au bon fonctionnement de l'application sont répertoriées dans la classe « **Constants.java** » présente sous le package « **org.ikya.constants** ». Chaque entité repose sur un bloc de constantes bien délimité afin de faciliter l'implémentation de l'application.

V. Gestion des utilisateurs et de leurs contacts.

Dans cette partie, on s'intéressera à l'implémentation des utilisateurs ainsi que leurs liens avec d'autres utilisateurs (contacts).

a. Abstraction d'accès à la base de données (DAO)

- **create(User) : void**
- **update(User) : void**
- **updateScoreById(Integer, Integer) : void**
- **findByPseudo(String) : User**
- **findByUserID(Integer) : User**
- **findByEmail(String) : User**
- **authenticateUser(String, String) : Boolean**
- **ifPseudoExist(String) : Boolean**
- **ifMailExist(String) : Boolean**
- **getAllContactByUserID(Integer) : Set<Contact>**
- **ifContactExist(Contact) : Boolean**
- **deleteContact(Contact) : void**
- **findContactById(Integer) : Contact**
- **changeContactStatut(Integer, Integer) : void**
- **getAll() : List<User>**
- **getAll(Integer) : List<User>**
- **delete(User) : void**

b. Traitement d'une demande de contact

Une demande de contact a 4 états :

- Demande : Non vue / Vue
- Demande : Acceptée / Refusée

Tout d'abord, l'utilisateur a la possibilité d'inviter qui il souhaite dans sa liste de contact.

Supposons qu'en tant que Sarah, on demande Laure en contact:

#ID	Login	Ajouter
1	Laure	+
2	Kev	+
3	James	+

LISTE DES UTILISATEURS

La demande de contact va alors passer en attente de validation par l'autre partie :

[Demande à : Laure](#)Non vue

DEMANDE COTE SARAH (NON VUE)

Dès que Laure accédera à ses contacts, elle aura la possibilité de valider ou non cette demande et on notifie en même temps Sarah, que Laure a bien reçu sa demande (Passage Etat Vue).

[Demande de : Sarah](#)Vue

Accepter Refuser

DEMANDE COTE LAURE



[Demande à : Laure](#)Vue

DEMANDE COTE SARAH (VUE)

- Si Laure refuse la demande d'ajout, alors la requête est supprimée des deux listes de contacts.
- Si Laure accepte, alors Sarah apparait désormais dans sa liste de contact avec la possibilité de dialoguer.

#	Pseudo	Email	Score	Messagerie	Supprimer
1	Sarah	sarah@gmail.com	0		

CONTACT COTE LAURE

#	Pseudo	Email	Score	Messagerie	Supprimer
1	Laure	laure@gmail.com	0		

[CONTACT COTE SARAH](#)

VI. Gestion des Challenges et de leurs partages sur Twitter (bonus)

Dans cette partie, on s'intéressera à l'implémentation des challenges, leur gestion et interactions

a. Abstraction d'accès à la base de données (DAO)

- create(Challenge) : void
- update(Challenge) : void
- updateWinner(Challenge, Integer) : void
- updateStatut(Challenge, Integer) : void
- setScore(Challenge) : void
- findById(Integer) : Challenge
- findByUserID(Integer) : List<Challenge>
- ifChallengeExist(Challenge) : Boolean
- deleteChallenge(Challenge) : void

b. Création d'un challenge

L'utilisateur a la possibilité de défier un de ses contacts en créant un nouveau challenge, il lui suffit de choisir le contact et de donner la description du défi.

Lancer un défi à un contact !

Sarah
Search !

Partie d'échec

Envoyer

Une fois un challenge créé il peut passer par 5 états :

- en attente de validation du contact défié : ON_HOLD
- accepté par le contact, en attente du score : ACCEPTED
- refusé par le contact : REFUSED
- jeu terminé, score en attente de validation : DONE
- score validé, challenge terminé : CHECKED

On peut vérifier l'état de tous les challenges réalisés par l'utilisateur connecté grâce à la liste des challenges. Celle-ci permet également d'interagir avec les challenges afin de les faire changer d'états.

Supposons que Laure lance un défi à Sarah, voici les différents cas possibles :

description : Partie d'échec

en attente

En attente de validation !

CHALLENGE COTE LAURE : ON_HOLD

description : Partie d'échec

Accepter Refuser

en attente

CHALLENGE COTE SARAH : ON_HOLD

Maintenant Sarah accepte le challenge, Laure doit donner le résultat du défi :

description : Partie d'échec

J'ai Gagné Match Nul J'ai Perdu

accepté

CHALLENGE COTE LAURE : ACCEPTED

description : Partie d'échec

accepté

Résultat à venir ...

CHALLENGE COTE SARAH : ACCEPTED

Laure indique qu'elle a gagné le défi, Sarah doit valider ce résultat :

description : Partie d'échec

terminé

Vous avez gagné !

Validation du challenge en cours...

description : Partie d'échec

J'ai Gagné Match Nul J'ai Perdu

accepté

CHALLENGE COTE LAURE : DONE

description : Partie d'échec

terminé

Vous avez perdu ...

Accepter Refuser

CHALLENGE COTE SARAH : DONE

Lorsque Sarah accepte ce résultat, le challenge passe alors à l'état CHECKED, les scores sont actualisés pour les deux personnes en fonction du gagnant et chacun garde une trace de ce challenge dans son historique :

description : Partie d'échec

validé

Challenge terminé, vous avez gagné ! [Share](#)

CHALLENGE COTE LAURE : CHECKED

description : Partie d'échec

validé

Challenge terminé, vous avez perdu ... [Share](#)

CHALLENGE COTE SARAH : CHECKED

Les utilisateurs peuvent aussi partager leurs résultats sur Twitter !

VII. Messagerie instantanée.

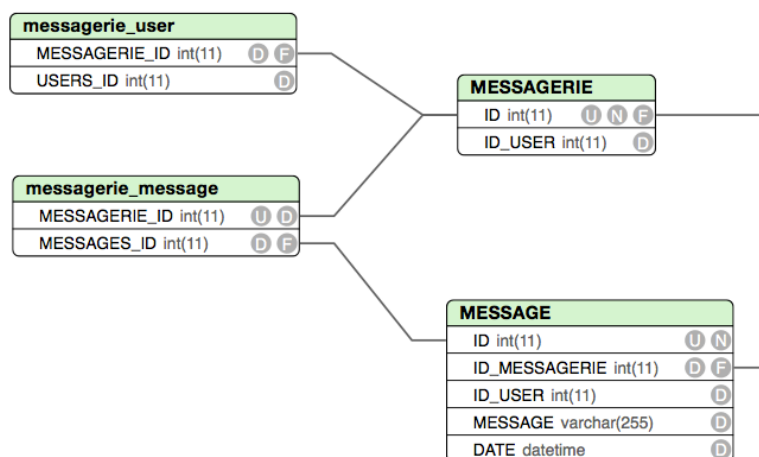
Pour la réalisation de la messagerie instantanée, nous avons créé une classe « Messagerie » contenant l'ID de l'utilisateur créant la messagerie.

La classe « Messagerie » se compose d'une collection d'utilisateur, ceci correspondant aux utilisateurs présents dans la messagerie et d'une collection de messages.

Les messages se composent :


- ⇒ ID_MESSAGERIE : ID de la messagerie pour laquelle le message est destiné
- ⇒ ID_USERS : ID du user émettant le message
- ⇒ MESSAGE : Contenu du message
- ⇒ DATE : Date d'émission du message

Figure 1 : Schéma de la BDD (Gestion de la messagerie)



Affichage Messagerie (discussion entre Laure et James):

Messagerie :

 James Laure

Laure
salut

James
hello

Type your message here...

Submit

Possibilité d'ajouter des contacts à la conversation (Sarah) :

Messagerie :

 James Sarah Laure

Laure
salut

James
hello

Laure
heyyyyyy !!!!

Sarah
i'm in

Type your message here...

Submit

Refresh

Ajouter des amis

Pour afficher les messages de manière instantanée, la <div> contenant les messages est appelée toute les 1000ms par une fonction Ajax qui permet de la remplir. Ceci permet aux utilisateurs de recevoir les messages envoyés de manière instantanée.

Figure 2 : Fonction Ajax pour remplir la div des messages

```
var interval = 1000;
function doAjax() {

    var mge = $('#btn-input').val();
    var idMessagerie = "<c:out value='${idMessagerie}' />";

    $.ajax({
        type: "POST",
        url: "RefreshServlet",
        data: { idMessagerie : idMessagerie},
        success: function( result ){
            console.log(result);

            $( "div.panel-body" ).html(result);

        },
        complete: function (data) {
            // Schedule the next
            setTimeout(doAjax, interval);
        }
    })
}
```

VIII. Conclusion & remerciement.

Pour ce qui est des outils recommandés pour la réalisation du projet, nous avons su les intégrer et tirer partie de leurs fonctionnalités.

Git permet réellement de travailler en équipe dans le sens où il gère non seulement le visionnement mais nous donne aussi la possibilité de faire des tests sur des branches parallèles au projet.

Les EJB 3.0 se sont révélés réellement très maniables puisque d'une fois implémentés ; ils peuvent être utilisés partout dans l'application par une simple injection.

OpenJPA nous a permis d'imposer notre logique métier à la base de données. Nous avons rencontré beaucoup moins d'obstacles au moment d'implémenter nos DAO.

A ce propos voici la liste des points du cahier que nous avons réussi à implémenter :

Les fonctionnalités implémentées dans le projet :

- ✓ Gestion des utilisateurs
- ✓ Inscription d'un utilisateur
- ✓ Authentification d'un utilisateur
- ✓ Désinscription d'un utilisateur (authentifié) logout et suppression de compte
- ✓ Gestion des contacts
- ✓ Recherche d'utilisateurs
- ✓ Demande d'ajout d'un utilisateur aux contacts
- ✓ Acceptation/refus d'une demande d'ajout
- ✓ Suppression d'un utilisateur des contacts

- ✓ Gestion des défis
- ✓ Envoie d'une demande de défi à un contact (avec AJAX)
- ✓ Acceptation/refus d'un défi
- ✓ Notification (asynchrone) avec persistance en BDD
- ✓ Saisie du résultat de la partie
- ✓ Validation du résultat par les membres du défi
- ✓ Classement des utilisateurs
- ✓ Messagerie instantanée (avec AJAX)
- ✓ Envoi/réception de messages aux contacts
- ✓ Notification (asynchrone) avec persistance en BDD

Options :

- ✓ Conversations à 3+ utilisateurs
- ✓ Publication des résultats sur les réseaux sociaux (Twitter)
- ✓ Visualisation de l'historique des défis (LeaderBoard)
 - Utiliser le système de classement ELO
 - Défis à 3+ utilisateurs
 - Notification des utilisateurs par mail / sms
 - Récupération d'informations sur les joueurs, les jeux, grâce à l'API Steam

L'équipe remercie Mr Jules Satin Chevalier pour les supports de cours, les TP ainsi que l'aide qu'il nous a apporté tout au long du projet.