



Master 1 Technologie de l'Internet

Web Avancée

Compte Rendu : Projet Belote

Auteurs :

M. Yan MICHIELS

M. Johann CEBOLLADO

Encadrant :

Pr. Eric CARIOU

Version 0.1 du 24 mai 2020

Table des matières

| | | |
|-----------|---|-----------|
| 0.1 | Introduction | 1 |
| I | Données | 3 |
| 0.2 | Schema BDD | 5 |
| 0.2.1 | Script SQL | 5 |
| 0.2.2 | Diagramme de la base de donnée belote | 7 |
| 0.2.3 | Diagramme de classe | 8 |
| 0.2.4 | Justification des types utilisés | 9 |
| 0.3 | Pojos | 12 |
| 0.4 | Code | 12 |
| 0.4.1 | table utilisé par JPA | 12 |
| 0.5 | déploiement de la base sur le cloud Azure | 13 |
| II | Web | 15 |
| 0.6 | Fonctionnement général du jeu | 17 |
| 0.7 | Interface Web | 17 |
| 0.8 | Serveur | 22 |
| 0.9 | JSP | 22 |
| 0.10 | WebSocket | 22 |
| 0.11 | Gestion de partie | 23 |
| 0.12 | GitLab | 23 |

0.1 Introduction

Projet de deuxième semestre de Master 1 Informatique, celui-ci consiste à l'implémentation du jeu de carte la belote. Nous avons développé l'application suivant les precepts de l'architecture 3 tiers du Web qui permet de modulariser notre partie données et web, permettant une meilleur évolutivité de l'application, un meilleur passage à l'échelle, et une maintenance plus aisé. Dans notre solution nous nous basons sur différentes technologies avec pour la partie données : MySQL pour la base de données, eclipseLink pour la persistance des données. Concernant la partie Web, nous utilisons le serveur Tomcat....

Pour pouvoir utilisé aux mieux ces technologies en gérant les librairies, les dépendances le versionning, mais également la compilation et déploiement des applications Java, nous avons utilisé l'outil Maven qui permet d'automatiser la gestion de projets Java. Le développement de l'application a été réalisé sur l'IDE IntelliJ. Dans notre première partie figure les informations concernant nos données avec la justifications des types choisie pour notre base. Puis dans la seconde partie, nous parlons de l'architecture de notre application Web, du fonctionnement de notre jeu de belote et de son interface.

Première partie

Données

0.2 Schema BDD

Nous utilisons pour notre projet le système de gestion de base de données **MySQL** en version 8.0.20. Mais pour le déploiement sur Azure, nous utilisons **MariaDB** qui est un fork de MySQL et compatible avec ses drivers.

0.2.1 Script SQL

</> File program 1: *Script SQL de la création des tables de la base de donnée belote* </>

```
1 CREATE TABLE `joueur` (  
2     `joueur_id` int(11) NOT NULL,  
3     `nb_victoire` int(11) DEFAULT NULL,  
4     `score_moyen` float DEFAULT NULL,  
5     `nb_partie` int(11) DEFAULT NULL,  
6     `pseudo` varchar(20) NOT NULL,  
7     `DTYPE` varchar(15) NOT NULL,  
8     PRIMARY KEY (`joueur_id`)  
9 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
10  
11  
12 CREATE TABLE `humain` (  
13     `joueur_id` int(11) NOT NULL,  
14     # hashera avec md5 pour le mot de passe  
15     `mot_de_passe` char(32) NOT NULL,  
16     `age` smallint(6) DEFAULT NULL,  
17     `ville` varchar(20) DEFAULT NULL,  
18     `sexe` char(1) DEFAULT NULL,  
19     PRIMARY KEY (`joueur_id`),  
20     FOREIGN KEY (`joueur_id`) REFERENCES `joueur` (`joueur_id`)  
21 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
22  
23  
24 CREATE TABLE `robot` (  
25     `joueur_id` int(11) NOT NULL,  
26     `programme` varchar(20) NOT NULL,  
27     PRIMARY KEY (`joueur_id`),  
28     FOREIGN KEY (`joueur_id`) REFERENCES `joueur` (`joueur_id`)  
29 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
30  
31  
32 CREATE TABLE `partie` (  
33     `partie_id` int(11) NOT NULL,  
34     `eq_1a` int(11) DEFAULT NULL,
```

```

35 `eq_1b` int(11) DEFAULT NULL,
36 `score_eq1` smallint(6) DEFAULT NULL,
37 `eq_2a` int(11) DEFAULT NULL,
38 `eq_2b` int(1) DEFAULT NULL,
39 `score_eq2` smallint(6) DEFAULT NULL,
40 `duree` time DEFAULT NULL,
41 PRIMARY KEY (`partie_id`),
42 FOREIGN KEY (`eq_1a`) REFERENCES `joueur` (`joueur_id`),
43 FOREIGN KEY (`eq_1b`) REFERENCES `joueur` (`joueur_id`),
44 FOREIGN KEY (`eq_2a`) REFERENCES `joueur` (`joueur_id`),
45 FOREIGN KEY (`eq_2b`) REFERENCES `joueur` (`joueur_id`)
46 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
47
48
49 CREATE TABLE `manche` (
50 `partie_id` int(11) NOT NULL,
51 `manche_nb` tinyint(4) NOT NULL,
52 `couleur_atout` char(8) DEFAULT NULL,
53 `joueur_prenant` int(11) DEFAULT NULL,
54 `atout_initial` varchar(13) DEFAULT NULL,
55 `atout_final` varchar(13) DEFAULT NULL,
56 `point_manche` smallint(6),
57 PRIMARY KEY (`partie_id`, `manche_nb`),
58 FOREIGN KEY (`partie_id`) REFERENCES `partie` (`partie_id`),
59 FOREIGN KEY (`joueur_prenant`) REFERENCES `joueur` (`joueur_id`)
60 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
61
62
63 CREATE TABLE `main` (
64 `partie_id` int(11) NOT NULL,
65 `manche_nb` tinyint(4) NOT NULL,
66 `joueur_id` int(11) NOT NULL,
67 `carte_1` varchar(13) DEFAULT NULL,
68 `carte_2` varchar(13) DEFAULT NULL,
69 `carte_3` varchar(13) DEFAULT NULL,
70 `carte_4` varchar(13) DEFAULT NULL,
71 `carte_5` varchar(13) DEFAULT NULL,
72 `carte_6` varchar(13) DEFAULT NULL,
73 `carte_7` varchar(13) DEFAULT NULL,
74 `carte_8` varchar(13) DEFAULT NULL,
75 PRIMARY KEY (`partie_id`, `manche_nb`, `joueur_id`),
76 FOREIGN KEY (`partie_id`, `manche_nb`) REFERENCES `manche`
77     (`partie_id`, `manche_nb`),
78 FOREIGN KEY (`joueur_id`) REFERENCES `joueur` (`joueur_id`)
79 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

```
79
80 CREATE TABLE `plis` (
81   `partie_id` int(11) NOT NULL,
82   `manche_nb` tinyint(4) NOT NULL,
83   `plis_nb` tinyint(4) NOT NULL,
84   `joueur_debut` int(11) DEFAULT NULL,
85   `carte_1` varchar(13) DEFAULT NULL,
86   `carte_2` varchar(13) DEFAULT NULL,
87   `carte_3` varchar(13) DEFAULT NULL,
88   `carte_4` varchar(13) DEFAULT NULL,
89   `note` char(1) DEFAULT ' ',
90   PRIMARY KEY (`partie_id`, `manche_nb`, `plis_nb`),
91   FOREIGN KEY (`partie_id`, `manche_nb`) REFERENCES `manche`
    → (`partie_id`, `manche_nb`),
92   FOREIGN KEY (`joueur_debut`) REFERENCES `joueur` (`joueur_id`)
93 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

0.2.2 Diagramme de la base de donnée belote

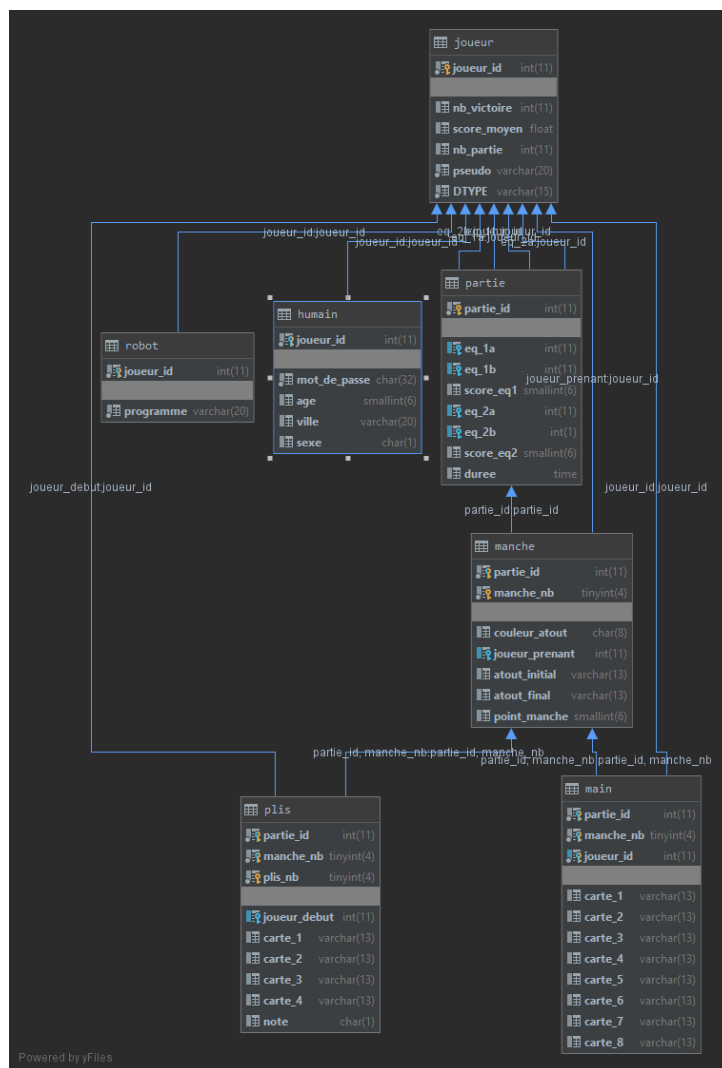


FIGURE 1 – Diagramme généré de base de données belote

0.2.3 Diagramme de classe

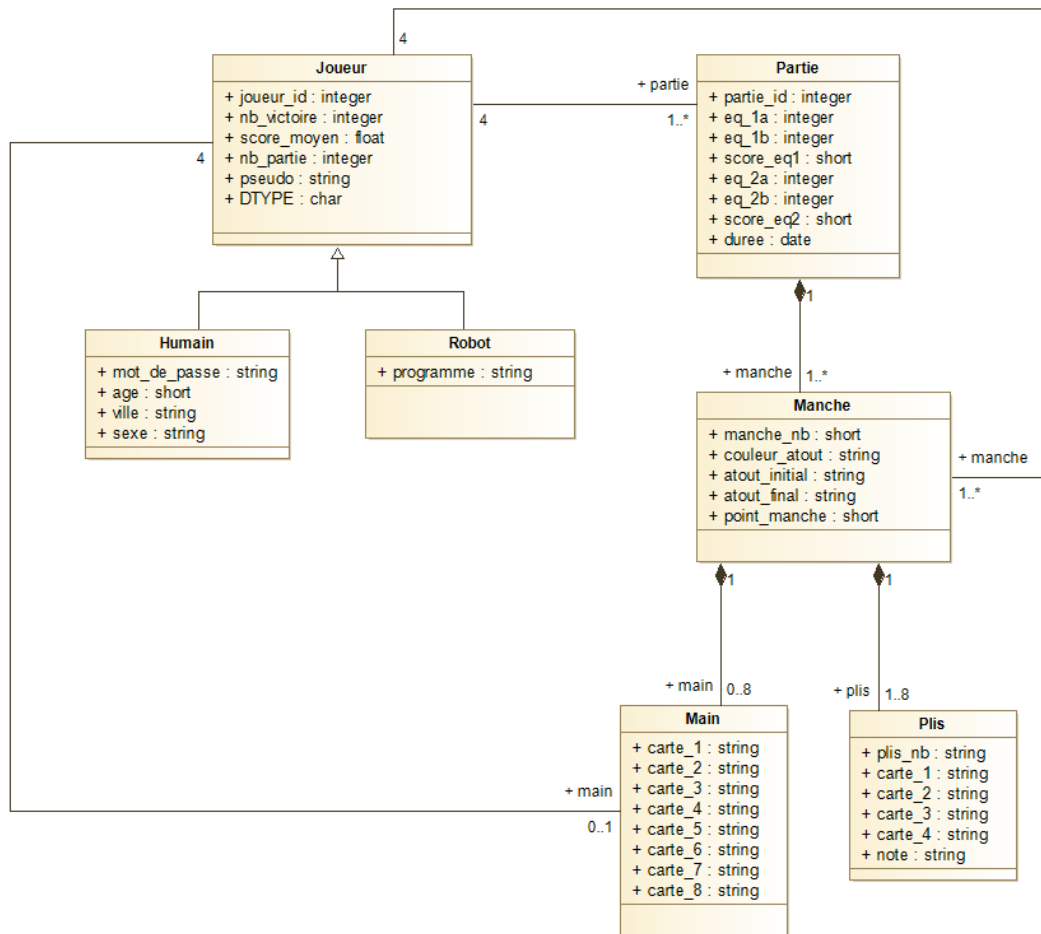


FIGURE 2 – Diagramme de classe modélisant le système

0.2.4 Justification des types utilisés

- Utilisation du type **char** plus optimal pour notre utilisation pour une donnée de type :
 - sexe
 - donnée : char(1) prenant soit H|F
 - mot_de_passe
 - donnée : char(32)
- Utilisation du type **varchar** plus optimal pour notre utilisation pour une donnée de type :
 - pseudo
 - donnée : varchar(32) stocke le pseudo d'un joueur
 - DTYPE
 - donnée : varchar(15) stocke le nom de la sous-classe d'un joueur (humain ou robot).
 - ville
 - donnée : varchar(20) stocke la ville du joueur.

- programme
 - donnée : varchar(20) stocke le nom du programme du robot implémenté dans la couche métier.
- atout_initial
 - donnée : varchar(20) stocke le nom du programme du robot implémenté dans la couche métier.
- atout_final
 - donnée : varchar(13) stocke la couleur et la valeur via une énumération.
- carte_1
 - donnée : varchar(13) carte 1 du joueur stocke la couleur et la valeur via une énumération.
- carte_2
 - donnée : varchar(13) carte 2 du joueur stocke la couleur et la valeur via une énumération.
- 'carte_3'
 - donnée : varchar(13) carte 3 du joueur stocke la couleur et la valeur via une énumération.
- carte_4
 - donnée : varchar(13) carte 4 du joueur stocke la couleur et la valeur via une énumération.
- carte_5
 - donnée : varchar(13) carte 5 du joueur stocke la couleur et la valeur via une énumération.
- carte_6
 - donnée : varchar(13) carte 6 du joueur stocke la couleur et la valeur via une énumération.
- carte_7
 - donnée : varchar(13) carte 7 du joueur stocke la couleur et la valeur via une énumération.
- carte_8
 - donnée : varchar(13) carte 8 du joueur stocke la couleur et la valeur via une énumération.
- note
 - donnée : varchar(8) prenant soit RIEN, BELOTE, REBELOTE ;
- Utilisation du type **int** plus optimal pour notre utilisation pour une donnée de type :
 - joueur_id
 - donnée : int(11) id du joueur pouvant aller de -2147483648 à 2147483647 ;
 - nb_victoire

- donnée : int(11) nombre de victoire du joueur pouvant aller de -2147483648 à 2147483647 ;
- nb_partie
 - donnée : int(11) nombre de partie du joueur pouvant aller de -2147483648 à 2147483647 ;
- partie_id
 - donnée : int(11) nombre de partie du joueur pouvant aller de -2147483648 à 2147483647 ;
- eq_1a
 - donnée : int(11) id du joueur a de l'équipe 1 a pouvant aller de -2147483648 à 2147483647 ;
- eq_1b
 - donnée : int(11) id du joueur b de l'équipe 1 a pouvant aller de -2147483648 à 2147483647 ;
- eq_2a
 - donnée : int(11) id du joueur a de l'équipe 2 a pouvant aller de -2147483648 à 2147483647 ;
- eq_2b
 - donnée : int(11) id du joueur b de l'équipe 2 a pouvant aller de -2147483648 à 2147483647 ;
- joueur_prenant
 - donnée : int(11) id du joueur qui prend l'atout pouvant aller de -2147483648 à 2147483647 ;
- joueur_debut
 - donnée : int(11) id du joueur qui est le premier à commencer le plis pouvant aller de -2147483648 à 2147483647 ;
- Utilisation du type **smallint** plus optimal pour notre utilisation pour une donnée de type :
 - age
 - donnée : smallint(6) age du joueur pouvant aller de -32768 à 32767.
 - score_eq1
 - donnée : smallint(6) score de l'équipe 1 a pouvant aller de -32768 à 32767. Justification, si l'une des deux équipes atteint de le score de 501 ou le dépasse alors elle a gagné.
 - score_eq2
 - donnée : smallint(6) score du joueur b de l'équipe 1 a pouvant aller de -32768 à 32767. Justification, si l'une des deux équipes atteint de le score de 501 ou le dépasse alors elle a gagné.
 - point_manche

- donnée : `smallint(6)` point qu'une équipe peut avoir lors d'une manche pouvant aller de -32768 à 32767. Justification, si résultat positif équipe 1 gagne la manche sinon équipe 2.
- Utilisation du type **`tinyint`** plus optimal pour notre utilisation pour une donnée de type :
 - `manche_nb`
 - donnée : `tinyint(4)` nombre de manche lors d'une partie de belote pouvant aller de -128 à 127.
 - `plis_nb`
 - donnée : `tinyint(4)` nombre de plis lors d'une manche de belote pouvant aller de -128 à 127. Justification : il y a 8 plis par manche.
- Utilisation du type **`float`** plus optimal pour notre utilisation pour une donnée de type :
 - `score_moyen`
 - donnée : `float` score moyen propre a chaque joueur lors d'une manche de belote. Justification : chiffre à virgule a cause de la division.

0.3 Pojos

0.4 Code

Nous avons rencontré des soucis lors du mappage de la table `Partie` en classe : initialement, nous avions prévus de sortir les données correspondant aux équipes dans une classe `Equipe` dans le but de factoriser les attributs (le score et les 2 joueurs).

Malheureusement, JPA ne semble pas apprécier les clefs étrangères dans les classes embarquées et refuse de fonctionner. Le code de la tentative est présent mais commenté dans la classe `Partie`. Si nous parvenons à trouver une solution à ce problème, elle sera mise en place au prochain livrable.

0.4.1 table utilisé par JPA

```
</> File program 2: Script de la table utilisée par JPA pour la persistance </>
1 create table `SEQUENCE` (
2     `SEQ_NAME` varchar(15) NOT NULL,
3     `SEQ_COUNT` int(11) DEFAULT '0',
4     PRIMARY KEY (`SEQ_NAME`)
5 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
6 insert into `SEQUENCE`
7     values (`joueur`, 0);
8 insert into `SEQUENCE`
9     values (`manche`, 0);
```

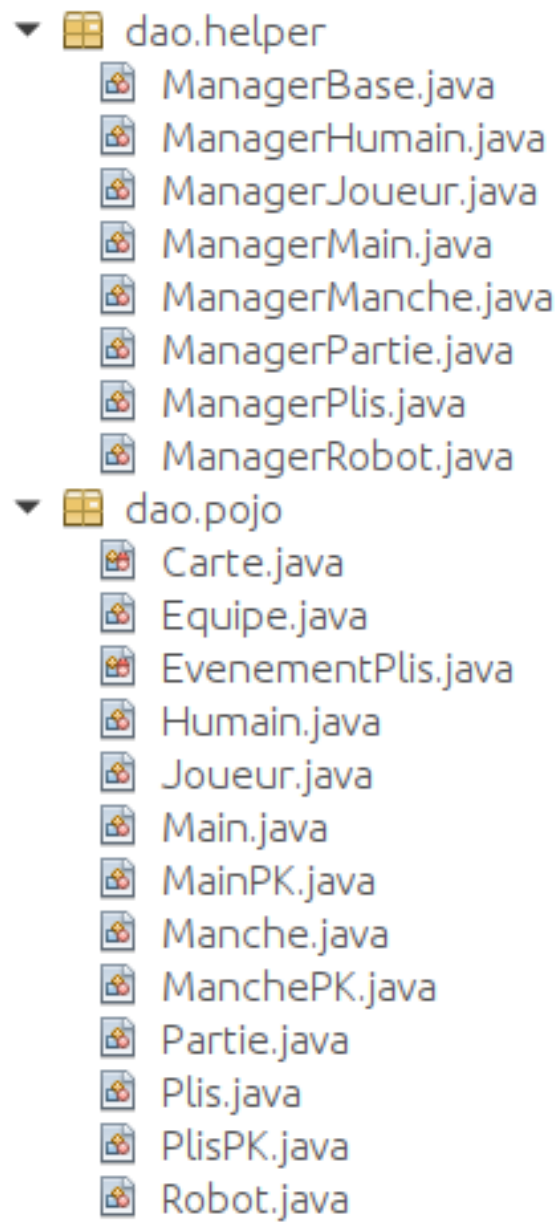



FIGURE 3 – Pojos générés par JPA

0.5 déploiement de la base sur le cloud Azure

Via notre module Cloud avec monsieur Khaled Khebbab, nous avons étudiés la solution cloud computing de chez Microsoft, **Microsoft Azure** (Mirabile Visu : [Vidéo de la présentation](#)), et nous avons voulu expérimenter l'hébergement de notre base de donnée sur cette solution de cloud.

Deuxième partie

Web

0.6 Fonctionnement général du jeu

Pour la réalisation de notre jeux nous avons mis en place un système où

- Un joueur possède quatre états :
 - déconnecté : le joueur est déconnecté
 - libre : le joueur est connecté et ne fait rien
 - en attente : le joueur est en attente de démarrage d'une partie
 - en jeux : le joueur est en train de jouer une partie
- le serveur détermine les cartes à jouer pour le client, lui envoi toutes les possibilités et le client n'a juste après qu'à sélectionner les cartes disponibles. Cette implémentation a été choisi pour minimiser le travail du code coté client et limité la confiance attribuer au client et ainsi limiter la possibilité de triche. Nous aurons donc pour réaliser cela une mise a jour dynamique au travers de WebSocket pour :
 - gérer le déroulement d'une partie
 - mise a jour de la liste de joueur connecté

Nous proposons dans un premier temps, plusieurs pages. Dans le cas le plus extrêmes, on pourra faire le "changement" de page via le javascript (et donc avoir un vrai MVC).

0.7 Interface Web

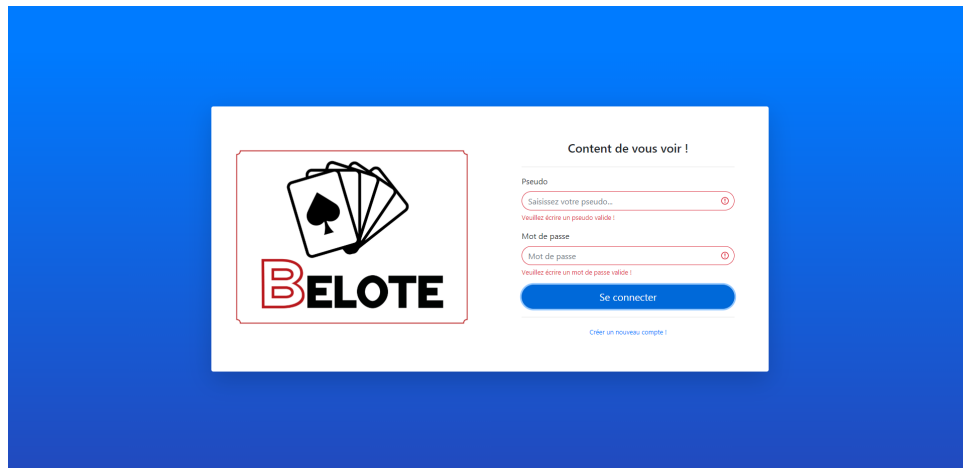
Pour une interface épurée et moderne, nous avons utilisés les outils comme :

- le framework de Twitter pour la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.)
- jQuery qui est une bibliothèque JavaScript créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web.
- chart.js pour la réalisation de graphique en html5.
- dataTables pour la réalisation de tableau dynamique.

L'interface de notre jeu est composée de :

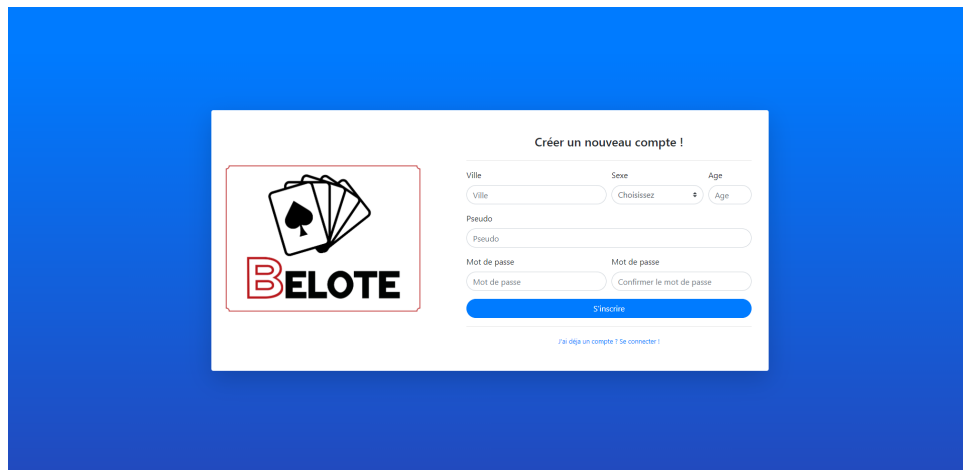
- page de connexion (première page lancée par le serveur).
- page d'inscription.
- page d'accueil du dashboard.
- page d'édition du profil du joueur du dashboard.
- page de création de partie du dashboard.
- page de statistiques du joueur du dashboard.
- page du jeu.

L'interface se compose des pages suivantes :



The screenshot shows the login page for 'BELOTE'. On the left is the logo, which consists of a fan of four playing cards (Ace, King, Queen, Jack of Spades) above the word 'BELOTE' in a bold, black, sans-serif font. The 'B' is stylized with a red outline. On the right, the heading 'Content de vous voir !' is displayed. Below it, there are two input fields: 'Pseudo' with a placeholder 'Saisissez votre pseudo...' and 'Mot de passe' with a placeholder 'Mot de passe'. Both fields have red outlines and a small red eye icon on the right. Below the 'Mot de passe' field is a blue button labeled 'Se connecter'. At the bottom, there is a link that says 'Créer un nouveau compte !'.

FIGURE 4 – Page de connexion pour le client



The screenshot shows the registration page for 'BELOTE'. On the left is the same logo as in Figure 4. On the right, the heading 'Créer un nouveau compte !' is displayed. Below it, there are several input fields: 'Ville' (with a dropdown arrow), 'Sexe' (with a dropdown arrow), 'Age' (with a dropdown arrow), 'Pseudo', 'Mot de passe' (with a placeholder 'Mot de passe'), and 'Confirmer le mot de passe' (with a placeholder 'Confirmer le mot de passe'). There are also red outlines around the 'Pseudo' and 'Mot de passe' fields. Below the 'Confirmer le mot de passe' field is a blue button labeled 'S'inscrire'. At the bottom, there is a link that says 'J'ai déjà un compte ? Se connecter !'.

FIGURE 5 – Page d'inscription pour le client pour le client

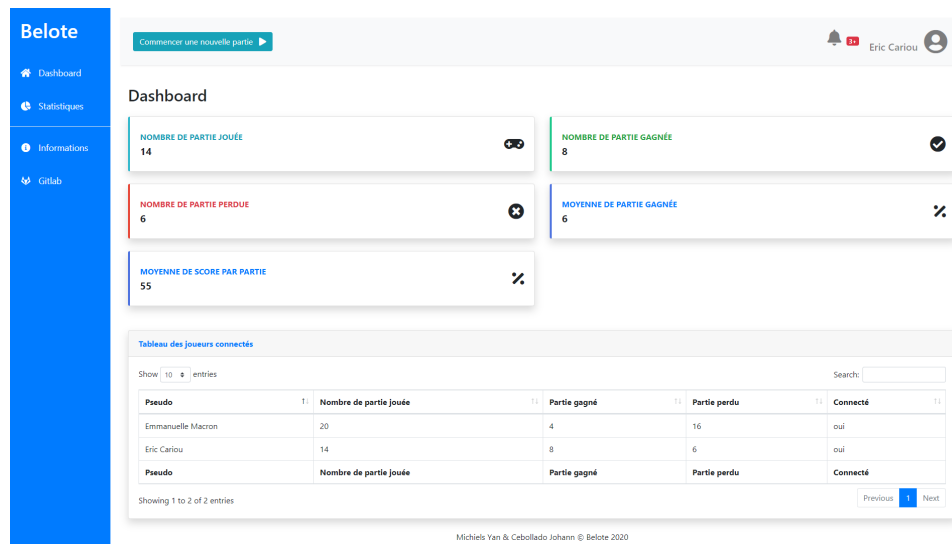


FIGURE 6 – Page d’accueil pour le client

The 'Editer le profil' page allows a user to update their personal information. It features a profile picture placeholder for 'Eric Carliou' and a form titled 'Informations Personnelles' with fields for Pseudo, Age, Sexe, Ville, and two password fields (Mot de passe). The form includes 'Sauvegarder' and 'Annuler' buttons. A footer note reads: 'Michiels Yan & Cebollado Johann © Belote 2020'.

FIGURE 7 – Page de modification du joueur pour le client

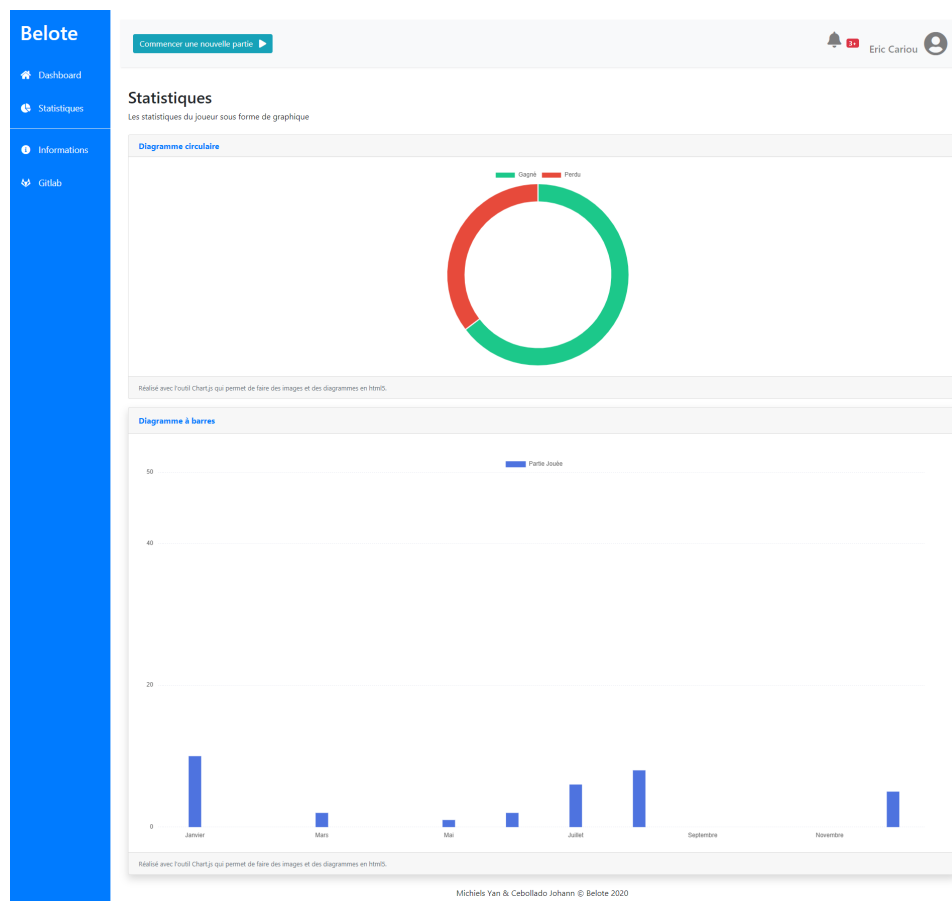


FIGURE 8 – Page de statistiques du joueur pour le client

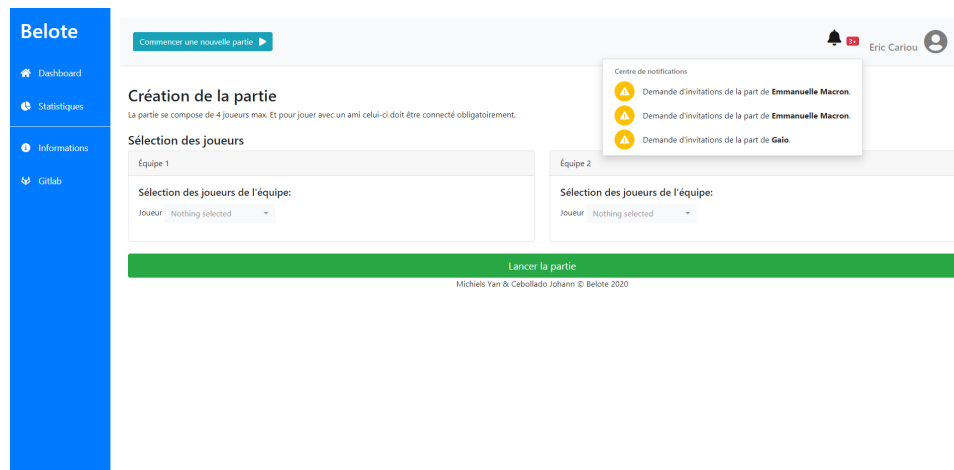


FIGURE 9 – Page de création de la partie du joueur pour le client

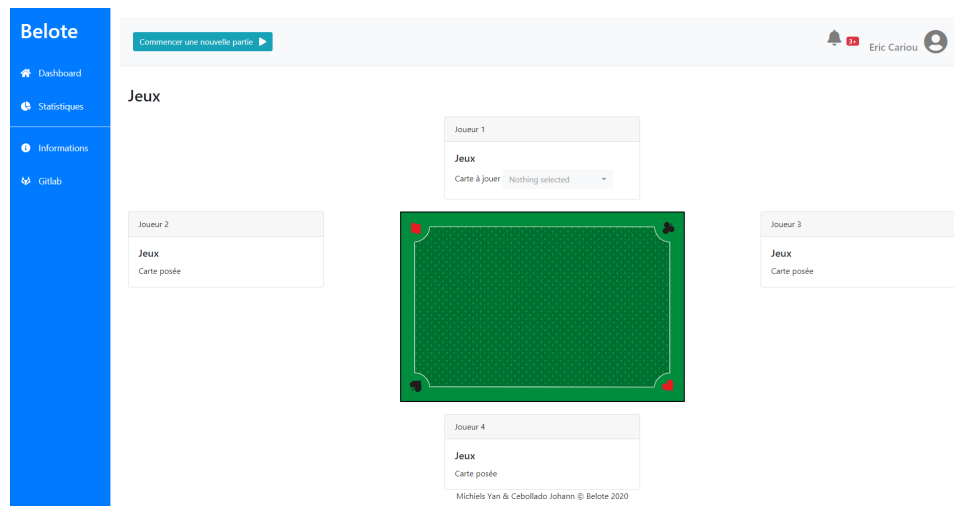


FIGURE 10 – Page du jeu pour le client

0.8 Serveur

Pour notre projet, nous utilisons la version 9 de Tomcat.

0.9 JSP

Pour la génération de code HTML, nous avons séparé les parties HTML de chaque pages pour rendre le code le plus modulaire possible.

0.10 WebSocket

La communication entre les utilisateurs et les serveurs se fait au travers des WebSockets. Ces ceruelet transmettent les classes des sous-package de **message** serializier en Json autravers de. Ces WebSockets sont définis dans les classes :

- **ws.WebSocketPlayer** : un Websocket sur le port **/ws/partie** chargé d’informer les utilisateurs de la liste des autres utilisateurs connecté ainsi que d’informé les utilisateurs.
- **ws.WebSocketUsers** : un Websocket sur le port **/ws/users** utilisé uniquement durant les parties pour transmettre l’état du tapis et la les des cartes. Les messages transmis par le servlet sont déclaré dans les packages :
 - **message.partie.serveur** contenant la liste des messages émit par le serveur en destination du client.
 - **message.partie.client** contenant la liste des messages émit par le client en destination du serveur.

Les messages dans le packages **message.setup** étaient destiner à la mise en place d’un protocole d’invitation qui a du être coupé par manque de temps et de moyens.

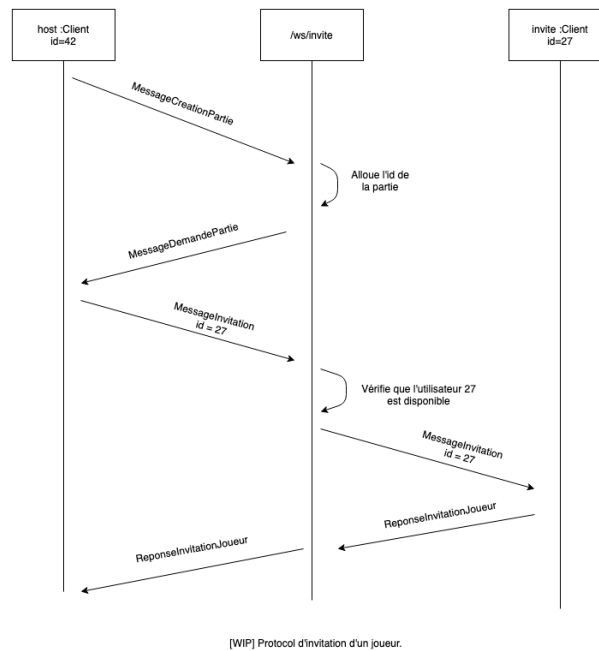


FIGURE 11 – Prototypé d'un protocole permettant a un utilisateur d'en inviter un autre

0.11 Gestion de partie

La partie est gérée par des composants logiques représentant les joueurs au travers de la classe abstraite **logic.joueur.Joueur** qui définit des méthodes utilisables pour envoyer les mises à jour de l'état de la partie au client et de récupérer les actions du joueur. Comme les interactions des joueurs sont des actions asynchrones, la classe **Future<T>** est utilisée pour conserver un semblant de flow séquentiel.

Plus généralement, les classes applicatives **logic.party** appellent des méthodes après chaque modification du tapis. Chaque méthode possède 2 versions, une version retournant des **Future** dans le cas où le joueur est supposé jouer une carte et des méthodes ne retournant rien pour les joueurs ne pouvant pas agir de sorte de les garder informés. Cette organisation permet de gérer efficacement simplement les clients asynchrones tout en concevant un flux d'exécution en apparence synchrone.

0.12 GitLab

Notre code est disponible sur le GitLab [GitLab](#)).

UPPA
Avenue de l'Université,
64000 Pau