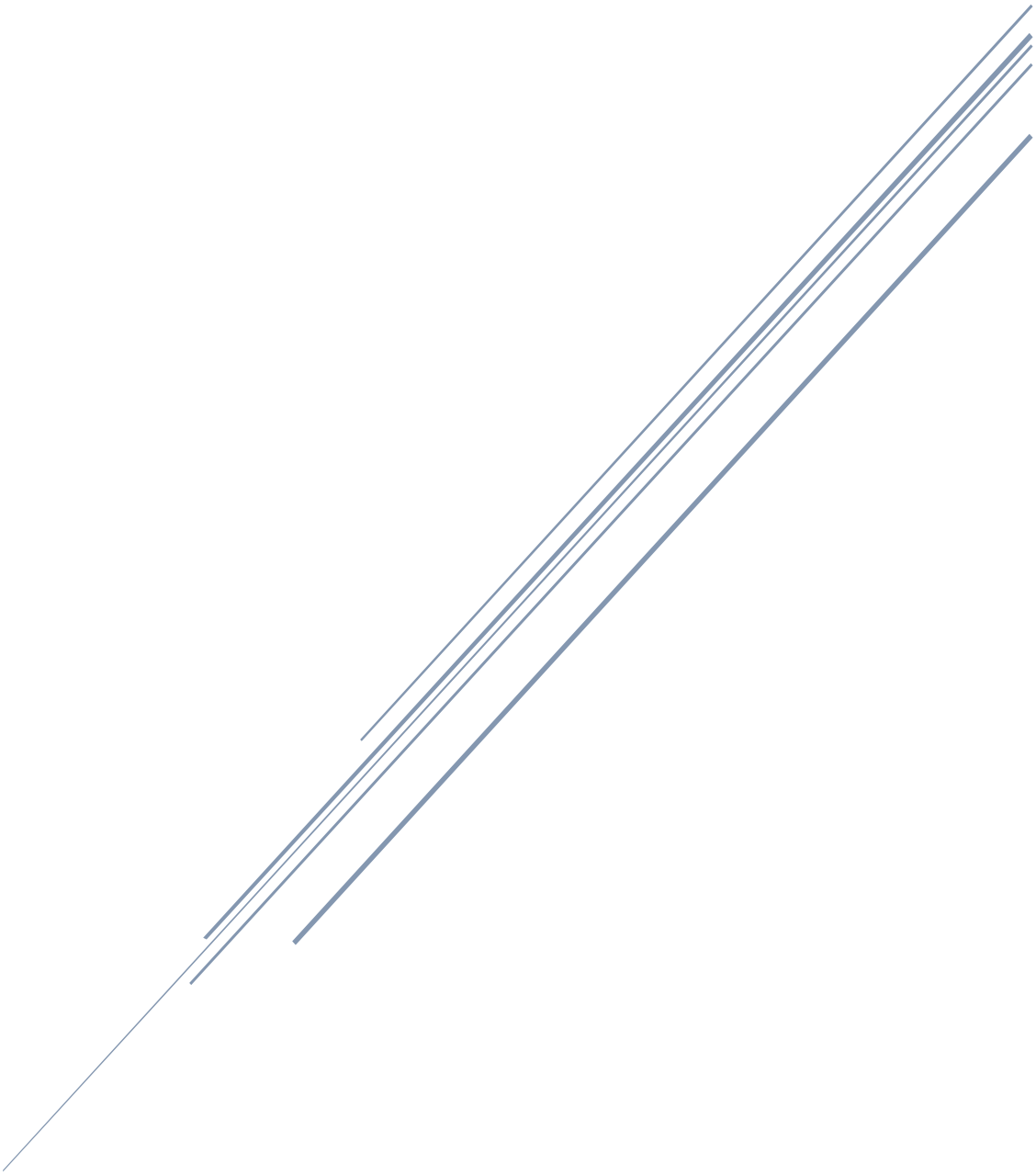
PROJET JEE : RAPPORT

Dervaux Xavier & Bianco Andy



HEPH-CONDORCET

Bloc 3 : Applications informatiques (Java EE)

Table des matières

1. Consignes du projet

« Il vous est demandé de réaliser le développement d'un site web par groupe de 2 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 Oracle + DAO

Le développement se fera avec Eclipse Néon et GlassFish 4.0.

En Option :

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

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 :
    a. Utilisation des procédures, fonctions stockées
    b. Utilisation des packages
    c. Utilisation des triggers
4. De gérer les exceptions
5. D'utiliser les curseurs, des variables de type record et des tableaux
6. De documenter les différents scripts d'accès à la base de données

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.

Anne Vandevorst & Brigitte Copin »

2. Notre énoncé

Notre programme va porter sur la réalisation d'un jeu d'Abalone en ligne, jouable par 2 personnes en simultané.

En voici les règles :

Abalone est un jeu de stratégie où s'affrontent deux joueurs l'un contre l'autre. L'un possède les billes blanches et l'autre les noires. Le gagnant est celui qui arrive à faire sortir 6 billes de l'adversaire en les poussant avec les siennes.

A chaque tour, le joueur peut déplacer 1 à 3 billes adjacentes vers des cases voisines.

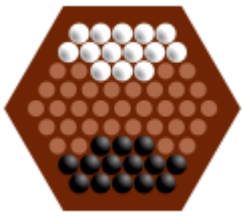


Figure 1- Position standard de depart

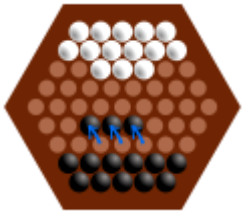


Figure 2 -Le joueur peut déplacer latéralement les billes

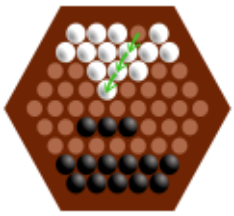


Figure 3 - On peut aussi les déplacer verticalement

On ne peut « pousser » les billes de l'adversaire que lorsqu'on est en supériorité numérique.

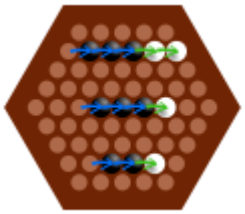


Figure 4 - Le joueur noir peut pousser les billes blanches car il est en supériorité numérique

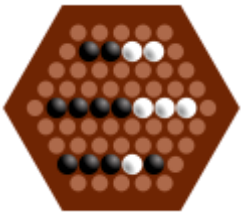


Figure 5 - Le joueur noir ne peut pas pousser une bille blanche si une bille de sa couleur est sur le chemin

D'un point de vue technique, le programme sera réalisé en Java EE conjointement avec une base de données Oracle interagissant avec le langage PL/SQL.

Le programme sera capable de :

- Proposer à deux joueurs de débuter une partie
- Respecter les règles du jeu.
- Détecter une partie gagnante et ainsi y mettre fin.
- Proposer un système de connexion utilisateur. Ils seront stockés en base de données
- Afficher les joueurs connectés et de permettre de démarrer une partie avec l'un deux.
- Conserver un historique des victoires et des défaites de chaque joueur, contre qui il a joué, le tout trié par date de partie.
- Gérer un système de trophées, succès pour donner un intérêt au joueur :
    - Gagner sa première partie.
    - Gagner 10 parties
    - Gagner sans perdre aucune bille.
    - …

3. Analyse

Afin de réaliser notre programme, nous avons mis en place une analyse UML en définissant à l'avance chacun des points qui allaient être programmés. De plus, s'agissant d'une application web, nous sommes partis du principe que l'interface utilisateur serait au centre de l'attention. Nous avons donc choisi l'approche de définir dans un premier temps nos écrans en nous basant sur les fonctionnalités que l'utilisateur à l'habitude d'utiliser dans son usage web de tous les jours. En accord avec notre énoncé personnel, nous avons défini que nous aurions besoin d'une page pour :

- La connexion / déconnexion / inscription
- L'affichage du menu
- Les modifications des paramètres du compte (Email, mot de passe)
- La gestion des Achievements
- L'affichage de l'historique
- Trouver un adversaire pour la partie
- Jouer une partie.

3.1. Les écrans

Voici les écrans que nous avons définis pour répondre à nos besoins :

3.1.1. La connexion / déconnexion / inscription

Un double écran, avec deux onglets. Chacun affichera un formulaire différent, un pour la connexion, l'autre pour l'inscription. Il est important que l'utilisateur soit connecté avant d'accéder a n'importe quelle autre page de l'application. On vérifiera donc sur chacune d'entre elle qu'il l'est bien, et si pas on le reverra sur cette page.

3.1.2. L'affichage du menu

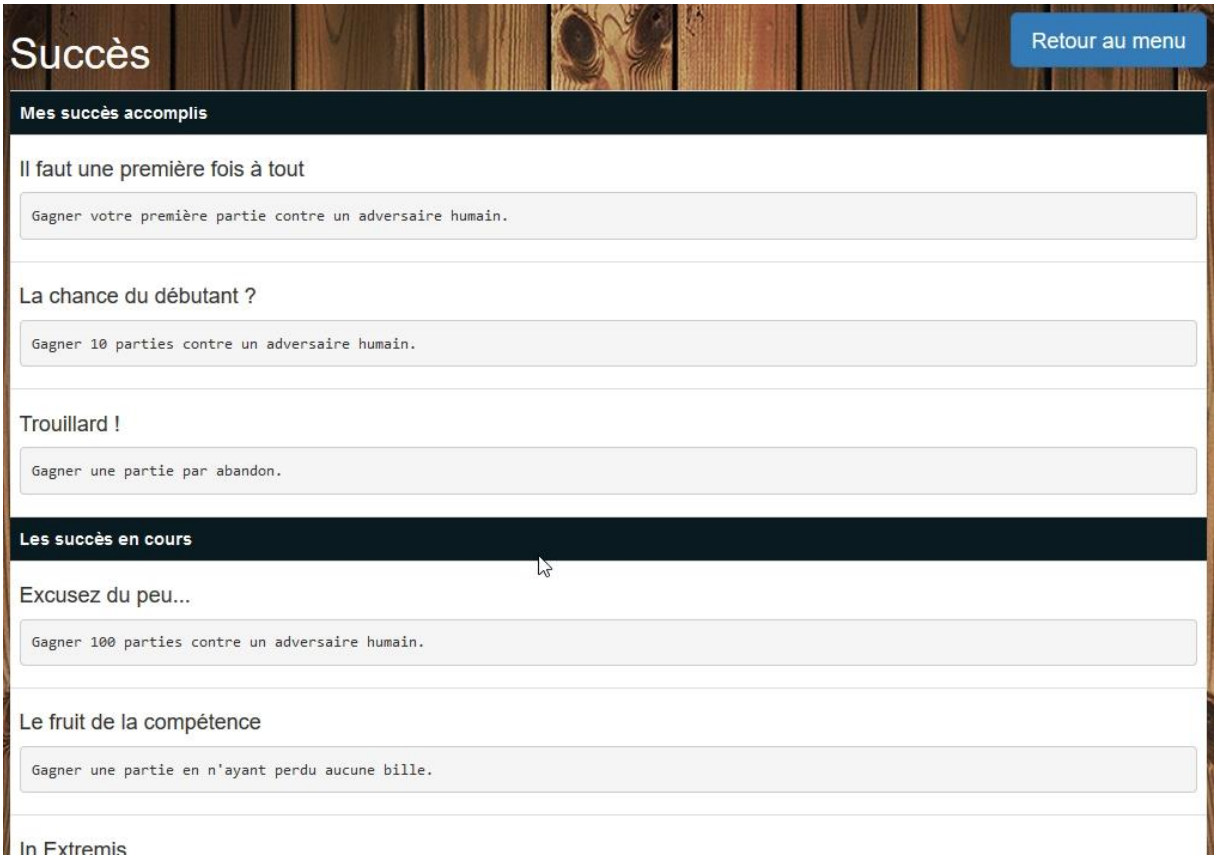
Le menu ne s'affichera que si l'utilisateur est connecté. Cinq boutons sont affichés, chacun redirigeant vers une fonctionnalité. Jouer renverra vers l'interface de recherche de joueur, Succès vers la gestion des Achievements, Historique vers la gestion des historiques, Paramètre permet de changer nos informations de compte, et Déconnecter permet de mettre fin à la session utilisateur.

3.1.3. Les modifications des paramètres du compte (Email, mot de passe)

Cette page proposera un formulaire permettant à l'utilisateur de modifier son mail, son mot de passe, ou les deux. Les champs vides seront ignorés tandis que les champs remplis seront mis à jour en base de données.

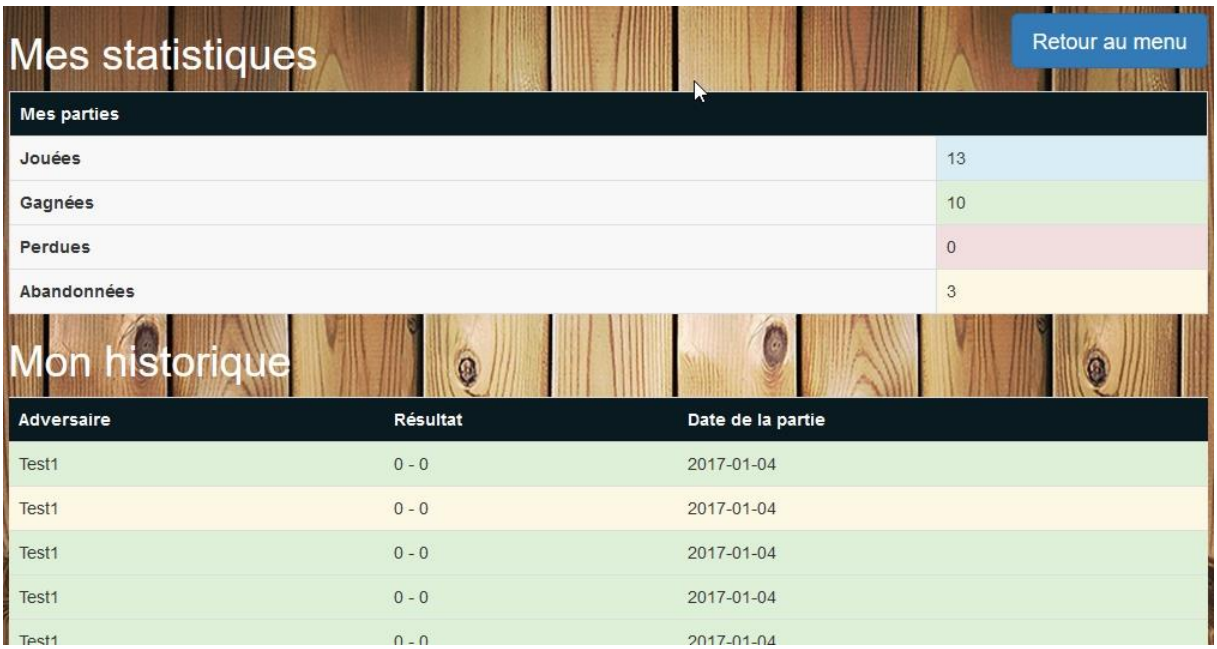
3.1.4. La gestion des Achievements

Cette page affichera les succès que l'utilisateur a déjà accomplis, et ceux qui sont encore à atteindre. Les succès seront gagnés pendant une partie en suivant les instructions données par ceux-ci.



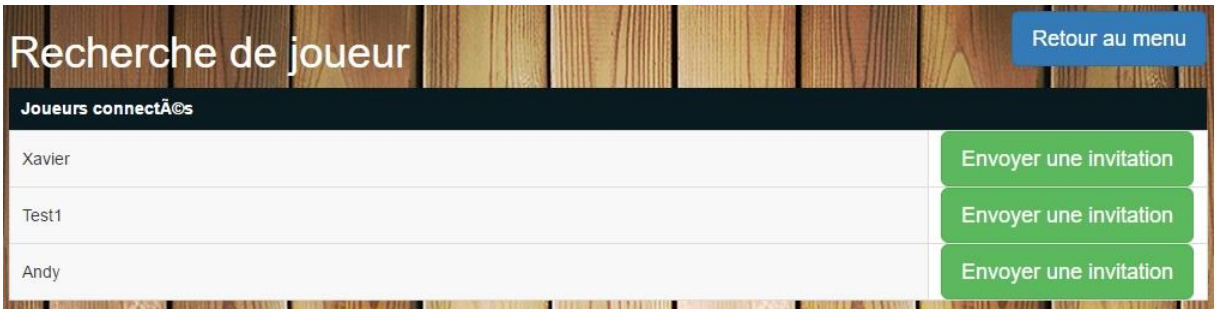
3.1.5. L'affichage de l'historique

Cette page propose l'affichage de l'historique des différentes parties du joueur, et signalera par un code couleur si celle-ci était une victoire, une défaite, ou un abandon de sa part.



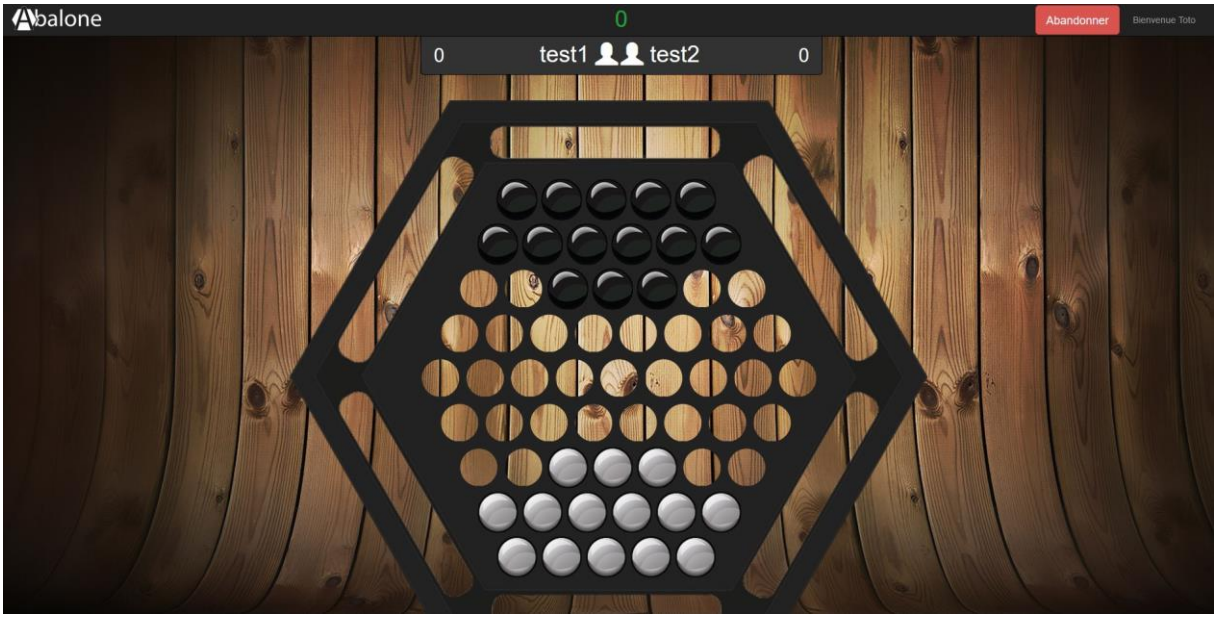
3.1.6. Trouver un adversaire pour la partie

Lorsque l'on appelle une partie, on sera dirigé vers cette page, ou on pourra trouver un adversaire pour démarrer une partie. Cette page affiche tous les joueurs connectés et en attente, on peut décider d'envoyer une invitation à l'un d'entre eux, et une fois qu'il aura accepté la partie démarrera.



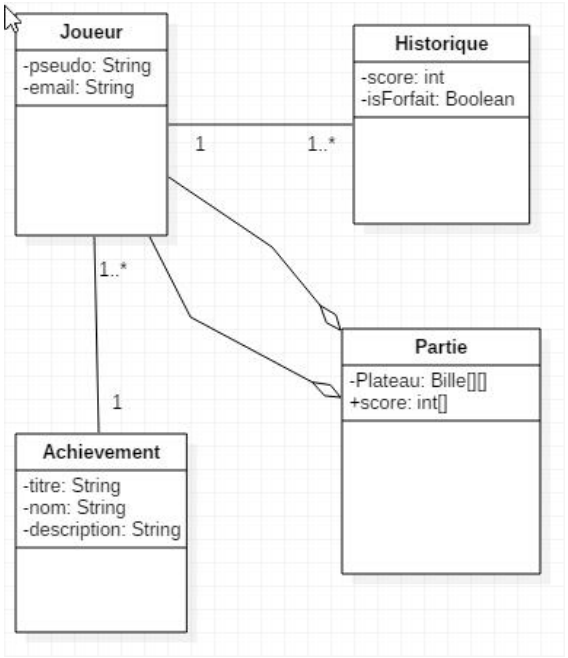
3.1.7. Jouer une partie.

Enfin on affiche la fenêtre de partie en elle-même, les deux joueurs ont accès au plateau, celui-ci est dynamiquement orienté afin que les billes du joueur soient vers le bas. La page affiche un bouton abandonner pour que le joueur puisse arrêter prématurément la partie s'il le souhaite.



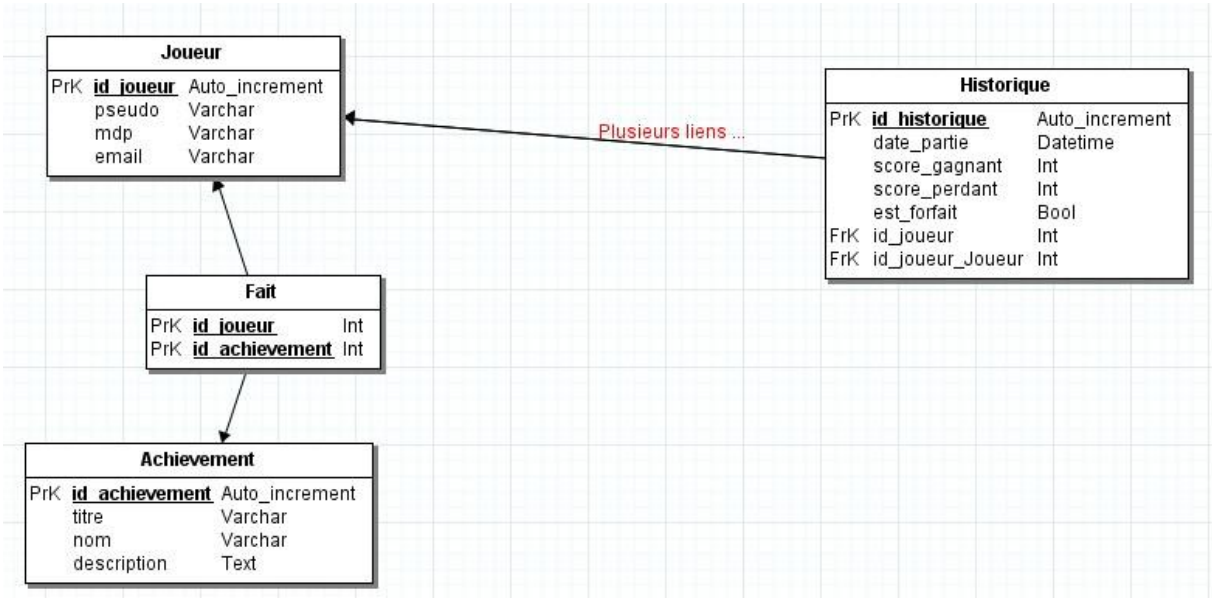
3.2. Analyse UML

Nous disposons de 4 classes métier : Joueur, Historique, Partie, et Achievement. Joueur centralisera tout l'aspect création de joueur. Achievement regroupera les créations, et les fonctions de détection d'Achievement. Historique centralisera toutes la gestion de l'historique d'un joueur. Enfin, partie, qui sera la classe centrale de cette application, gérera les règles d'une partie entre 2 joueurs, et enregistrera l'historique d'une partie à la victoire de celle-ci, en plus de gérer les succès éventuels accomplis lors de celle-ci.



3.3. Schéma relationnel SQL

Pour les particularités du schéma, nous avons un éclatement entre Joueur et Achievement car à un joueur peut correspondre plusieurs Achievement et à un Achievement peut correspondre plusieurs joueurs. De plus, un historique correspond à 2 joueurs, ainsi elle contiendra l'id de chacun d'eux.



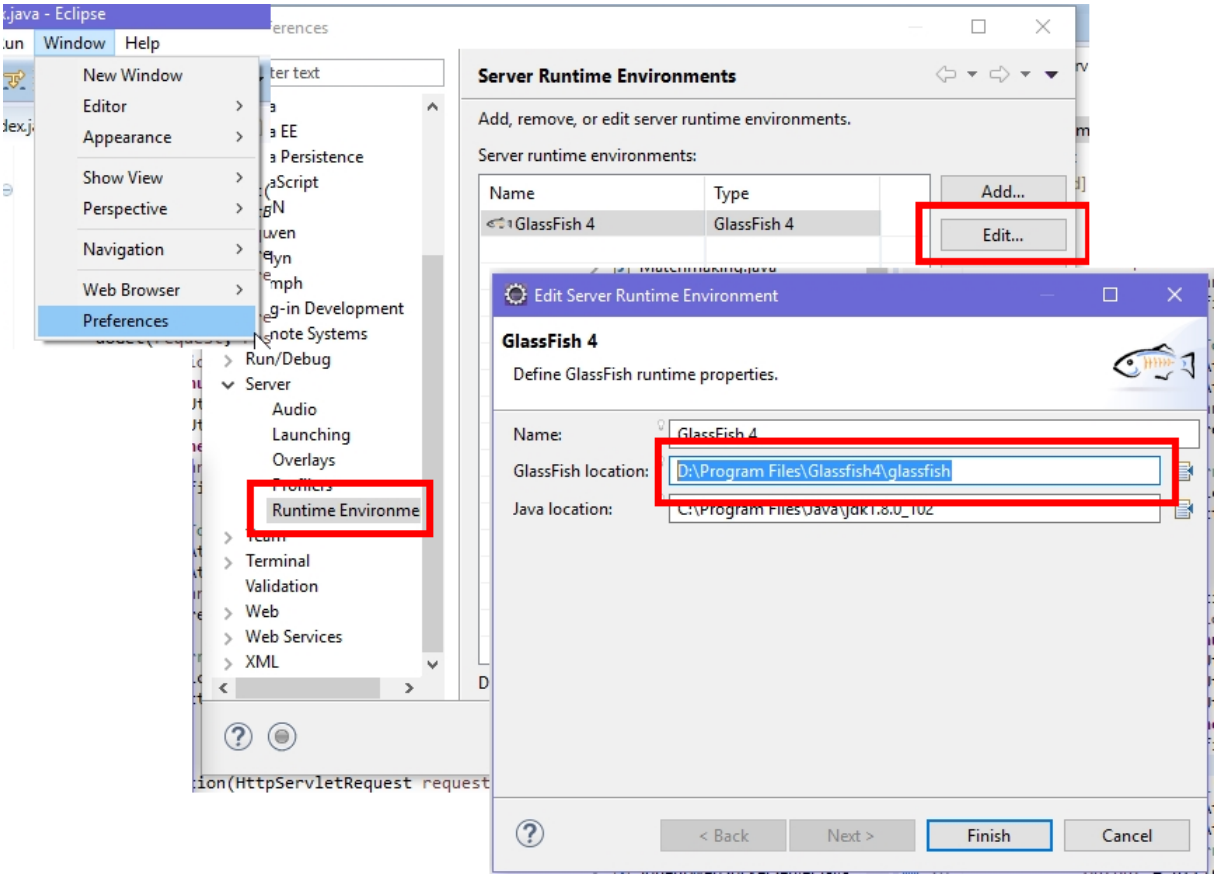
4. Utilisation du programme

Avant de pouvoir tester le programme, quelques détails seront à régler. En effet, étant donné notre problème d'installation de GlassFish avec le serveur SQL Oracle, nous avons dû l'installer dans un répertoire différent. Il faudra donc redéfinir l'emplacement de votre GlassFish pour pouvoir utiliser l'application. Pour ce faire, il suffit de procéder comme ceci.

Dans l'environnement Eclipse, cliquer dans la barre de menu sur :

Window > Preferences > (Dans la liste) Server > Runtime Environments

Dans cette fenêtre, cliquer sur GlassFish4, puis sur Edit, et dans la nouvelle fenêtre, modifiez le chemin de GlassFish par celui du GlassFish installé sur votre système.



Maintenant que GlassFish est correctement réglé, le programme devrait fonctionner comme prévu.

Par défaut, lorsqu'Eclipse compile une application, il l'ouvre dans un navigateur intégré au programme. Nous avons modifié les options du Workspace afin que cela ne soit pas le cas, et que celui-ci s'ouvre dans le navigateur par défaut du système. Pourquoi ? Car nous avons utilisé pour réaliser notre application des fonctionnalités HTML5 / CSS3 avancées qui ne sont pas prises en compte par le navigateur interne. Le programme risque de ne pas s'exécuter correctement s'il est testé depuis Eclipse. Lorsque le programme est démarré, la fenêtre de connexion / inscription s'affiche. Vous pouvez créer votre propre utilisateur, ou utiliser un des utilisateurs de test :

- test1@test.com mdp : 12345678
- test2@test.com mdp : 12345678

Libre ensuite à vous d'utiliser le programme comme vous l'entendez.

5. Technologies utilisées, quelques explications

Dans cette section, nous expliquons quelques particularités au niveau des technologies utilisées, pourquoi nous les avons choisies, et leur application concrète.

5.1. Bootstrap et jQuery : Des framework JavaScript

Bootstrap est un framework HTML, CSS, JavaScript qui a besoin de jQuery pour fonctionner correctement. Sa particularité est d'avoir énormément de composant préparés pour faciliter le développement d'interface utilisateur et nous permettre de nous concentrer sur des choses plus importantes, si celle-ci n'est pas au centre des préoccupations.

JQuery est une librairie rependue et très puissante contenant une série fonctions qui facilite le développement au quotidien en JavaScript.

5.2. Les WebSockets : Gestion des échanges client-serveur

Les WebSockets sont une technologie de communication orientée connexion, et représentent le moyen idéal d'établir une connexion entre un client et un serveur. Etant orienté connexion, nous sommes capables d'établir avec certitude la connexion d'un client, ainsi que sa déconnexion, ce qui offre un grand avantage par rapport aux requêtes AJAX classiques. Pour implémenter nos WebSockets, nous avons besoin de définir un serveur, et plusieurs clients peuvent s'y connecter. En JEE, nous définissons un socket en important les bibliothèques javax.websocket.*. Une classe seront considérée comme un serveur WebSocket si elle implémente les fonctionnalités suivantes :

```
@ServerEndpoint("/socket") //Défini l'adresse pour se connecter à la WebSocket
public class WebSocketServer {
    @OnOpen
    public void open(Session session) {
    } //Actions à effecter à la première connexion d'un client

    @OnClose
    public void close(Session session) {
    } //Actions à effectuer si le client se déconnecte ou ferme brutalement le navigateur

    @OnError
    public void onError(Throwable error) {
    } //Actions à effectuer en cas d'erreur

    @OnMessage
    public void OnMessage(String message, Session session)
    } //Actions à effectuer lorsqu'on reçoit un message de la part d'un client.
}
```

La première instruction, `@ServerEndpoint("/socket")` sert à définir le chemin d'accès à la socket. En pratique il faudrait donc se connecter à http://localhost/[application]/socket. Ensuite interviens une série de fonction qu'il nous incombe de gérer comme on l'entends, chacune représentant un évènement lié aux Sockets. Les fonctions open et close sont fiables, elles seront systématiquement appelées à chaque nouvelle connexion/déconnexion d'un client. De cette manière, nous pouvons gérer une liste de clients. La fonction la plus importante est OnMessage, qui sera appelée à chaque fois qu'un client envoie une information. Au sein de celle-ci, il nous incombe de vérifier le type du message, et ensuite envoyer les données fournies à une fonction de la classe appropriée. En effet, pour maintenant une architecture MVC, nous ne pouvons pas traiter immédiatement l'information dans la Socket. Celle-ci équivaut à un Controller en MVC, et doit donc renvoyer l'information à une classe Model pour que celle-ci soit gérée. Dans la pratique, on préfèrera utiliser une classe intermédiaire Handler pour gérer les messages de retour envoyés au client suite au résultat du traitement du Model.

6. Répartition des tâches

Côté serveur	*Côté client*
DAO, connexion Oracle : **DERVAUX**	**BIANCO** : Deployement descriptor web.xml
Création de la base de données : **DERVAUX**	**BIANCO** : Design de l'application
Servlet Histo.java : **DERVAUX**	**BIANCO** : JSP historique.jsp
Servlet Index.java : **DERVAUX**	**BIANCO** : JSP index.jsp
Servlet Jouer.java : **BIANCO**, **DERVAUX**	**BIANCO**, **DERVAUX** : JSP jouer.jsp
Servlet Matchmaking.java : **DERVAUX**	**BIANCO** : JSP matchmaking.jsp
Servlet Menu.java : **DERVAUX**	**BIANCO** : JSP menu.jsp
Servlet Settings.java : **BIANCO**	**DERVAUX** : JSP setting.jsp
Servlet Succès.java : **DERVAUX**	**BIANCO** : JSP succes.jsp
Classe Utilitaire : **BIANCO**	**BIANCO** : Fichier JavaScript abalone.js
Classe Achievement.java : **DERVAUX**	**BIANCO**, **DERVAUX** : Fichier JavaScript partie.js
Classe Historique.java : **BIANCO**	**BIANCO** : Fichier CSS style.css
Classe Identification.java : **DERVAUX**	
Classe Joueur.java : **DERVAUX**	
Classe Partie.java : **BIANCO**, **DERVAUX**	
WebSockets Servers : **DERVAUX**	**BIANCO** : WebSocket Client socket.js
WebSockets Handlers : **DERVAUX**	**BIANCO** : WebSocket Client partie.js
JavaBeans : **DERVAUX**	

Nous avons tenté au maximum de répartir les tâches de sorte que nous fassions chacun une partie de développement client, et chacun une partie de développement serveur. Bien qu'au final nous soyons plus restés dans notre domaine respectif, nous avons toutefois expérimentés respectivement chacun des aspects du développement en Java EE.

7. Annexe : Script de création de la base de données

```
CREATE TABLE joueur(
    id           INTEGER           NOT NULL,
    pseudo    VARCHAR2(64)   NOT NULL,
    mdp       VARCHAR2(256)  NOT NULL,
    email     VARCHAR2(64)   NOT NULL,
    CONSTRAINT pk_joueur PRIMARY KEY(id)
);

CREATE TABLE historique(
    id               INTEGER       NOT NULL,
    date_partie      DATE          NOT NULL,
    score_gagnant    INTEGER       NOT NULL,
    score_perdant    INTEGER       NOT NULL,
    est_forfait      NUMBER(1,0)   NOT NULL,
    id_gagnant       INTEGER       NOT NULL,
    id_perdan INTEGER       NOT NULL,
    CONSTRAINT pk_historique       PRIMARY KEY(id),
    CONSTRAINT fk_histo_gagnant  FOREIGN KEY(id_gagnant) REFERENCES joueur(id) ON DELETE CASCADE,
    CONSTRAINT fk_histo_perdant  FOREIGN KEY(id_perdant) REFERENCES joueur(id) ON DELETE CASCADE
);

CREATE TABLE achievement(
    id               INTEGER       NOT NULL,
    titre            VARCHAR2(256) NOT NULL,
    nom              VARCHAR2(256) NOT NULL,
    description      VARCHAR2(512) NOT NULL,
    CONSTRAINT pk_achievement      PRIMARY KEY(id)
);

CREATE TABLE fait(
    id_joueur               INTEGER NOT NULL,
    id_achievement          INTEGER NOT NULL,
    CONSTRAINT pk_fait      PRIMARY KEY(id_joueur ,id_achievement),
    CONSTRAINT fk_fait_joueur FOREIGN KEY(id_joueur) REFERENCES joueur(id) ON DELETE CASCADE,
    CONSTRAINT fk_fait_achievement FOREIGN KEY(id_achievement) REFERENCES achievement(id) ON DELETE CASCADE
);

--
-- Package utilisé pour l'aspect persistance, stocke le dernier id créé
--
create or replace PACKAGE last_inserted_id IS
  lastId INTEGER;
END;

--
-- Triggers pour Auto Increment
--
CREATE SEQUENCE joueur_seq START WITH 1;
CREATE SEQUENCE histo_seq START WITH 1;
CREATE SEQUENCE achiev_seq START WITH 1;

CREATE OR REPLACE TRIGGER joueur_bir
BEFORE INSERT ON joueur
FOR EACH ROW
BEGIN
  SELECT joueur_seq.NEXTVAL
  INTO   :new.id
  FROM   dual;
  last_inserted_id.lastId := :new.id; --On stocke l'id créé dans le package
END;
/
```

```
CREATE OR REPLACE TRIGGER historique_bir
BEFORE INSERT ON historique
FOR EACH ROW
BEGIN
  SELECT histo_seq.NEXTVAL
  INTO   :new.id
  FROM   dual;
  last_inserted_id.lastId := :new.id; --On stocke l'id créé dans le package
END;
/


CREATE OR REPLACE TRIGGER achiev_bir
BEFORE INSERT ON achievement
FOR EACH ROW
BEGIN
  SELECT achiev_seq.NEXTVAL
  INTO   :new.id
  FROM   dual;
  last_inserted_id.lastId := :new.id; --On stocke l'id créé dans le package
END;
/

--
-- Procédure stockée pour récupérer le dernier id stocké
--
create or replace FUNCTION last_inserted_rowid (no INTEGER)
    RETURN INTEGER IS last_rowid INTEGER;
BEGIN
    last_rowid := last_inserted_id.lastId;
    RETURN last_rowid;
END;
```

Pour toute consultation de la base de données en elle-même, voici les identifiants de connexion :

- Utilisateur : exa7
- Mot de passe : AbaloneJEE23#42@12
- Adresse BDD : char-oracle11.condorcet.be