

Ecole Nationale Supérieure  
d'Électrotechnique, d'Électronique, d'Informatique, d'Hydraulique et des Télécommunications  
**Projet de Java EE**

Rapport

# Ingénierie des systèmes

---

Fabien RENAUD  
Romain KASSEL  
Jean-Patrick BALBO  
Mathieu MOREL  
Groupes E/F



# Table des matières

<b>1</b>	<b>Présentation du projet</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.1.1	Objectifs du projet . . . . .	1
1.1.2	Sujet développé . . . . .	1
1.2	Choix technologiques . . . . .	2
1.2.1	Langages . . . . .	2
1.2.2	Serveurs . . . . .	2
1.2.3	Logiciels . . . . .	3
<b>2</b>	<b>Prérequis</b>	<b>3</b>
2.1	Dépendances et nettoyage du système . . . . .	3
2.2	Maven & MySQL . . . . .	3
2.3	NetBeans & GlassFish . . . . .	4
2.3.1	Installation . . . . .	4
2.3.2	Configuration . . . . .	4
2.3.3	Test . . . . .	5
<b>3</b>	<b>Conception</b>	<b>7</b>
3.1	Design du site . . . . .	7
3.1.1	Visiteur . . . . .	8
3.1.2	Membre inscrit . . . . .	8
3.1.3	Administrateur . . . . .	8
3.2	Modèle de données . . . . .	8
3.2.1	Vue d'ensemble . . . . .	8
3.2.2	User . . . . .	9
3.2.3	Bet . . . . .	10
3.2.4	BetChoice . . . . .	10
3.2.5	BetInstance . . . . .	11
3.3	Trois couches . . . . .	11
3.3.1	Vue d'ensemble . . . . .	11
3.3.2	DAO . . . . .	11
3.3.3	Service . . . . .	11
3.3.4	Controller . . . . .	11
<b>4</b>	<b>Implémentation</b>	<b>11</b>
4.1	Modèle de données . . . . .	11
4.1.1	MySQL . . . . .	11
4.1.2	Hibernate & POJO . . . . .	12
4.2	Trois couches . . . . .	16
4.2.1	Exemple de DAO . . . . .	16
4.2.2	Exemple de Service . . . . .	17
4.2.3	Exemple de Controller . . . . .	19
4.3	Spring Security . . . . .	20
4.3.1	En quelques mots... . . . .	20
4.3.2	Beans . . . . .	21
4.3.3	Authentication . . . . .	22
4.3.4	Utilisation des sessions Spring Security . . . . .	22
<b>5</b>	<b>Aperçu du site</b>	<b>23</b>
5.1	Mode déconnecté . . . . .	23
5.1.1	Vue d'ensemble . . . . .	23
5.1.2	Inscription . . . . .	24
5.1.3	Accès aux paris . . . . .	25
5.2	Mode connecté . . . . .	25
5.2.1	Vue d'ensemble . . . . .	25

5.2.2	Profil utilisateur . . . . .	26
5.2.3	Accès aux paris . . . . .	26
5.3	Côté administrateur . . . . .	27
5.3.1	Vue d'ensemble . . . . .	27
5.3.2	Gestion des utilisateurs . . . . .	28
5.3.3	Gestion des paris . . . . .	29
5.3.4	Gestion des transactions . . . . .	29
<b>6</b>	<b>Conclusion</b>	<b>30</b>
6.1	Difficultés rencontrées . . . . .	30
6.2	Limites et améliorations . . . . .	31
6.3	Apports personnels . . . . .	31
<b>A</b>	<b>Sources et liens</b>	<b>32</b>
A.1	Général . . . . .	32
A.2	Maven . . . . .	32
A.3	Spring . . . . .	32
A.4	Hibernate . . . . .	32
<b>B</b>	<b>Annexe</b>	<b>33</b>
B.1	Base de données . . . . .	33
B.1.1	Vue d'ensemble . . . . .	33
B.1.2	Table BankTransaction . . . . .	33
B.1.3	Table Bet . . . . .	34
B.1.4	Table BetChoice . . . . .	34
B.1.5	Table BetInstance . . . . .	34
B.1.6	Table Category . . . . .	35
B.1.7	Table PaymentMethod . . . . .	35
B.1.8	Table SiteAccount . . . . .	36
B.1.9	Table Transaction . . . . .	37
B.1.10	Table User . . . . .	37
B.1.11	Clés étrangères . . . . .	38
B.1.12	Contenu minimal . . . . .	39
B.2	Couche modèle - Entités . . . . .	39
B.2.1	Bet . . . . .	39
B.2.2	BetChoice . . . . .	45
B.2.3	BetInstance . . . . .	48
B.2.4	User . . . . .	52
B.2.5	UserRememberMe . . . . .	68
B.3	Data Access Objects . . . . .	69
B.3.1	BankTransactionDao . . . . .	69
B.3.2	BetDao . . . . .	70
B.3.3	BetChoiceDao . . . . .	72
B.3.4	BetInstanceDao . . . . .	73
B.3.5	CategoryDao . . . . .	75
B.3.6	PaymentMethodDao . . . . .	75
B.3.7	SiteAccountDao . . . . .	76
B.3.8	TransactionDao . . . . .	77
B.3.9	UserDao . . . . .	78
B.4	Services . . . . .	80
B.4.1	BankTransactionManager . . . . .	80
B.4.2	BetManager . . . . .	82
B.4.3	CategoryManager . . . . .	90
B.4.4	PaymentMethodManager . . . . .	92
B.4.5	SiteAccountManager . . . . .	94
B.4.6	TransactionManager . . . . .	95
B.4.7	UserManager . . . . .	96

B.5	Controllers . . . . .	101
B.5.1	HomeController . . . . .	101
B.5.2	RegistrationController . . . . .	103
B.5.3	SectionController . . . . .	105
B.6	Java Server Pages . . . . .	106
B.6.1	Header . . . . .	106
B.6.2	Footer . . . . .	107
B.6.3	Index . . . . .	107
B.6.4	In-Login . . . . .	108
B.6.5	Out-Login . . . . .	108
B.6.6	Registration . . . . .	109



# 1 Présentation du projet

## 1.1 Introduction

### 1.1.1 Objectifs du projet

Le projet d'ingénierie des systèmes consiste en la création d'une "petite" application de commerce électronique basée sur Java EE (anciennement nommé J2EE).

Ce projet est à réaliser en groupe de quatre sur une durée d'environ sept semaines, celui-ci comporte plusieurs objectifs.

Le premier, de poursuivre, et au-delà, le travail effectué en travaux pratiques d'applications internet afin que chacun consolide ses connaissances dans les diverses technologies utilisées, survolées, voire non enseignées et apprises sur le tas durant les travaux pratiques. Cela permettra ainsi à chacun de bien appréhender comment un site d'e-commerce peut se concevoir et quelles technologies peuvent être déployées pour y parvenir.

Le second objectif est clairement un objectif d'organisation et de gestion de projet. En effet, aux vues du temps mis à disposition pour réaliser ce projet nécessitant l'utilisation de nombreuses technologies différentes et ayant toutes leurs particularités et aux vues de nos agendas, effectuer ce projet sans avoir une certaine logistique établie garantirait une quasi impossibilité de le terminer dans les temps impartis...

Le but du projet n'est pas de réaliser un gros site pouvant concurrencer d'éventuels acteurs importants du marché actuel. Ses ambitions sont plus modestes.

Il est simplement demandé de fournir/réaliser une application d'e-commerce proposant des fonctionnalités caractéristiques de type *compte utilisateur*, *panier d'achats*, *interface administrateur*, etc. Le nombre de fonctionnalités reste libre et aucune contrainte n'est véritablement fixée.

De plus, il n'est pas demandé à ce que le site soit déployé dans un cadre réel. Autrement dit, tous les tests seront et resteront locaux !

Les seuls véritables contraintes de ce projet sont les suivantes :

- faire un site d'e-commerce basé sur Java EE.
- travailler en équipe.
- avoir une expérience utilisateur riche (*ie.* interface ergonomique et plaisante).
- trouver un thème/sujet de site d'e-commerce (*ie.* quoi vendre?).

### 1.1.2 Sujet développé

C'est dans cette optique que J.P. Balbo a proposé que l'équipe s'attèle à la réalisation d'un site de paris en ligne...

Après discussion, nous sommes tombés d'un commun accord sur ses principales caractéristiques :

- le site ne sera développé que pour une seule langue qui sera bien entendu le français
- le site comportera trois catégories principales : Sports, TV et Actualités. Chacune de ses catégories comportera diverses sous-catégories.
- un visiteur non inscrit/non connecté sur le site pourra :
  - ◇ voir tous les paris proposés par le site
  - ◇ remplir une sorte de panier, que l'on appellera *coupon*, de divers paris
  - ◇ accéder aux conditions générales de ventes (CGV) et aux mentions légales du site
  - ◇ contacter un webmaster du site
- un visiteur connecté, que l'on appellera *membre*, pourra :
  - ◇ faire tout ce qu'un visiteur non membre peut faire
  - ◇ déposer de l'argent sur son compte membre
  - ◇ valider (*ie.* payer) les paris placés dans son *coupon*
  - ◇ modifier ses informations personnelles et bancaires
  - ◇ accéder à tout instant à un historique de ses paris
  - ◇ accéder à tout instant à la liste de ses paris en cours sur le site
  - ◇ proposer éventuellement des paris dans une zone spéciale du site prévue à cet effet

## 1.2 Choix technologiques

### 1.2.1 Langages

Face à la multitude de langages de programmation/présentation disponibles et permettant de réaliser des applications internet, que choisir ?

Hormis le fait que ce projet doivent se faire avec Java EE, reste à choisir tous les langages allant autour, leurs variantes, leurs versions, les frameworks qui y sont adaptés, etc.

Dans la mesure où nous n'avons aucune véritable contrainte commerciale et que ce projet a pour but premier d'améliorer nos connaissances en technologies webs, certains langages encore non officialisés ont été choisis aux vues des améliorations significatives qu'ils apportent déjà aux développeurs, tout en apportant une meilleure expérience utilisateur.

C'est ainsi que, bien que le choix eusse pu se porter sur XHTML 1.1 au lieu de HTML 4.01 pour le langage de présentation des pages, ce fut finalement HTML 5 qui fut choisi.

Cette nouvelle mouture d'HTML propose en effet des fonctionnalités (*ie.* tags) très intéressantes, telles :

- les Scalable Vector Graphics (SVG) intégrés dans le Document Object Model (DOM)
- balises de dessin scriptable (**canvas**)
- balises améliorant le référencement telles **header**, **footer** pour les entêtes et pieds de page, **nav** pour les menus, **article** pour les zones de contenus, etc.
- support natif du drag & drop, de supports multimédias **video** et **audio**
- support des primitives HTTP PUT, GET et DELETE dans les formulaires
- *background workers*
- *application cache*
- géolocalisation, etc.

Dans un même souci de simplification du code et d'exploitation des dernières technologies, ce fut CSS *level 3* qui fut choisi pour implémenter le design des pages. Cette version de CSS propose notamment par rapport à la 2.1 les nouveautés suivantes :

- bordures arrondies, bordures d'images
- arrières-plans multiples (un seul **div** peut contenir plusieurs arrières-plans superposables)
- ombres de textes et de boîtes (*ie.* **text-shadow** et **box-shadow**)
- polices de font embarqués (**@font-face**)
- transitions et transformations 2D, etc.

Ce langage sera massivement exploité de sorte à réduire au maximum le code HTML (*ie.* utilisation de sélecteurs CSS plutôt que de classes).

HTML 5 et CSS 3 n'étant bien supportés que sous très peu de navigateurs pour l'instant, les meilleurs étant *Opera*, *Safari* et *Google Chrome*, notre site n'aura pour objectif que d'être utilisable/affichable correctement que sur ces navigateurs, voire sur un seul d'entre eux.

Pas d'Internet Explorer donc...

Quant aux frameworks, ce sont Spring et jQuery qui ont été retenus respectivement pour la couche métier Java EE et le JavaScript, ceux-ci offrant un haut niveau d'abstraction et des fonctionnalités simplifiant considérablement l'écriture et améliorant la lisibilité du code.

### 1.2.2 Serveurs

Tomcat, JBoss, Geronimo, GlassFish, **dm Server** ? Lequel de ces serveur open source choisir pour notre site web ?

Alors que **Tomcat** allie légèreté, performance et simplicité de configuration mais n'est qu'un serveur de servlets, **JBoss** a pour objectif de fournir un *application server* tout compris. De son côté **Geronimo** est le seul serveur d'application certifié Java 5 et **GlassFish** et **dm Server** sont tellement récents (respectivement 3 ans et 2 ans pour leur date de parution) que les informations à leur sujet sont encore fragmentaires.

Notre choix s'est très rapidement limité à deux possibilités : **Tomcat** ou **GlassFish**. Ceux-ci sont en effet facilement installables et intégrables à notre environnement de développement comme nous le verrons plus loin.

Notre premier choix fut **Tomcat** : simple, léger et le plus performant des serveurs listés ci-dessus, la documentation à son propos est facilement trouvable et nous l'avions même configuré à un moment pour l'intégrer à Apache 2.

Cependant, **Tomcat** n'est qu'un serveur de servlets et ne gère pas par conséquent les **JTA** (Java Transaction API) ou les **JPA** (Java Persistence API), ce dont nous avons besoin pour notre projet. Notre choix s'est donc finalement tourné vers **GlassFish**, serveur sous *license CDDL* et basé sur **Sun Java System Application Server**.

Quant au serveur de base de données, la décision de prendre **MySQL** fut rapidement prise. **MySQL** étant en effet le système de gestion de bases de données (SGBD) probablement le mieux documenté du web en raison de son succès et de sa popularité.

Les outils associés sont également fortement pratiques.

### 1.2.3 Logiciels

Pour mener à bien le développement du site, ce fut **NetBeans** dans sa version 6.9 qui fut choisi. En effet, cette version, en plus de la traditionnelle simplicité d'utilisation de l'EDI, permet de gérer facilement des projets **Maven** et intègre totalement les dernières versions de **Spring** et **Hibernate**, ce que nous voulions, avec de nombreux avantages et aides au développement.

De plus cette version propose lors de l'installation d'installer et configurer automatiquement les serveurs **Tomcat** et/ou **GlassFish** dans leur dernière version.

Côté gestion de projet, nous avons choisi d'utiliser **Maven** plutôt que **Ant** pour la raison principale que **Maven** semblait apporter une simplification importante lors de la compilation du projet, sans avoir besoin de trop écrire de fichiers de configuration éparpillés pour chaque étape de compilation.

**phpMyAdmin** fut également installé pour gérer de façon graphique et donc simple la base de données **MySQL**.

## 2 Prérequis

### 2.1 Dépendances et nettoyage du système

Toutes les installations et configurations détaillées ci-après sont des installations manuelles. Les gestionnaires d'installation et de configuration Linux ne sont pas utilisés ici.

Toutes les explications et codes fournis ont été testés pour *Ubuntu 10.04 LTS - the Lucid Lynx*.

Seul prérequis : avoir la dernière version de Java installée ainsi que son JDK. Sous ubuntu, il s'agit par défaut du package `openjdk`. Pour installer les packages Ubuntu Java nécessaires, ouvrez un terminal et entrez :

```
# Supprime toutes les versions de Java provenant des packages Sun.
sudo apt-get --purge remove sun-j*
# Installe le JRE, le JDK et la documentation pour Java 6
sudo apt-get install openjdk-6-jre openjdk-6-jdk openjdk-6-doc
```

### 2.2 Maven & MySQL

Installons tout en même temps : **Maven**, le serveur **MySQL** et **phpMyAdmin**. Dans un terminal :

```
sudo apt-get install maven2 mysql-server phpmyadmin
```

Remarque : Installer **phpMyAdmin** nécessite bien entendu d'avoir PHP d'installé... Les dépendances étant gérées automatiquement, la ligne de commande précédente revient à installer une solution **LAMP** (Linux, Apache, MySQL, PHP) et **Maven**.

Lors de l'installation de MySQL, choisir comme mot de passe *botv* pour le mot de passe *root*.

## 2.3 NetBeans & GlassFish

### 2.3.1 Installation

Téléchargez la dernière version de **NetBeans** (la 6.9 lors de l'écriture de ce rapport) sur le site officiel :

<http://netbeans.org/>

Choisissez la version comprenant au moins *Java Web and EE* et les serveurs **Tomcat** et **GlassFish**.

Une fois téléchargé, rendez le fichier exécutable (`chmod +x ...`) et exécutez le en mode super-utilisateur. Suivez alors les étapes d'installation. Lorsque vous arriverez sur la page vous présentant les différents modules à installer, veillez à ce que au moins *Java SE* et *Java Web and EE* soient cochés. N'installez pas **GlassFish** ici! Nous le ferons plus tard...

Installez **NetBeans** dans le dossier `/usr/local/share/netbeans6.9` par exemple...

### 2.3.2 Configuration

Une fois l'installation terminée, lancez **NetBeans**. Nous allons à présent configurer les serveurs **GlassFish** et **MySQL** pour pouvoir les utiliser facilement depuis **NetBeans**.

Commencez par ouvrir un terminal. Nous allons créer le dossier où nous allons installer **GlassFish**. Entrez :

```
# Création des dossiers
sudo mkdir /usr/local/share/glassfish
sudo mkdir /usr/local/share/glassfish/v3
# Droits utilisateur actuel sur les dossier créés
sudo chown -R $USER:$USER /usr/local/share/glassfish
```

Ensuite dans **NetBeans**, allez dans le menu **Tools > Servers** et ajoutez un serveur **GlassFish 3**.



FIGURE 1 – Créez un serveur **GlassFish**

Choisissez ensuite pour répertoire d'installation `/usr/local/share/glassfish/v3`. Acceptez la licence et cliquez sur le bouton de téléchargement.

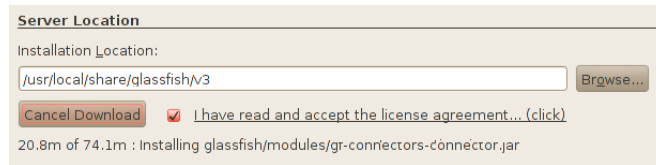


FIGURE 2 – Choisissez le dossier d'installation

Sur la page suivante, choisissez **bet-on-tv** pour nom de domaine et terminez l'installation du serveur. Il ne vous reste plus qu'à renseigner dans l'onglet **Java** le chemin vers la JVM. sous Ubuntu 10.04, par défaut : `/usr/lib/jvm/default-java/bin/java`.

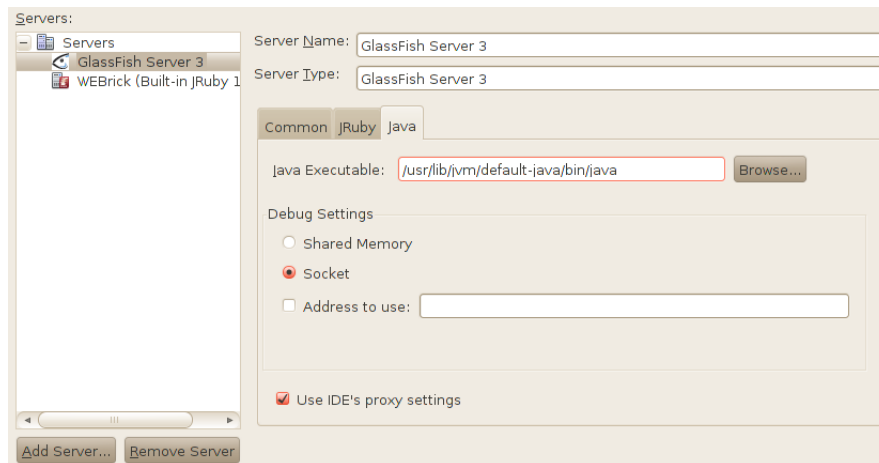


FIGURE 3 – Réglez le chemin vers la JVM

Votre serveur **GlassFish** est prêt à présent.

### 2.3.3 Test

Nous allons pouvoir tester à présent très rapidement le bon fonctionnement de nos installations. En quelques mots : nous allons créer une nouvelle application web via **maven** et la tester sur notre serveur **GlassFish** en ajoutant déjà éventuellement quelques dépendances **Spring** et **Hibernate** pour faire travailler un peu **maven** (ie. téléchargement des dépendances requises).

En quelques images, cela donne :

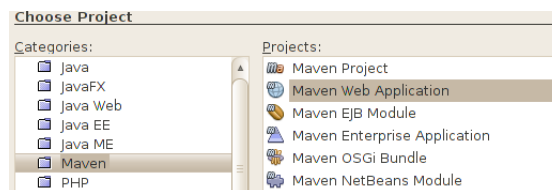


FIGURE 4 – Créez une nouvelle **Maven Web Application**

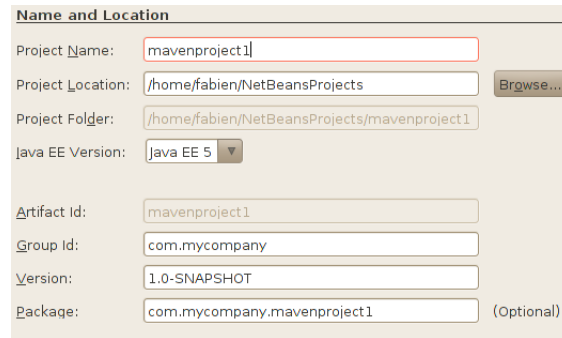


FIGURE 5 – Choisissez la version 5 de Java EE

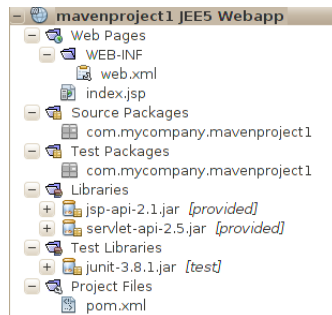


FIGURE 6 – Maven génère un projet de type **web application** avec une arborescence standard



FIGURE 7 – Dans les propriétés du projet nouvellement créé, choisissez quel serveur utiliser...

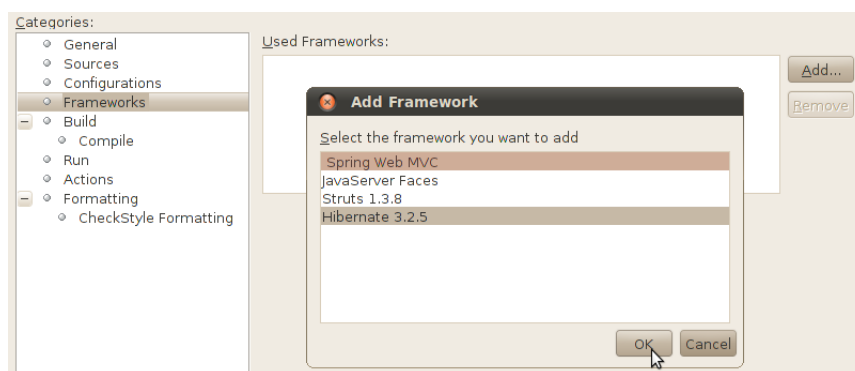


FIGURE 8 – Toujours dans les propriétés du projet, ajoutez les dépendances dont nous aurons besoin plus tard...

Il suffit à présent de lancer le projet (**Run Project** ou **F6**). Maven va alors télécharger les dépendances manquantes (cela peut durer quelques minutes), compiler toutes les sources, lancer les éventuels test **JUnit** trouvés puis **NetBeans** lancera **GlassFish** et ouvrira directement votre navigateur favori. Vous verrez alors se

dessiner la page suivante, indiquant que tout fonctionne correctement.

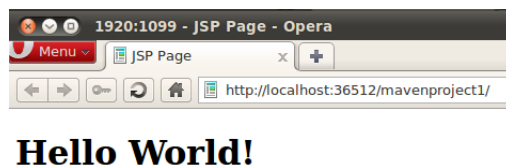


FIGURE 9 – Hello World !

## 3 Conception

### 3.1 Design du site

Il est très important d'accorder au design d'un site une attention toute particulière. En effet, la première chose qu'un utilisateur va juger en arrivant sur un site, c'est son apparence. Au-delà donc du design de notre site qui sera présenté dans la partie 5, nous avons voulu que le site soit fonctionnel avec des zones clairement identifiables par l'utilisateur.

Voici donc, de manière schématique, la façon dont nous avons agencé notre site :

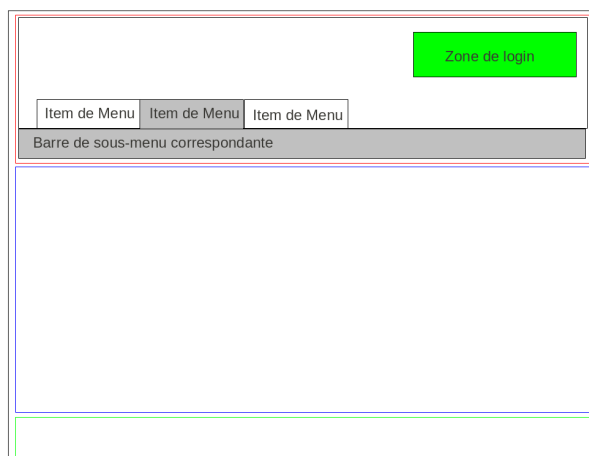


FIGURE 10 – Schéma fonctionnel du site

On remarque donc plusieurs parties :

- Le cadre rouge correspond au **header**. C'est un élément qui reste de manière permanente, quelque soit la page sur laquelle se trouve le visiteur. On y retrouve la zone de login qui permet à l'utilisateur de s'identifier puis éventuellement de se déconnecter. Dans le bas du **header**, on trouve également l'espace permettant la navigation dans le site. Elle est composée d'une barre de menu et lorsque l'on clique sur un élément de ce menu, le sous-menu correspondant s'affiche juste en dessous.
- Le cadre bleu correspond au corps de la page. Son contenu va changer en fonction de la navigation de l'utilisateur et en fonction du fait qu'il soit connecté ou pas.
- Enfin le cadre vert représente le **footer** de la page. Tout comme le **header**, c'est un élément permanent. Il contient les liens vers les Conditions Générales d'Utilisation, vers les Mentions Légales ainsi que vers la page pour contacter les webmasters du site.

### 3.1.1 Visiteur

On appelle visiteur un utilisateur qui, bien sûr, visite notre site, mais surtout qui n'est pas connecté. Un visiteur a donc accès à toute la partie publique du site.

Sur la page d'accueil, il peut donc voir dans le corps de la page les différents éléments rappelant les catégories du site. Et dans les différentes parties du site, il a accès à chaque fois à toutes les informations sur le sujet correspondant, ainsi qu'aux différents paris mais sans pouvoir parier.

### 3.1.2 Membre inscrit

Le visiteur a la possibilité de s'inscrire sur le site. Le lien est situé dans la zone de login. Il a alors accès à plusieurs pages de formulaires dont il doit remplir les champs.

Au niveau du design, ces pages sont identiques aux pages publiques : les formulaires sont dans le corps de la page et le **header** et le **footer** sont identiques. Enfin, il y a des pages d'édition de compte : gestion des informations personnelles, gestion des moyens de paiement, gestion des paris en cours. Là encore, ces pages sont conçues de la même manière.

### 3.1.3 Administrateur

C'est dans la partie d'administration du site que se situe la différence. En effet, les pages de cette partie diffèrent du reste du site par leur agencement et leur design. Cette différence marquée est intentionnelle pour bien séparer la partie publique du site de la partie administrative du site.



FIGURE 11 – Schéma fonctionnel de la partie administrative

La partie administrative se décompose comme suit :

- la partie verte est la barre de menu administrative. On peut y choisir les éléments à consulter et/ou éditer. Ces différents éléments sont par exemple les utilisateurs, les paris ou les transactions bancaires.
- ces informations s'affichent dans le corps du texte en gris. Grâce à la partie rouge notamment, il est possible de passer du mode consultation au mode d'édition.
- dans la partie bleue, il est possible de sélectionner les différentes catégories et sous-catégories du site. Ainsi par exemple l'administrateur peut décider de n'afficher que les paris correspondant à un sport en particulier pour les éditer.
- enfin, la partie rouge correspond aux boutons permettant de passer en mode édition des données.

## 3.2 Modèle de données

### 3.2.1 Vue d'ensemble

Le modèle de données que nous avons adopté est un modèle qui permet de prendre en charge toutes les fonctionnalités du site. Voici donc les différentes entités dont nous avons eu besoin pour cela.

- **User**. Un utilisateur, élément central du fonctionnement du site. En effet pour pouvoir parier, il faut être inscrit et connecté sur le site. L'entité utilisateur doit donc contenir toutes les informations personnelles d'un joueur visitant le site, ainsi que des données propres à celui-ci et nécessaires à la gestion du site (dates de dernière connexion, accord ou refus pour recevoir une newsletter, etc.).
- **PaymentMethod**. Un moyen de paiement immédiatement lié à l'utilisateur et que celui-ci doit pouvoir enregistrer pour jouer.
- **Bet**. Second élément incontournable pour le site, il s'agit du pari. Un pari contient toutes les informations présentant un événement sur lequel va pouvoir parier un utilisateur.
- **BetChoice**. Un choix qu'il est possible de faire quant à l'issue d'un pari. Par exemple si un match est représenté par un pari, un choix pourra être le fait qu'une équipe gagne, ou un résultat en nombre de buts, ou bien d'autres encore.
- **BetInstance**. Une instance de pari, elle relie les informations nécessaires pour pouvoir jouer : pour un pari donné, un utilisateur fait un certain choix relatif à ce pari et il le fait à un instant précis, pour un montant donné.
- **Category**. Une des catégories pour laquelle le site propose de parier.
- **SiteAccount**. Un compte qui sert au site à gérer les flux d'argent lorsque les paris se terminent. Il faut débiter les perdants et créditer les gagnants.
- **Transaction**. Une transaction modélisant un flux d'argent entre le compte d'un utilisateur et un compte du site.
- **BankTransaction**. Une transaction bancaire réelle qu'un utilisateur effectue grâce à un moyen de paiement afin de créditer/débiter son compte pour pouvoir jouer ou retirer l'argent gagné.

D'où la représentation graphique des tables de notre base de données et des champs qui la composent décrite figure 12.

On peut ainsi remarquer les liens entre les tables, notamment les clés étrangères présentes dans certaines tables et faisant référence à des clés primaires d'autres tables. Passons en revue plus précisément les différentes tables.

### 3.2.2 User

La table **User** est la table la plus importante de la base de données. Nous aurions pu à la fois la scinder en plusieurs tables plus petites, par exemple en créant une table pour stocker séparément les coordonnées postales, et à la fois la rendre plus importante en incluant par exemple le moyen de paiement d'un utilisateur dans les champs de la table **User**.

Cependant, découper la table en plusieurs petites autres tables aurait complexifié la gestion de la base de données et diminuer les performances — notamment à cause des nouvelles clés étrangères qui auraient dû être définies — mais garder certaines tables à part, telle la table **PaymentMethod**, offre une plus grande souplesse du code et facilite une éventuelle expansion de la base de données.

Ainsi, si nous avons décidé en cours de route que les moyens de paiement n'étaient pas forcément uniques pour un même utilisateur, la base de données n'aurait même pas eu besoin d'être modifiée...

Quelques remarques concernant cette table :

- On constate qu'un utilisateur est identifié par deux champs : son identifiant (**id**) et son login qui doivent être uniques.
- Le mot de passe est crypté en md5, mais laisse la place (64 caractères) à d'autres algorithmes plus sûrs...
- De nombreux champs sont consacrés aux informations postales, téléphoniques et électronique d'un utilisateur.
- Le champ **title** correspond à la civilité d'un utilisateur (0 = Monsieur, 1 = Madame, 2 = Mademoiselle).
- Le champ **balance** représente la somme d'argent dont l'utilisateur dispose pour parier.
- Des informations permettant de gérer le compte de l'utilisateur : les dates de création de compte de dernières modifications et de dernières connexion aident à surveiller l'activité des comptes. Enfin, des entiers indiquent l'état du compte utilisateur (banni, non validé, en cours de validation et validé) et de savoir s'il est d'accord pour recevoir les newsletters du site et des partenaires du site.

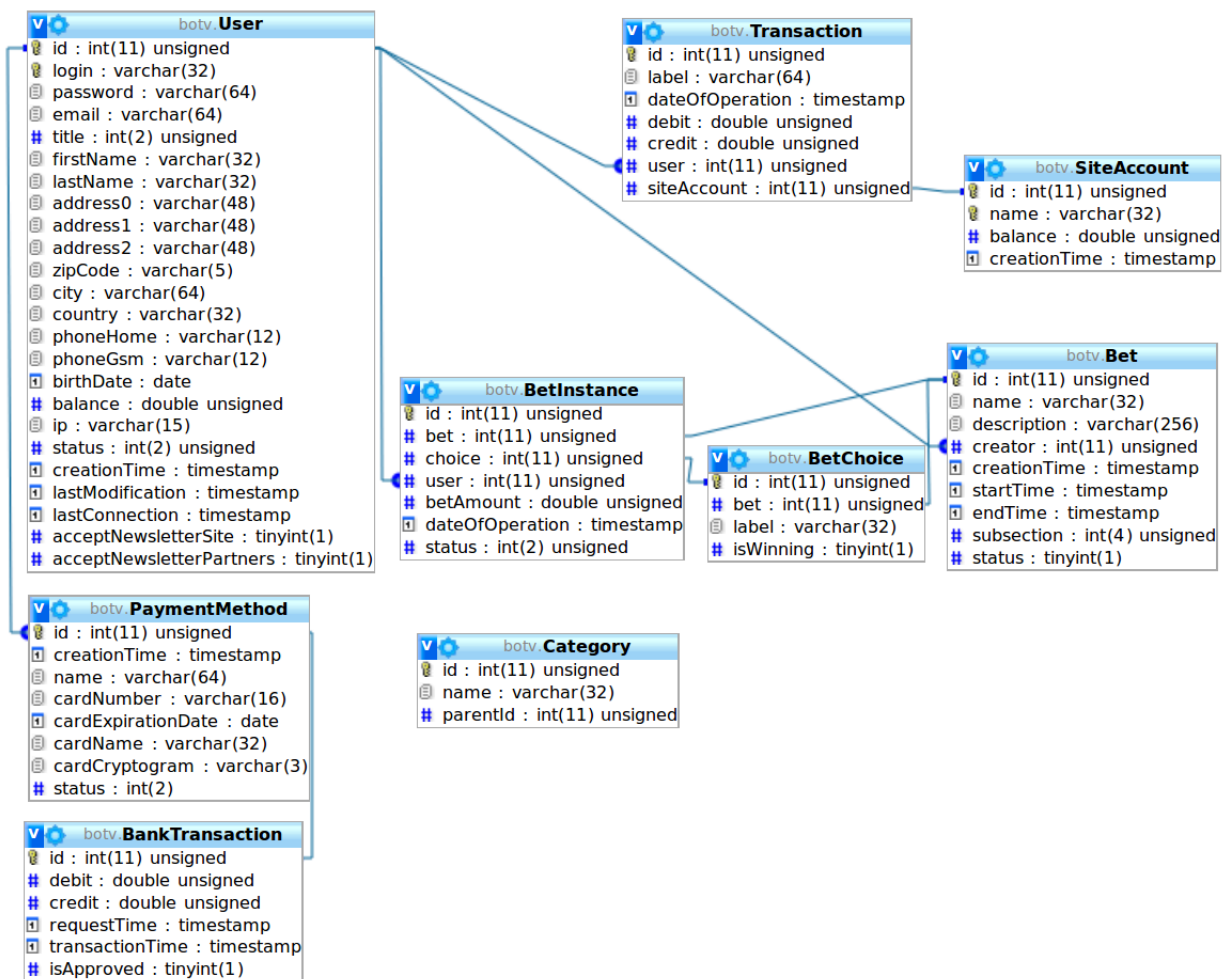


FIGURE 12 – Représentation graphique de la base de données

### 3.2.3 Bet

Chaque pari est enregistré dans la table **Bet**, identifié donc de manière unique par un identifiant (**id**), mais repéré également par un nom et une description. Le champs subsection correspond à la catégorie dans laquelle nous avons classé le pari.

Par exemple, un pari aurait pour nom "Match de football entre A et B", sa description préciserait dans le cadre de quelle compétition ou de quel championnat il a lieu et la *subsection* serait alors celle correspondant au football.

La date de création correspond à la date à laquelle le pari a été enregistré dans la base de données, les dates de début et de fin marquent les dates d'ouverture et de clôture du pari.

Le champs **creator** renseigne l'utilisateur ayant créé le pari, pour la tracabilité. . .

### 3.2.4 BetChoice

Une solution pour préciser les issues des paris aurait pu être d'avoir 2 champs dans la table **Bet** : *champ1* et *champ2*. Mais cela nous aurait limité à des paris à deux choix possibles uniquement. . .

Là encore, pour plus de souplesse, nous avons séparé les issues possibles d'un pari. Un pari peut donc avoir 2 choix ou plus (il n'y a pas de limite). Un choix de pari est identifié de manière unique par un numéro, il contient l'identifiant du pari auquel il est lié et l'intitulé lui correspondant.

À la fin d'un pari, on place le champs **isWinning** du choix vainqueur à **true**.

### 3.2.5 BetInstance

Une instance de pari correspond réellement au jeu d'un joueur. Identifiée bien sûr de manière unique, elle met en relation un pari (`id`), un choix de pari (`id`), l'utilisateur qui fait le pari (`id`), la date à laquelle a été fait le pari et la somme mise.

Le champs status vaut 0 si le pari est en cours, 1 si le pari est terminé et 2 s'il est annulé.

## 3.3 Trois couches

### 3.3.1 Vue d'ensemble

La partie métier du code que nous avons développé se décompose en 3 couches :

- Les **Controller** : ils gèrent les requêtes venant des pages du site, lancent les éventuelles exécutions nécessaires en passant par les **Service** puis en récupèrent les résultats et gèrent les redirections vers les pages du site.
- Les **Service** : appelés depuis les **Controller**, ils effectuent les opérations demandées en faisant appel aux DAO pour les accès simples à la base de données.
- Les DAO : ils sont les opérateurs pour les échanges avec la base de données. Ils permettent d'effectuer toutes les opérations basiques nécessaires.

### 3.3.2 DAO

Les DAO constituent la couche d'échange avec la base de données. Ils permettent donc d'effectuer toutes les opérations de sélection selon des critères multiples, d'insertion et de modification pour les données présentes dans la base. Pour chaque table de notre base de donnée, il y a donc un DAO associé.

### 3.3.3 Service

Les **Service** sont la couche supérieure aux DAO. Ils les utilisent pour effectuer les tâches plus complexes qui peuvent leur être demandées. Nous n'avons codé que 7 **Service** car le **Service** gérant les paris fait appel aux 3 DAO suivants : `BetDAO`, `BetChoiceDAO` et `BetInstanceDAO`.

### 3.3.4 Controller

Les **Controller** gèrent directement les requêtes venant des pages web du site. Ils assurent la redirection vers les pages JSP et récupèrent les paramètres éventuellement passés en méthode POST ou en méthode GET afin d'effectuer les tâches nécessaires à l'affichage des pages demandées. En Effet, il peuvent passer des paramètres de retour aux pages JSP qui pourront ainsi les traiter pour les afficher.

## 4 Implémentation

### 4.1 Modèle de données

#### 4.1.1 MySQL

Côté base de données, les différentes tables mentionnées ci-avant ont été directement réalisées en MySQL et de manière graphique grâce à `phpMyAdmin`.

Les transactions et clés étrangères n'existent pas dans le moteur par défaut de MySQL (`MyISAM`). Pour un tel support, il faut choisir le moteur `InnoDB`, moteur qui respecte également les contraintes `ACID`.

Ci-dessous, un équivalent de code MySQL pour créer certaines de ces tables.

Listing 1 – Bet.sql

```
1 CREATE TABLE IF NOT EXISTS 'Bet' (  
2   'id' int(11) unsigned NOT NULL AUTOINCREMENT,  
3   'name' varchar(32) COLLATE utf8_bin NOT NULL,  
4   'description' varchar(256) COLLATE utf8_bin NOT NULL,  
5   'creator' int(11) unsigned NOT NULL,
```

```

6  'creationTime' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
7  'startTime' timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
8  'endTime' timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
9  'subsection' int(4) unsigned NOT NULL,
10 'status' tinyint(1) NOT NULL DEFAULT '0',
11 PRIMARY KEY ('id'),
12 KEY 'creator' ('creator')
13 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1 ;

```

Listing 2 – BetInstance.sql

```

1 CREATE TABLE IF NOT EXISTS 'BetInstance' (
2   'id' int(11) unsigned NOT NULL AUTO_INCREMENT,
3   'bet' int(11) unsigned NOT NULL,
4   'choice' int(11) unsigned NOT NULL,
5   'user' int(11) unsigned NOT NULL,
6   'betAmount' double unsigned NOT NULL,
7   'dateOfOperation' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
8   'status' int(2) unsigned NOT NULL DEFAULT '0',
9   PRIMARY KEY ('id'),
10  KEY 'bet' ('bet'),
11  KEY 'user' ('user'),
12  KEY 'choice' ('choice')
13 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1 ;

```

Remarque : Les autres tables sont fournies en annexe.

On réalise également quelques insertions par défaut, touchant notamment aux comptes liés au site.

Listing 3 – Insertions de base

```

1 INSERT INTO 'User' ('id', 'login', 'password', 'email', 'title',
2 'firstName', 'lastName', 'address0', 'address1', 'address2', 'zipCode',
3 'city', 'country', 'phoneHome', 'phoneGsm', 'birthDate', 'balance', 'ip',
4 'status', 'creationTime', 'lastModification', 'lastConnection',
5 'acceptNewsletterSite', 'acceptNewsletterPartners') VALUES
6 (1, 'admin', MD5('admin'), 'admin@site.fr', 0, 'Admin', 'Strateur',
7 'Admin & Co Ltd.', '1, Avenue de la Gestion', '78e étage, bureau n°1',
8 '75000', 'Paris', 'France', '0110203040', '', '1980-05-01', 0, '127.0.0.1',
9 2, '2010-06-01 00:00:00', '2010-06-13 00:00:00', '2010-06-17 11:04:34', 0, 0);
10
11 INSERT INTO 'SiteAccount' ('id', 'name', 'balance', 'creationTime') VALUES
12 (1, 'Compte Paris', 0, '2010-06-06 04:41:49'),
13 (2, 'Compte Taxe Site', 0, '2010-06-10 18:58:49'),
14 (3, 'Compte Taxe État', 0, '2010-06-10 18:58:49'),
15 (4, 'Compte Bonus', 0, '2010-06-10 15:34:01');

```

#### 4.1.2 Hibernate & POJO

Une fois notre base de données en place, il est très facile avec **NetBeans** — et les outils **Hibernate** que celui-ci nous fournit — de générer les objets **Java** à partir de nos tables **MySQL**.

En effet, une fois notre fichier **hibernate.cfg.xml** en place et la connexion à la base de données opérationnelle depuis **NetBeans**, il suffit d'effectuer les opérations suivantes :

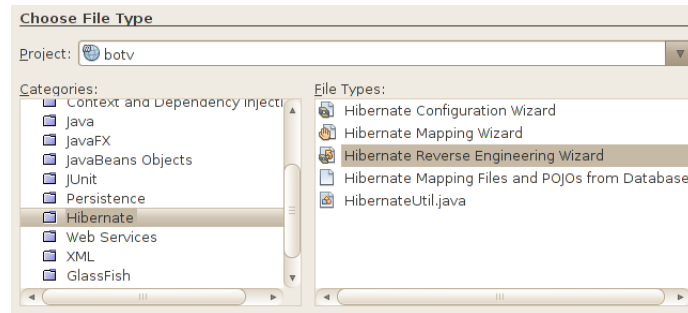


FIGURE 13 – Choisir Hibernate Reverse Engineering Wizard

1. File >New File... [Ctrl + N].  
Dossier **Hibernate** >**Hibernate Reverse Engineering Wizard**.  
Next. Next.
2. Sur la bage **Database Tables**, on peut choisir les tables à importer ou non. Ici, nous allons tout importer.

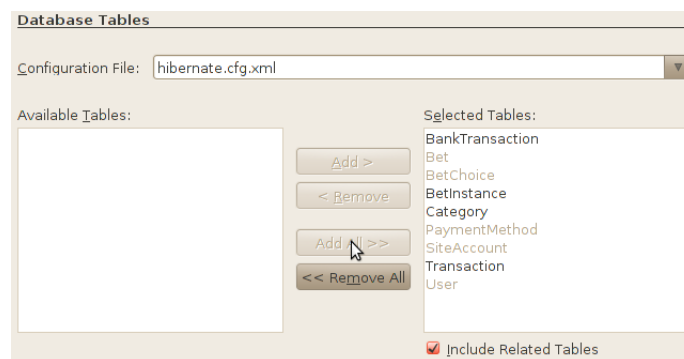


FIGURE 14 – Importer toutes les tables

Finish.

NetBeans nous génère alors un fichier `hibernate.reveng.xml` listant les tables à importer.

Listing 4 – hibernate.reveng.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-reverse-engineering PUBLIC
3 "–//Hibernate/Hibernate Reverse Engineering DTD 3.0//EN"
4 "http://hibernate.sourceforge.net/hibernate-reverse-engineering-3.0.dtd">
5 <hibernate-reverse-engineering>
6   <schema-selection match-catalog="botv" />
7   <table-filter match-name="BetChoice" />
8   <table-filter match-name="Bet" />
9   <table-filter match-name="BankTransaction" />
10  <table-filter match-name="Category" />
11  <table-filter match-name="BetInstance" />
12  <table-filter match-name="SiteAccount" />
13  <table-filter match-name="User" />
14  <table-filter match-name="Transaction" />
15  <table-filter match-name="PaymentMethod" />
16 </hibernate-reverse-engineering>

```

À partir de là, on peut générer tous les objets Java en quelques clics :

1. File >New File... [Ctrl + N].  
Dossier **Hibernate** >**Hibernate Mapping Files and POJOs from Database**.  
Next.
2. On arrive alors sur une page proposant différentes options : annotations EJB3, généricité (Java 5), génération des fichiers de mapping XML, génération des fichiers Java, etc. Dans notre cas, on va tout faire et choisir pour package de destination **com.boc.botv.model**.

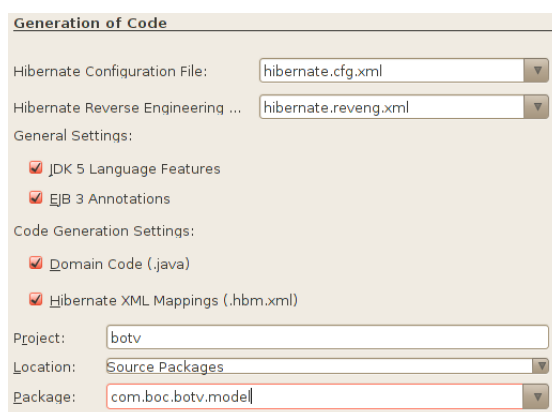


FIGURE 15 – Importer toutes les tables

**NetBeans** nous génère alors autant de fichiers **Java** qu'il y a de classes et ces fichiers sont déjà prêts à l'emploi !  
Ci-dessous un exemple d'un tel fichier avec la table/objet **SiteAccount**.

Listing 5 – **SiteAccount.java** généré

```
1  *package com.boc.botv.model;  
2  
3  import java.util.Date;  
4  import java.util.HashSet;  
5  import java.util.Set;  
6  import javax.persistence.CascadeType;  
7  import javax.persistence.Column;  
8  import javax.persistence.Entity;  
9  import javax.persistence.FetchType;  
10 import javax.persistence.GeneratedValue;  
11 import static javax.persistence.GenerationType.IDENTITY;  
12 import javax.persistence.Id;  
13 import javax.persistence.OneToOne;  
14 import javax.persistence.Table;  
15 import javax.persistence.Temporal;  
16 import javax.persistence.TemporalType;  
17 import javax.persistence.UniqueConstraint;  
18  
19 @Entity  
20 @Table(name = "SiteAccount", catalog = "botv", uniqueConstraints =  
21 @UniqueConstraint(columnNames = "name"))
```

```

22 public class SiteAccount implements java.io.Serializable {
23
24     private Integer id;
25     private String name;
26     private double balance;
27     private Date creationTime;
28     private Set<Transaction> transactions = new HashSet<Transaction>(0);
29
30     public SiteAccount() {
31     }
32
33     public SiteAccount(String name, double balance, Date creationTime) {
34         this.name = name;
35         this.balance = balance;
36         this.creationTime = creationTime;
37     }
38
39     public SiteAccount(String name, double balance, Date creationTime,
40         Set<Transaction> transactions) {
41         this.name = name;
42         this.balance = balance;
43         this.creationTime = creationTime;
44         this.transactions = transactions;
45     }
46
47     @Id
48     @GeneratedValue(strategy = IDENTITY)
49     @Column(name = "id", unique = true, nullable = false)
50     public Integer getId() {
51         return this.id;
52     }
53
54     public void setId(Integer id) {
55         this.id = id;
56     }
57
58     @Column(name = "name", unique = true, nullable = false, length = 32)
59     public String getName() {
60         return this.name;
61     }
62
63     public void setName(String name) {
64         this.name = name;
65     }
66
67     @Column(name = "balance", nullable = false, precision = 22, scale = 0)
68     public double getBalance() {
69         return this.balance;
70     }
71
72     public void setBalance(double balance) {
73         this.balance = balance;
74     }
75
76     @Temporal(TemporalType.TIMESTAMP)
77     @Column(name = "creationTime", nullable = false, length = 19)

```

```

78     public Date getCreationTime() {
79         return this.creationTime;
80     }
81
82     public void setCreationTime(Date creationTime) {
83         this.creationTime = creationTime;
84     }
85
86     @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
87         mappedBy = "siteAccount")
88     public Set<Transaction> getTransactions() {
89         return this.transactions;
90     }
91
92     public void setTransactions(Set<Transaction> transactions) {
93         this.transactions = transactions;
94     }
95 }

```

À présent, si l'on veut ajouter des attributs ou getter à ce type de classe et qu'on ne veut pas que cet attribut/getter soit stocké en base de données, il faudra ajouter devant le getter — éventuellement créé pour un attribut mais l'attribut n'est pas obligatoire — l'annotation `@Transient`.

Voir les fichiers de la couche `Model` donnés en annexe pour des exemples.

## 4.2 Trois couches

### 4.2.1 Exemple de DAO

À partir de là, nous allons créer, comme décrit dans la partie *Conception*, les `Data Access Objects` associés à chacune de ces classes.

Ces DAO se décomposent toujours en deux fichiers : les interfaces, qui seront exploitées par la couche `Service`, et leur implémentation. La couche `Service` n'accède jamais directement à ces implémentations !

Exemple de DAO, celui associé à l'objet `SiteAccount` et utilisant les annotations `Spring 2.5`.

Listing 6 – Interface : `SiteAccountDao.java`

```

1  *package com.boc.botv.dao;
2
3  import com.boc.botv.model.SiteAccount;
4  import java.util.List;
5  import org.hibernate.Session;
6
7  public interface SiteAccountDao {
8      public List<SiteAccount> getAccounts();
9      public SiteAccount getAccount(Integer id);
10     public void saveAccount(SiteAccount account);
11     public void updateAccount(SiteAccount account);
12     public void deleteAccount(SiteAccount account);
13 }

```

Listing 7 – Classe : `SiteAccountDaoImpl.java`

```

1  *package com.boc.botv.dao.impl;
2
3  import com.boc.botv.common.Database;

```

```

4 import com.boc.botv.dao.SiteAccountDao;
5 import com.boc.botv.model.SiteAccount;
6 import java.util.List;
7 import org.springframework.stereotype.Repository;
8
9 @Repository("siteAccountDao")
10 public class SiteAccountDaoImpl implements SiteAccountDao {
11
12     public List<SiteAccount> getAccounts() {
13         return (List<SiteAccount>) Database.getSession().
14             createQuery("from SiteAccount").list();
15     }
16
17     public SiteAccount getAccount(Integer id) {
18         return (SiteAccount) Database.getSession().load(SiteAccount.class, id);
19     }
20
21     public void saveAccount(SiteAccount account) {
22         Database.getSession().save(account);
23     }
24
25     public void updateAccount(SiteAccount account) {
26         Database.getSession().update(account);
27     }
28
29     public void deleteAccount(SiteAccount account) {
30         Database.getSession().delete(account);
31     }
32 }

```

Nous reviendrons dans les paragraphes suivants sur l'intérêt de certaines des annotations utilisées ici.

#### 4.2.2 Exemple de Service

Cette couche implémente des objets de plus au niveau, réalisant des opérations plus complexes, et est basée sur la couche des DAO, qu'elle exploite.

Pour rappel : un élément d'une couche peut accéder à plusieurs éléments de la couche inférieure et uniquement via ses interfaces, mais ne doit pas faire d'accès transversal : pas de service appelant un autre service donc, juste des DAO.

Voyons le code du service exploitant le DAO précédent.

Listing 8 – Interface : SiteAccountManager.java

```

1 *package com.boc.botv.service;
2
3 import com.boc.botv.model.SiteAccount;
4 import java.util.List;
5
6 public interface SiteAccountManager {
7     public List<SiteAccount> getAccounts();
8     public SiteAccount getAccount(Integer id);
9     public void createAccount(SiteAccount account);
10    public void changeName(int accountId, String name);
11    public void mergeAccounts(SiteAccount account1, SiteAccount account2, String name);
12 }

```

Listing 9 – Classe : SiteAccountManagerImpl.java

```

1  *package com.boc.botv.service.impl;
2
3  import com.boc.botv.common.Database;
4  import com.boc.botv.dao.SiteAccountDao;
5  import com.boc.botv.model.SiteAccount;
6  import com.boc.botv.service.SiteAccountManager;
7  import java.util.Date;
8  import java.util.List;
9  import org.apache.commons.logging.Log;
10 import org.apache.commons.logging.LogFactory;
11 import org.hibernate.Session;
12 import org.springframework.beans.factory.annotation.Autowired;
13 import org.springframework.stereotype.Service;
14 import org.springframework.transaction.annotation.Transactional;
15
16 @Service("siteAccountManager")
17 public class SiteAccountManagerImpl implements SiteAccountManager {
18
19     private final Log logger = LogFactory.getLog(SiteAccountManagerImpl.class);
20     private SiteAccountDao siteAccountDao;
21
22     @Autowired
23     public void setSiteAccountDao(SiteAccountDao siteAccountDao) {
24         this.siteAccountDao = siteAccountDao;
25     }
26
27     public List<SiteAccount> getAccounts() {
28         logger.info("[SiteAccountManager] Getting all SiteAccount");
29         List<SiteAccount> sal = siteAccountDao.getAccounts();
30
31         return sal;
32     }
33
34     public SiteAccount getAccount(Integer id) {
35         logger.info("[SiteAccountManager] Getting SiteAccount #" + id);
36         SiteAccount sa = siteAccountDao.getAccount(id);
37
38         return sa;
39     }
40
41     public void createAccount(SiteAccount account) {
42         logger.info("[SiteAccountManager] Creating SiteAccount with account");
43         siteAccountDao.saveAccount(account);
44     }
45
46
47     public void changeName(int accountId, String name) {
48         logger.info("[SiteAccountManager] Changing siteAccount name to " + name + " ");
49         SiteAccount account = siteAccountDao.getAccount(accountId);
50         account.setName(name);
51     }
52
53
54     public void mergeAccounts(SiteAccount account1, SiteAccount account2, String name) {
55         logger.info(String.format("[SiteAccountManager] Merging accounts"

```

```

56     + "%1$d and %2$d under name '%3$s'", account1.getId(),
57     account2.getId(), name));
58     SiteAccount sa = new SiteAccount();
59     sa.setName(name);
60     sa.setBalance(account1.getBalance() + account2.getBalance());
61     sa.setCreationTime(new Date());
62     siteAccountDao.deleteAccount(account1);
63     siteAccountDao.deleteAccount(account2);
64     siteAccountDao.saveAccount(sa);
65 }
66 }

```

L'implémentation de notre service utilise donc bel et bien l'interface **SiteAccountDao** et jamais il n'est fait référence à l'implémentation de ce DAO pour instancier l'attribut **siteAccountDao**.

C'est en fait **Spring** qui va faire tout le travail d'instanciation, et ceci automatiquement grâce aux annotations et notamment l'annotation **@Autowired**.

Cette annotation **@Autowired** fonctionne de la manière suivante :

- placée devant la déclaration d'un attribut, elle va indiquer à **Spring** lors de la compilation d'instancier cet attribut avec la classe de même type (ie. nom). Ceci n'est possible que si la classe ciblée a elle-même été annotée avec **@Repository**, **@Service**, **@Controller** ou **@Component**.

Cependant dans notre cas **SiteAccountDao** n'est pas une classe mais une interface. Ininstanciable donc ! C'est pour cette raison que dans lorsque nous avons défini le code de la classe **SiteAccountDaoImpl**, nous avons ajouté en annotation devant cette classe **@Repository('SiteAccountDao')**. De cette façon, **Spring** associe, grâce à un registre, notre objet **SiteAccountDao** de notre service à l'objet **SiteAccountDaoImpl**. L'instanciation se réalise donc !

- placée devant une méthode, le fonctionnement est le même, mais ce sont chacun des paramètres de la méthode qui seront automatiquement instanciés par **Spring**. Pratique donc pour instancier plusieurs attributs "en même temps".

Ce fonctionnement de l'annotation **@Autowired** est identique quelque soit la couche applicative dans laquelle nous puissions être.

Les autres services sont placés en annexe.

### 4.2.3 Exemple de Controller

La couche **Controller** est la plus élevée de notre architecture. C'est elle qui va permettre de créer nos pages et de faire le lien entre les **Java Server Pages** (JSP) et les services.

Lorsque **Spring** trouve une classe précédée de l'annotation **@Controller**, il mémorise cette classe comme pouvant être liée à des URL et tentera de créer un tel lien à chaque fois qu'il trouvera l'annotation **@RequestMapping**, annotation qui peut être placée aussi bien devant la classe elle même que devant une méthode.

Voyons un petit exemple de **Controller** pour mieux en cerner la syntaxe et le fonctionnement.

Listing 10 – LoginController.java

```

1  *package com.boc.botv.web.controller;
2
3  import com.boc.botv.common.Database;
4  import com.boc.botv.service.UserManager;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Controller;
7  import org.springframework.web.bind.annotation.RequestMapping;
8  import org.springframework.web.bind.annotation.RequestMethod;
9  import org.springframework.web.bind.annotation.RequestParam;

```

```

10
11 @Controller
12 public class LoginController {
13
14     @Autowired
15     private UserManager manager;
16
17     @RequestMapping(value = "/login", method = RequestMethod.GET)
18     public void loginSetUp() {
19     }
20
21     @RequestMapping(value = "/login", method = RequestMethod.POST)
22     public String loginProcess(
23         @RequestParam(value = "username", required = true) String login,
24         @RequestParam(value = "password", required = true) String password,
25         @RequestParam(value = "from", required = false) String from) {
26         Database.startTransaction(true);
27         if (from == null || from.equals("")) {
28             from = "/index.htm";
29         }
30         manager.logIn(login, password);
31         Database.closeTransaction();
32         return "redirect:" + from;
33     }
34
35     @RequestMapping(value = "/logout", method = RequestMethod.POST)
36     public String logoutProcess(
37         @RequestParam(value = "from", required = false) String from) {
38         if (from == null || from.equals("")) {
39             from = "/index.htm";
40         }
41         return "redirect:" + from;
42     }
43 }

```

Ici, ce petit controller ne crée qu'une seule page accessible à l'internaute. Il s'agit de la page `/login.htm` (`.htm` car configuré ainsi dans le `web.xml`). C'est en effet la seule page accessible par méthode GET.

Les deux autres méthodes, accessibles en POST servent à gérer des requêtes issues généralement de formulaires ou de requêtes AJAX (toutes en POST évidemment).

## 4.3 Spring Security

### 4.3.1 En quelques mots...

Pour gérer les différents droits d'accès au site (visiteur, membre, administrateur), nous avons décidé d'utiliser **Spring Security**. Décision allant dans la continuité d'exploiter au maximum le framework **Spring**.

Il y a différentes façon de configurer **Spring Security** en vue d'appliquer une authentification basée sur des tests Java (pour accéder à la base de donnée). On peut même directement accéder aux tables de notre base de données avec un simple fichier de configuration **Spring Security** et effectuer toutes les vérifications depuis le fichier XML. Mais cela nous aurait contraint à adopter les schémas de tables de **Spring Security**, ce que nous ne voulions pas.

Mais quelque soit la méthode employée, celle-ci passe immanquablement par un fichier de configuration et quelques filtres à ajouter au fichier `web.xml`.

Voyons donc ces fichiers de configuration d'un peu plus près.

### 4.3.2 Beans

Commençons par le plus court.

Pour pouvoir utiliser **Spring Security** sur notre site web, il faut commencer par ajouter au fichier `web.xml` les filtres suivants :

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Et comme nous allons détailler la configuration de **Spring Security** dans un autre fichier XML pour plus de clareté, il nous faut également ajouter les lignes suivantes au même fichier :

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext.xml, /WEB-INF/security-applicationContext.xml</param-value>
</context-param>
```

**Spring Security** est prêt à être utilisé. Reste plus qu'à faire quelques réglages de base dans le fichier `security-applicationContext.xml` :

Listing 11 – Fichier `security-applicationContext.xml`

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans:beans xmlns="http://www.springframework.org/schema/security"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:beans="http://www.springframework.org/schema/beans"
5   xmlns:util="http://www.springframework.org/schema/util"
6   xsi:schemaLocation="http://www.springframework.org/schema/security
7     http://www.springframework.org/schema/security/spring-security-3.0.xsd
8     http://www.springframework.org/schema/beans
9     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
10    http://www.springframework.org/schema/util
11    http://www.springframework.org/schema/util/spring-util-3.0.xsd">
12
13   <authentication-manager alias="authenticationManager">
14     <authentication-provider>
15       <user-service>
16         <user name="visitor" password="visitor" authorities="ROLE_ANONYMOUS" />
17       </user-service>
18     </authentication-provider>
19   </authentication-manager>
20
21
22   <http>
23     <intercept-url pattern="/css/**" filters="none" />
24     <intercept-url pattern="/js/**" filters="none" />
25     <intercept-url pattern="/admin/**" access="ROLE_ADMIN" />
26     <intercept-url pattern="/profile/**" access="ROLE_USER" />
27     <intercept-url pattern="/in-logout.htm*" access="ROLE_USER" />
28     <intercept-url pattern="/in-login.htm*" access="ROLE_ANONYMOUS" />
29     <intercept-url pattern="/registration.htm*" access="ROLE_ANONYMOUS" />
```

```

30     <intercept-url pattern="/lost-password.htm*" access="ROLE_ANONYMOUS" />
31     <form-login login-page="/login.htm" />
32     <logout logout-url="/logout.htm" />
33     <anonymous />
34 </http>
35 </beans:beans>

```

Le code est relativement clair. Dans ce fichier, on configure un utilisateur par défaut (requis) et surtout, on définit quels droits d'accès sont nécessaires pour accéder à un dossier en particulier.

Les lignes `form-login` et `logout` permettent d'afficher en cas de besoin une page de connexion *standalone*.

#### 4.3.3 Authentication

Il ne reste plus qu'à s'identifier et à passer les bons droits à chaque utilisateur.

Pour cela, rien de plus simple. Une fois les informations de l'utilisateur récupérées de façon tout à fait classique (par méthode POST par exemple), il suffit à partir de ces informations de générer une instance de la classe `RememberMeAuthenticationToken` et d'ajouter cette instance au contexte sécurité.

En Java, cela donne :

```

void login(String login, String password) {
    User user = userDao.getUser(login);
    if (user == null || !encryptPassword(password).equals(user.getPassword())) {
        return;
    }
    // Liste des autorisations de l'utilisateur
    List<GrantedAuthority> ga = new ArrayList<GrantedAuthority>();
    ga.add(new GrantedAuthorityImpl("ROLE_USER"));
    if (user.getId() == 1) { // Spécial admin !
        ga.add(new GrantedAuthorityImpl("ROLE_ADMIN")); // L'admin a donc 2 rôles : admin + user
    }
    // On crée le token
    Authentication result = new RememberMeAuthenticationToken(login, new UserRememberMe(user), ga);
    SecurityContextHolder.getContext().setAuthentication(result); // et on l'ajoute au contexte
}

```

On notera que l'objet passé en paramètre du *token* est l'objet mémorisé en session. Il est donc inutile, voire même dangereux, de stocker une instance complète de la classe `User` (qui contient mots de passe et autres informations sensibles). En effet, seules certaines informations caractéristiques nous intéressent...

#### 4.3.4 Utilisation des sessions Spring Security

L'intérêt d'une telle identification, c'est qu'il est ensuite très facile de récupérer les informations clés (*ie.* stockées dans l'objet `UserRememberMe`) concernant l'utilisateur. Il suffit pour cela d'accéder à l'objet principal, dans un contexte Spring Security, depuis les pages JSP. Par exemple, la zone de déconnexion est gérée côté JSP de la façon suivante :

Listing 12 – Un air de `logout.jsp`

```

1 <%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
2 <%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
3 <%@ page contentType="text/html" pageEncoding="UTF-8"%>
4 <nav id="userlinks">
5     <ul>
6         <sec:authorize ifNotGranted="ROLE_ADMIN">
7             <li>
8                 Bienvenue <sec:authentication property="principal.title" />

```

```

9         <sec:authentication property="principal.firstname"/>
10        <sec:authentication property="principal.lastname"/>
11    </li>
12    <li>Solde : <sec:authentication property="principal.balance"/> euros</li>
13    <li>
14        <a href="/profil/<sec:authentication property="principal.login"/>
15            /index.htm">Mon profil</a>
16    </li>
17 </sec:authorize>
18 <sec:authorize ifAllGranted="ROLE_ADMIN">
19     <li>Bonjour Monsieur l'Administrateur</li>
20     <li><a href="/admin/index.htm" rel="nofollow">Administration</a></li>
21 </sec:authorize>
22 <li>
23     <a href="/logout.htm?from=<c:url value="{currentUrl}"/>">Déconnexion</a>
24 </li>
25 </ul>
26 </nav>

```

Notez bien ici l'utilisation des balises `authentication` pour accéder aux informations de session...

À mi chemin de la création/enregistrement du token et de son utilisation, il est aussi possible d'accéder à la session utilisateur depuis n'importe quel fichier Java. En effet :

```

Authentication a = SecurityContextHolder.getContext().getAuthentication();
UserRememberMe urm = (UserRememberMe) a.getPrincipal();

```

On récupère ainsi un total contrôle sur l'objet en session et on peut alors directement le modifier, comme un objet normal... Les répercussions de ces modifications sont automatiquement transmises à la session.

Par exemple, lorsque qu'un membre joue, le solde de son compte diminue (car il mise de son argent). Cette modification doit se voir immédiatement dans le *header*. D'où :

```

logger.info("[BetManager] Crediting (output) client account");
user.decreaseBalance(amount);
Authentication a = SecurityContextHolder.getContext().getAuthentication();
((UserRememberMe) a.getPrincipal()).setBalance(user.getBalance());

```

## 5 Aperçu du site

### 5.1 Mode déconnecté

#### 5.1.1 Vue d'ensemble

Lorsqu'un nouvel internaute arrive sur notre site de paris sportifs, la page d'accueil lui permet, dès le premier coup d'œil, de connaître les diverses catégories proposées, ainsi que les événements sportifs, d'actualités ou télévisuels qui du moment.

Cette page a donc pour objectif d'attirer l'attention du client vers une des catégories du site, tout en lui révélant l'ensemble des types de paris sur lesquels il peut miser.

Grâce au *header* de la page, il peut directement s'inscrire via un formulaire classique, lequel le guidera jusqu'à la validation de son inscription.

S'il ne souhaite pas s'inscrire tout de suite, il peut parcourir tout le site mais sans parier. Il peut ainsi visualiser la diversité des paris proposés. Si l'internaute est déjà un client, alors, le *header* présent sur toutes les pages en mode déconnecté lui permet de se *logger* par l'usage du pseudo et mot de passe choisis lors de son inscription.



FIGURE 16 – Page d'accueil en mode déconnecté

La présence du *footer* lui donne accès aux informations relatives au site, qui sont :

- les mentions légales
- les conditions générales d'utilisations (CGU)
- l'accès à une page de contact

### 5.1.2 Inscription

L'internaute a accès à la page d'inscription depuis n'importe quelle page du site et est invité à s'inscrire s'il veut miser sur un pari.

Son inscription est obligatoire s'il veut pouvoir ouvrir un compte et parier !

L'inscription se déroule en 3 étapes :

1. Demande des identifiants du client : *pseudo*, *mot de passe* et *adresse de courriel*.

Des fonctionnalités AJAX vérifient en temps réel si le pseudo choisi et l'adresse de courriel entrés ne sont pas déjà utilisés...

FIGURE 17 – Inscription : Étape 1

2. Saisie obligatoire des principales informations postale et civile (*nom*, *prénom*, *adresse*, etc).
3. Acceptation des CGU obligatoire et validation de l'inscription.

Une fois l'inscription terminée, l'utilisateur recevra un email contenant un lien pour qu'il puisse activer son compte. Il pourra alors se connecter avec les informations renseignées lors de l'inscription.

**Données personnelles**

Civilité \* : ☒ Monsieur ☐ Madame ☐ Mademoiselle

Prénom \* :

Nom \* :

Adresse \* :

Code Postal \* :

Ville \* :

Pays \* :

Date de naissance \* : 1970-01-01 ▼

Téléphone fixe :

Téléphone portable :

[< Étape précédente](#) [Étape suivante >](#)

FIGURE 18 – Inscription : Étape 2

**Confirmation**

Conditions Générales d'Utilisation

1. Définitions

Chaque terme débutant ci-après par une majuscule a le sens qui lui est donné dans sa définition qui figure au présent article.  
 "Conditions d'Utilisation" : désigne le présent document.  
 "Services" : désigne l'espace de discussion, appelé forum, ainsi que l'espace encyclopédique, appelé wiki, fournis par bet-on-win aux Utilisateurs et accessibles par le biais du Site.  
 "Site" : désigne le site portail accessible à l'adresse paris-tv.com et/ou à toute autre adresse que bet-on-win lui substituerait ou lui adjoindrait.  
 "Utilisateurs" : désigne toute personne physique ou morale utilisant un ou des Services.

☒ J'ai lu et j'accepte les conditions générales d'utilisation  
☐ J'accepte de recevoir la newsletter du site  
☐ J'accepte de recevoir les newsletters des partenaires de bet-on-tv.com

[< Étape précédente](#) [Terminer l'inscription !](#)

FIGURE 19 – Inscription : Étape 3

### 5.1.3 Accès aux paris

L'accès aux pages de paris est possible en mode déconnecté mais il est impossible de parier. Chaque pari propose au visiteur non connecté de s'inscrire sur le site pour pouvoir parier.

Les Derniers Paris			
Enjeux	Description	Date limite	Parier
Clermont-Toulouse	Final Top 14	30/06/2010 00:00:00	S'inscrire pour parier
Federer-Nadal	match de tennis Federer VS Nadal	30/06/2010 00:00:00	S'inscrire pour parier

FIGURE 20 – Liste des paris en mode déconnecté

## 5.2 Mode connecté

### 5.2.1 Vue d'ensemble

Lorsqu'un utilisateur se connecte, le *header* de la page change et devient spécifique au client. On y retrouve alors sa civilité, son nom, son prénom ainsi que son solde disponible.

Le lien **Profil** permet à l'utilisateur de modifier toutes les informations qu'il a donné lors de son inscription.

FIGURE 21 – Header en mode connecté

### 5.2.2 Profil utilisateur

La page informations personnelles, accessible depuis le profil, permet de voir et de modifier les informations fournies lors de l’inscription.

L’édition est interactive : l’utilisateur n’a qu’à cliquer sur l’un des liens **Éditer** pour éditer la partie correspondante. JavaScript génère alors automatiquement le formulaire d’édition et les modifications seront enregistrées de manière asynchrone via AJAX.

FIGURE 22 – Édition du profil

L’espace de profil permet également de créer son compte virtuel. Il faut en effet que l’utilisateur lie son compte membre **bet-on-tv** à l’un de ses comptes bancaire réel pour pouvoir effectuer un premier virement et ensuite pouvoir jouer.

Si l’utilisateur n’a pas de compte bancaire enregistré, il est invité à en créer un. Un formulaire se génère alors pour lui permettre de saisir ses données bancaires.

FIGURE 23 – Invitation à la création d’un compte

### 5.2.3 Accès aux paris

La page des paris en mode connecté est identique à celle du mode connecté à une exception près : on peut désormais parier !

FIGURE 24 – Saisie des informations bancaires

Les Derniers Paris			
Enjeux	Description	Date limite	Parier
Clermont-Toulouse	Final Top 14	30/06/2010 00:00:00	Parier
Federer-Nadal	match de tennis Federer VS Nadal	30/06/2010 00:00:00	Parier

FIGURE 25 – Page des paris sportif en mode connecté

Il suffit ensuite de rentrer le montant concernant le pari, ainsi que le choix souhaité.

Une fois le pari validé, le compte virtuel du client est débité de la somme concernant son pari. Le client pourra accéder à la liste des paris auxquels il a joué depuis sa zone *profil* via un historique lui indiquant les paris joués/gagnés/perdus et les montant misés à chaque fois.

## 5.3 Côté administrateur

### 5.3.1 Vue d'ensemble

On accède au panneau d'administration en se connectant en tant que `admin:admin`. Une fois connecté en tant qu'administrateur, on peut alors accéder au panneau d'administration en cliquant simplement sur le lien **Administration** disponible dans le *header*.

On arrive alors sur la page suivante :



FIGURE 26 – Index de la page d'administration

On distingue alors trois onglets :

- gestion des utilisateurs
- gestion des paris
- gestion des transactions

Remarque : L'espace d'administration est hautement interactif et exploite intensivement les animations et fonctionnalités **AJAX** de **jQuery**.

### 5.3.2 Gestion des utilisateurs

Cet onglet donne accès :

- à la liste de tous les utilisateurs/membres du site
- à des fonctionnalités de bannissement ou de suppression de compte membre
- à des fonctionnalités d'affichage détaillé des informations personnelles de l'utilisateur ainsi que de ses transactions (flux d'argent)
- à la possibilité de valider ou d'invalider un moyen de paiement soumis par un utilisateur

Gestion des Utilisateurs									
Liste des utilisateurs									
<input type="checkbox"/>	Pseudo	Courriel	Solde	Adresse IP	DDN	Création	Mise à jour	Connexion	
<input type="checkbox"/>	admin	admin@site.fr	1000.0	127.0.0.1	01/05/1980	01/06/2010	13/06/2010	16/06/2010	● ●
<input type="checkbox"/>	Aynwaer	test@me.org	0.0	0.0.0.0	11/02/1987	13/06/2010	13/06/2010	13/06/2010	● ●
<input type="checkbox"/>	aghnar	aghnar@gmail.com	2000.0	192.168.1.1	11/03/1963	13/06/2010	16/06/2010	18/06/2010	● ●
<input type="checkbox"/>	jojo80	jojo@funmail.com	100.0	234.167.9.156	16/08/1985	13/06/2010	13/06/2010	13/06/2010	● ●
<input type="checkbox"/>	vera	vera_11@enseiht.fr	450.0	234.145.9.8	28/12/1972	13/06/2010	16/06/2010	16/06/2010	● ●
<input type="checkbox"/>	balbinho	jkjkl@jkh	0.0	0.0.0.0	01/01/1970	22/06/2010	22/06/2010	22/06/2010	● ●

FIGURE 27 – Liste des utilisateurs

Remarque : Les deux dernières colonnes correspondent, de gauche à droite, à :

- l'état d'activation du compte membre. Vert : activé, orange : en attente de confirmation (email), rouge : banni.
- au statut du moyen de paiement de l'utilisateur. Vert : accepté, orange : en cours de vérification, rouge : refusé.

En sélectionnant un membre ou en cliquant directement sur le pseudo du membre, l'administrateur accède aux données personnelles de l'utilisateur.

Accueil Administration			
Gestion des Utilisateurs			
Gestion des Paris			
Transactions Bancaires			
Détails de l'utilisateur			
Transaction de l'utilisateur			
Détails de l'utilisateur user4			
<b>Informations personnelles</b> Civilité : Monsieur Prénom : Fuser4 Nom : Luser4 Date de naissance : 04/01/1970 Login : user4 Adresse IP : 0.0.0.0	<b>Coordonnées</b> Adresse : Addruser4 Code Postal : 00004 Ville : User4 city Pays : User4 country Téléphone fixe : Téléphone portable : Email : user4@botv	<b>Informations bancaires</b> Balance du compte : 183.0 Type de carte : Electron Date de création : 2010-06-17 15:50:07.0 Numéro de carte : 1457789534777985 Date d'expiration : 2010-06-17 Nom du détenteur de la carte : Lola Cryptogramme : 004 Le moyen de paiement est activé.	<b>Informations sur le compte</b> Email : user4@botv Date de création : 17/06/2010 12:27:40 Date de dernière modification : 17/06/2010 12:27:40 Date de dernière connexion : 17/06/2010 12:27:40 Newsletter du site : Refusée Newsletter des partenaires : Refusée Statut du compte : ● En attente de confirmation

FIGURE 28 – Informations liées à l'utilisateur

Et à ses transactions...

Détails de l'utilisateur		Transaction de l'utilisateur		
Liste des transactions de l'utilisateur user4				
Date de la transaction	Compte Site	Libellé	Débit	Crédit
17/06/2010 15:50:07	Compte Bonus	Bonus De Création De Compte Bancaire	0.00€	100.00€
18/06/2010 12:01:52	Compte Paris	PRÉLEVEMENT PARI France - Mexique   CHOIX : `Mexique`	40.00€	0.00€
18/06/2010 12:30:07	Compte Paris	GAIN PARI `France - Mexique`   CHOIX `Mexique`	0.00€	130.00€
18/06/2010 12:30:07	Compte Paris	PRÉLEVEMENT TAXE POUR LE SITE	4.00€	0.00€
18/06/2010 12:30:07	Compte Paris	PRÉLEVEMENT TAXE POUR L'ÉTAT	3.00€	0.00€

FIGURE 29 – Liste des utilisateurs

### 5.3.3 Gestion des paris

C'est ici qu'on crée, édite et clôture les paris du site. La page principale liste tout simplement les paris existant.

des Utilisateurs

Gestion des Paris

Transactions Bancaires

Liste des paris

<input type="checkbox"/>	Nom	Catégorie	Création	Début	Fin	
<input type="checkbox"/>	France - Uruguay	Football	17/06/2010 15:28:39	01/06/2010 15:28:00	18/06/2010 20:20:00	
<input type="checkbox"/>	France - Mexique	Sport	18/06/2010 00:50:19	14/06/2010 00:50:00	17/06/2010 22:30:00	
<input type="checkbox"/>	France - Afrique du Sud	Sport	18/06/2010 15:37:39	18/06/2010 15:37:00	21/06/2010 15:37:00	

Valider

Annuler

Supprimer

Ajouter

FIGURE 30 – Liste des paris

De la même façon que pour les utilisateurs, sélectionner un pari nous donne accès à une zone d'édition/clôture du pari.

Créer un pari se fait facilement via le champ **Ajouter** sous la liste des paris, qui déploie automatiquement le volet d'édition...

Editer un pari		Choix
Nom :	France - Uruguay <input checked="" type="checkbox"/> Activé	<input type="radio"/> France
Description :	Le 1er match de l'équipe de France pour la coupe du monde.	<input type="radio"/> Uruguay
		<input type="button" value="Clôturer le pari"/>
Catégorie :	Football	
Date de début :	2010-06-01 15:28 UTC	
Date de fin :	2010-06-18 20:20 UTC	

FIGURE 31 – Création/édition/clôture d'un pari

On a également facilement accès à la liste des membres ayant participé à ce pari, aux sommes mises et aux gagnants/perdants du pari.

### 5.3.4 Gestion des transactions

Ce dernier onglet donne un accès global à toutes les transactions du site.

Editer un pari

Afficher les joueurs

Liste des joueurs

Joueur	Choix	Mise	Date de l'opération	Statut
user1	France	10.0€	18/06/2010 11:52:09	
user2	France	30.0€	18/06/2010 11:58:02	
user3	France	20.0€	18/06/2010 11:59:53	
user4	Mexique	40.0€	18/06/2010 12:01:52	Gagné !
user5	France	30.0€	18/06/2010 12:03:39	

FIGURE 32 – Joueurs associés à un pari

des Utilisateurs		Gestion des Paris		Transactions Bancaires	
Liste des transactions					
Compte du Site	Utilisateur	Libellé	Date de l'opération	Débit	Crédit
Compte Bonus	user1	Bonus de création de compte bancaire	17/06/2010 15:37:30	0.00€	100.00€
Compte Bonus	user2	Bonus de création de compte bancaire	17/06/2010 15:46:38	0.00€	100.00€
Compte Bonus	user3	Bonus de création de compte bancaire	17/06/2010 15:48:26	0.00€	100.00€
Compte Bonus	user4	Bonus de création de compte bancaire	17/06/2010 15:50:07	0.00€	100.00€
Compte Bonus	user5	Bonus de création de compte bancaire	17/06/2010 15:51:33	0.00€	100.00€
Compte Bonus	user6	Bonus de création de compte bancaire	17/06/2010 15:52:43	0.00€	100.00€
Compte Paris	user1	PRÉLÈVEMENT PARI France - Mexique   CHOIX : `France`	18/06/2010 11:52:09	10.00€	0.00€
Compte Paris	user2	PRÉLÈVEMENT PARI France - Mexique   CHOIX : `France`	18/06/2010 11:58:02	30.00€	0.00€
Compte Paris	user3	PRÉLÈVEMENT PARI France - Mexique   CHOIX : `France`	18/06/2010 11:59:53	20.00€	0.00€
Compte Paris	user4	PRÉLÈVEMENT PARI France - Mexique   CHOIX : `Mexique`	18/06/2010 12:01:52	40.00€	0.00€
Compte Paris	user5	PRÉLÈVEMENT PARI France - Mexique   CHOIX : `France`	18/06/2010 12:03:39	30.00€	0.00€
Compte Paris	user4	GAIN PARI `France - Mexique`   CHOIX `Mexique`	18/06/2010 12:30:07	0.00€	130.00€
Compte Paris	user4	PRÉLÈVEMENT TAXE POUR LE SITE	18/06/2010 12:30:07	4.00€	0.00€
Compte Paris	user4	PRÉLEVEMENT TAXE POUR L'ÉTAT	18/06/2010 12:30:07	3.00€	0.00€

FIGURE 33 – Gestionnaire de transactions

## 6 Conclusion

### 6.1 Difficultés rencontrées

Nous avons rencontré principalement deux difficultés sur ce projet. L'une liée à **Spring Security** et l'autre liée aux transactions.

La documentation est en effet peu claire au sujet de **Spring Security** lorsqu'il s'agit de créer nous même les sessions depuis du code **Java** et beaucoup de tutoriels à ce sujet se ressemblent et sont rarement complet (présence uniquement du code **XML** et pas du code **Java** par exemple).

Après plusieurs dizaines d'heures de recherche, de tests et de débogage, nous avons cependant trouvé comment exploiter **Spring Security** en **Java** et avons pu continuer le développement dans d'autres directions.

Un autre problème important auquel nous avons fait face est celui des transactions (base de données). En effet, il est possible en **Spring** et si on utilise un serveur d'application (**JBoss**, **GlassFish**... mais pas **Tomcat**) de gérer celles-ci automatiquement via la **Java Transaction API (JTA)** par l'usage d'annotations **@Transactional** placées devant les méthodes où classes.

L'usage de ces annotations et de **JTA** permet alors de gérer automatiquement les début et fin de transaction et notamment (et surtout) les **auto commit** et **auto rollback**.

Cependant, bien qu'ayant passé plusieurs dizaines d'heures sur ce problème, nous n'avons pas réussi à le résoudre, malgré d'intenses recherches et de très nombreux tests, et sommes revenus à un système manuel de gestion des transactions.

## 6.2 Limites et améliorations

Le site web en l'état n'est pas tout à fait terminé. Le design de la partie utilisateur est inachevé dans certaines zones et certaines fonctionnalités, telles les tris ou filtrage des paris, sont manquantes.

De même, le panneau d'administration, bien que très fortement avancé, n'offre pas encore toutes les fonctionnalités escomptées (bannissements, suppressions...)

Cependant le site est suffisamment bien avancé pour être exploitable de bout en bout sans avoir à effectuer de tâches "à la main" (*ie.* accès manuel à la base de données) et permet de créer des utilisateurs, des comptes bancaires, des paris, de jouer sur ces paris et de gagner éventuellement. Le tout fourni dans un environnement sécurisé et contrôlé (pas de pari si pas d'argent).

Les améliorations à apporter sont innombrables si on compare ce site à ceux de professionnels actuellement en ligne : préférences et habitudes utilisateurs, super-cookies, gestion dynamique de la page d'accueil, site multilingue, etc.

## 6.3 Apports personnels

Ce projet ayant été réalisé entièrement avec des technologies non enseignées à l'ENSEEIH (Spring MVC, Spring Security, Hibernate, HTML 5, CSS 3, jQuery, MySQL, transactions) et dans leur dernière version, il nous a apporté beaucoup de connaissances sur ces différents frameworks et nous a permis de mieux appréhender comment concevoir, développer et déployer un site web basé sur **Java Enterprise Edition**.

## A Sources et liens

### A.1 Général

- Support d'HTML 5 par les différents navigateurs  
<http://html5readiness.com/>
- Étude comparant JBoss, Geronimo et Tomcat  
<http://www.javaworld.com/javaworld/jw-12-2007/jw-12-appservers.html>
- Comparaison entre différents serveurs Java EE  
<http://www.slideshare.net/OpenLogic/comparison-of-open-source-app-servers-final-presentation>

### A.2 Maven

- Tutoriel : Utiliser Maven 2  
<http://matthieu-lux.developpez.com/tutoriels/java/maven/>
- tutoriel : Créer un site avec Maven 2  
<http://baptiste-wicht.ftp-developpez.com/tutoriel/java/maven/site/maven-site.pdf>
- Déploiement pour Tomcat 6, par Maven  
<http://www.waltercedric.com/java-j2ee-mainmenu-53/361-maven-build-system/1555-deploy-to-tomcat-6-using-maven.html>

### A.3 Spring

Spring & NetBeans :

- Introduction to Spring Web MVC  
<http://netbeans.org/kb/docs/web/quickstart-webapps-spring.html>

Documentation officielle et générale :

- Developing a Spring Framework MVC application step-by-step (version 2.5)  
<http://static.springsource.org/docs/Spring-MVC-step-by-step/>
- Spring Framework, Reference Documentation, version 3.0  
<http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/>
- Form Processing with Spring MVC  
<http://levelup.lishman.com/spring/form-processing/>

Spring Security :

- Implement a Fixed Login Form in Spring  
<http://www.developer.com/java/web/article.php/3846816/Implement-a-Fixed-Login-Form-in-Spring.htm>
- [http://www.jtips.info/index.php?title=Spring\\_Security](http://www.jtips.info/index.php?title=Spring_Security)

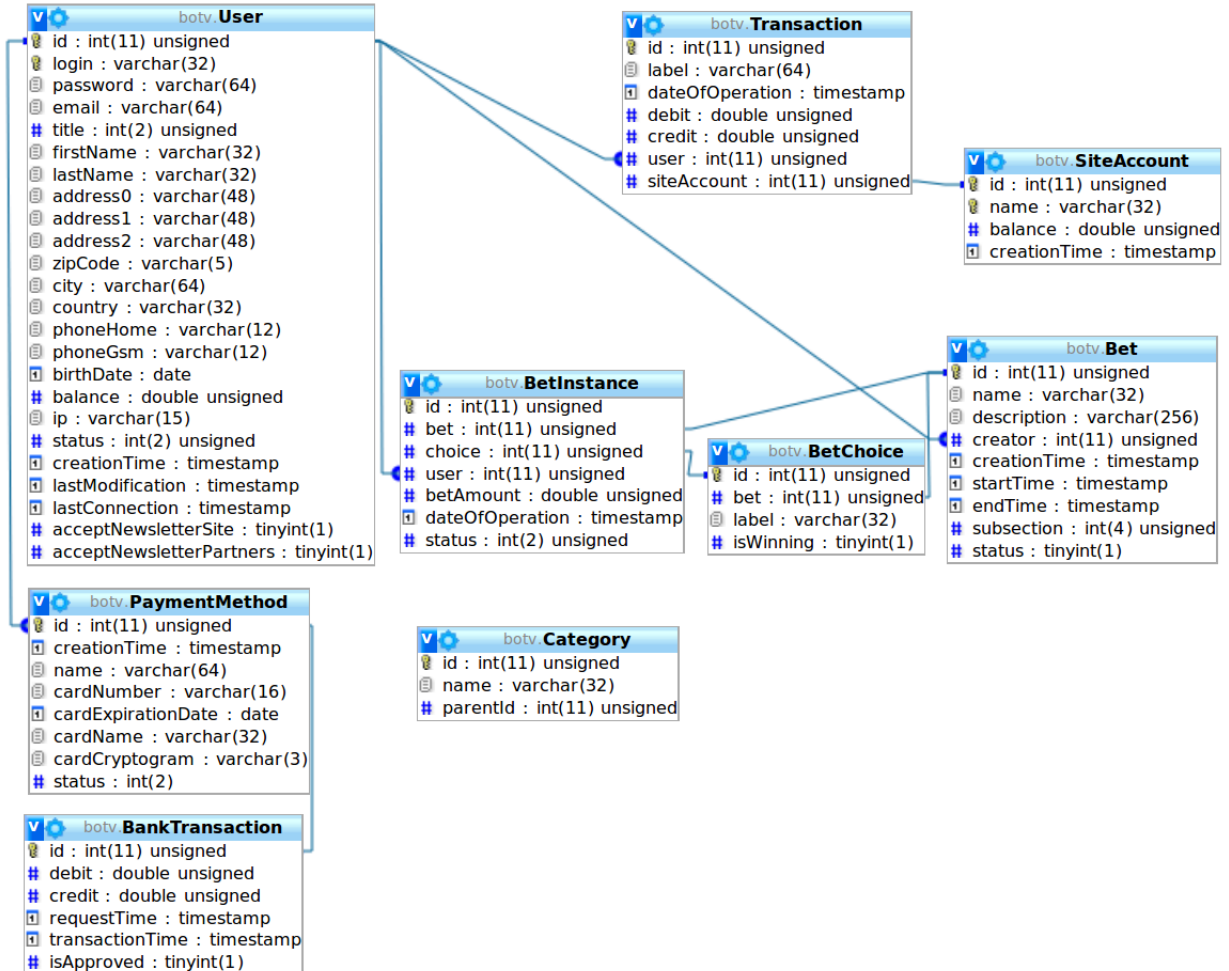
### A.4 Hibernate

- Using Hibernate in a Web Application  
<http://netbeans.org/kb/docs/web/hibernate-webapp.html>
- Tutoriel : Spring 3.0 and Hibernate  
<http://singihpraditya.wordpress.com/2010/02/13/spring-3-0-and-hibernate-tutorial-part-1/>
- Premier projet avec Tapestry5, Spring et Hibernate  
<http://baptiste-meurant.developpez.com/tutoriaux/tapestry5-spring-hibernate/>

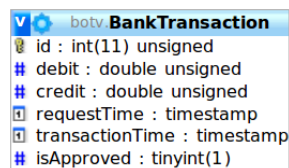
## B Annexe

### B.1 Base de données

#### B.1.1 Vue d'ensemble



#### B.1.2 Table BankTransaction



```

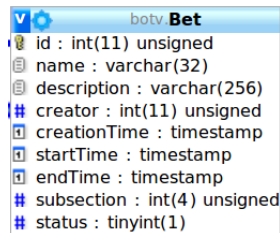
1 CREATE TABLE IF NOT EXISTS 'BankTransaction' (
2   'id' int(11) unsigned NOT NULL AUTOINCREMENT,
3   'debit' double unsigned NOT NULL DEFAULT '0',
4   'credit' double unsigned NOT NULL DEFAULT '0',
5   'requestTime' timestamp NOT NULL DEFAULT CURRENTTIMESTAMP,
6   'transactionTime' timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  
```

```

7   'isApproved' tinyint(1) NOT NULL,
8   PRIMARY KEY ('id')
9 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1 ;

```

### B.1.3 Table Bet



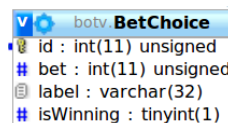
Column	Type	Constraints
id	int(11) unsigned	Primary Key
name	varchar(32)	
description	varchar(256)	
creator	int(11) unsigned	
creationTime	timestamp	
startTime	timestamp	
endTime	timestamp	
subsection	int(4) unsigned	
status	tinyint(1)	

```

1 CREATE TABLE IF NOT EXISTS 'Bet' (
2   'id' int(11) unsigned NOT NULL AUTO_INCREMENT,
3   'name' varchar(32) COLLATE utf8_bin NOT NULL,
4   'description' varchar(256) COLLATE utf8_bin NOT NULL,
5   'creator' int(11) unsigned NOT NULL,
6   'creationTime' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
7   'startTime' timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
8   'endTime' timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
9   'subsection' int(4) unsigned NOT NULL,
10  'status' tinyint(1) NOT NULL DEFAULT '0',
11  PRIMARY KEY ('id'),
12  KEY 'creator' ('creator')
13 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1 ;

```

### B.1.4 Table BetChoice



Column	Type	Constraints
id	int(11) unsigned	Primary Key
bet	int(11) unsigned	
label	varchar(32)	
isWinning	tinyint(1)	

```

1 CREATE TABLE IF NOT EXISTS 'BetChoice' (
2   'id' int(11) unsigned NOT NULL AUTO_INCREMENT,
3   'bet' int(11) unsigned NOT NULL,
4   'label' varchar(32) COLLATE utf8_bin NOT NULL,
5   'isWinning' tinyint(1) NOT NULL DEFAULT '0',
6   PRIMARY KEY ('id'),
7   KEY 'bet' ('bet')
8 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1 ;

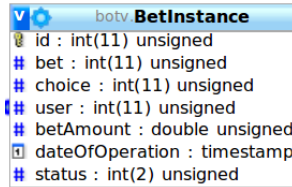
```

### B.1.5 Table BetInstance

```

1 CREATE TABLE IF NOT EXISTS 'BetInstance' (
2   'id' int(11) unsigned NOT NULL AUTO_INCREMENT,
3   'bet' int(11) unsigned NOT NULL,

```



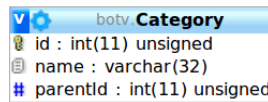
botv.BetInstance	
id	int(11) unsigned
bet	int(11) unsigned
choice	int(11) unsigned
user	int(11) unsigned
betAmount	double unsigned
dateOfOperation	timestamp
status	int(2) unsigned

```

4  'choice' int(11) unsigned NOT NULL,
5  'user' int(11) unsigned NOT NULL,
6  'betAmount' double unsigned NOT NULL,
7  'dateOfOperation' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
8  'status' int(2) unsigned NOT NULL DEFAULT '0',
9  PRIMARY KEY ('id'),
10 KEY 'bet' ('bet'),
11 KEY 'user' ('user'),
12 KEY 'choice' ('choice')
13 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1 ;

```

### B.1.6 Table Category



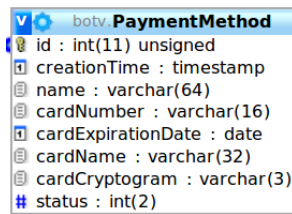
botv.Category	
id	int(11) unsigned
name	varchar(32)
parentId	int(11) unsigned

```

1 CREATE TABLE IF NOT EXISTS 'Category' (
2   'id' int(11) unsigned NOT NULL AUTO_INCREMENT,
3   'name' varchar(32) COLLATE utf8_bin NOT NULL,
4   'parentId' int(11) unsigned NOT NULL,
5   PRIMARY KEY ('id')
6 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1 ;

```

### B.1.7 Table PaymentMethod



botv.PaymentMethod	
id	int(11) unsigned
creationTime	timestamp
name	varchar(64)
cardNumber	varchar(16)
cardExpirationDate	date
cardName	varchar(32)
cardCryptogram	varchar(3)
status	int(2)

```

1 CREATE TABLE IF NOT EXISTS 'PaymentMethod' (
2   'id' int(11) unsigned NOT NULL,
3   'creationTime' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
4   'name' varchar(64) COLLATE utf8_bin NOT NULL,
5   'cardNumber' varchar(16) COLLATE utf8_bin NOT NULL,
6   'cardExpirationDate' date NOT NULL,
7   'cardName' varchar(32) COLLATE utf8_bin NOT NULL,
8   'cardCryptogram' varchar(3) COLLATE utf8_bin NOT NULL,

```

```

9      'status' int(2) NOT NULL
10     PRIMARY KEY ('id')
11 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

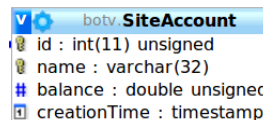
```

### B.1.8 Table SiteAccount

```

1 ALTER TABLE 'BankTransaction'
2   ADD CONSTRAINT 'BankTransaction_ibfk_1' FOREIGN KEY ('id')
3     REFERENCES 'PaymentMethod' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;
4
5 ALTER TABLE 'Bet'
6   ADD CONSTRAINT 'Bet_ibfk_1' FOREIGN KEY ('creator')
7     REFERENCES 'User' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;
8
9 ALTER TABLE 'BetChoice'
10  ADD CONSTRAINT 'BetChoice_ibfk_1' FOREIGN KEY ('bet')
11    REFERENCES 'Bet' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;
12
13 ALTER TABLE 'BetInstance'
14  ADD CONSTRAINT 'BetInstance_ibfk_1' FOREIGN KEY ('bet')
15    REFERENCES 'Bet' ('id') ON DELETE NO ACTION ON UPDATE CASCADE,
16  ADD CONSTRAINT 'BetInstance_ibfk_2' FOREIGN KEY ('choice')
17    REFERENCES 'BetChoice' ('id') ON DELETE NO ACTION ON UPDATE CASCADE,
18  ADD CONSTRAINT 'BetInstance_ibfk_3' FOREIGN KEY ('user')
19    REFERENCES 'User' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;
20
21 ALTER TABLE 'PaymentMethod'
22  ADD CONSTRAINT 'PaymentMethod_ibfk_1' FOREIGN KEY ('id')
23    REFERENCES 'User' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;
24
25 ALTER TABLE 'Transaction'
26  ADD CONSTRAINT 'Transaction_ibfk_1' FOREIGN KEY ('user')
27    REFERENCES 'User' ('id') ON DELETE NO ACTION ON UPDATE CASCADE,
28  ADD CONSTRAINT 'Transaction_ibfk_2' FOREIGN KEY ('siteAccount')
29    REFERENCES 'SiteAccount' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;

```



```

1 CREATE TABLE IF NOT EXISTS 'SiteAccount' (
2   'id' int(11) unsigned NOT NULL AUTO.INCREMENT,
3   'name' varchar(32) COLLATE utf8_bin NOT NULL,
4   'balance' double unsigned NOT NULL DEFAULT '0',
5   'creationTime' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
6   PRIMARY KEY ('id'),
7   UNIQUE KEY 'name' ('name')
8 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO.INCREMENT=1 ;
9
10 INSERT INTO 'SiteAccount' ('id', 'name', 'balance', 'creationTime') VALUES
11 (1, 'Compte Paris', 0, '2010-06-06 04:41:49'),
12 (2, 'Compte Taxe Site', 0, '2010-06-10 18:58:49'),

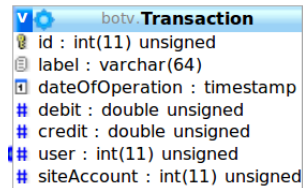
```

```

13 (3, 'Compte Taxe Attat', 0, '2010-06-10 18:58:49'),
14 (4, 'Compte Bonus', 0, '2010-06-10 15:34:01');

```

### B.1.9 Table Transaction



Field	Type	Unsigned	Zero	NotNull	Default	Extra
id	int(11)	Yes	Yes	Yes		AUTO INCREMENT
label	varchar(64)	No	Yes	Yes		
dateOfOperation	timestamp	No	Yes	Yes		DEFAULT CURRENT_TIMESTAMP
debit	double	Yes	Yes	Yes	0	
credit	double	Yes	Yes	Yes	0	
user	int(11)	Yes	Yes	Yes		
siteAccount	int(11)	Yes	Yes	Yes	1	

```

1 CREATE TABLE IF NOT EXISTS 'Transaction' (
2   'id' int(11) unsigned NOT NULL AUTO INCREMENT,
3   'label' varchar(64) COLLATE utf8_bin NOT NULL,
4   'dateOfOperation' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
5   'debit' double unsigned NOT NULL DEFAULT '0',
6   'credit' double unsigned NOT NULL DEFAULT '0',
7   'user' int(11) unsigned NOT NULL,
8   'siteAccount' int(11) unsigned NOT NULL DEFAULT '1',
9   PRIMARY KEY ('id'),
10  KEY 'user' ('user'),
11  KEY 'siteAccount' ('siteAccount')
12 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO INCREMENT=1 ;

```

### B.1.10 Table User



Field	Type	Unsigned	Zero	NotNull	Default	Extra
id	int(11)	Yes	Yes	Yes		AUTO INCREMENT
login	varchar(32)	No	Yes	Yes		
password	varchar(64)	No	Yes	Yes		
email	varchar(64)	No	Yes	Yes		
title	int(2)	Yes	Yes	Yes		
firstName	varchar(32)	No	Yes	Yes		
lastName	varchar(32)	No	Yes	Yes		
address0	varchar(48)	No	Yes	Yes		
address1	varchar(48)	No	Yes	Yes		
address2	varchar(48)	No	Yes	Yes		
zipCode	varchar(5)	No	Yes	Yes		
city	varchar(64)	No	Yes	Yes		
country	varchar(32)	No	Yes	Yes		
phoneHome	varchar(12)	No	Yes	Yes		
phoneGsm	varchar(12)	No	Yes	Yes		
birthDate	date	No	Yes	Yes		
balance	double	Yes	Yes	Yes		
ip	varchar(15)	No	Yes	Yes		
status	int(2)	Yes	Yes	Yes		
creationTime	timestamp	No	Yes	Yes		
lastModification	timestamp	No	Yes	Yes		
lastConnection	timestamp	No	Yes	Yes		
acceptNewsletterSite	tinyint(1)	No	Yes	Yes		
acceptNewsletterPartners	tinyint(1)	No	Yes	Yes		

```

1 CREATE TABLE IF NOT EXISTS 'User' (
2   'id' int(11) unsigned NOT NULL AUTO INCREMENT,
3   'login' varchar(32) COLLATE utf8_bin NOT NULL DEFAULT '',
4   'password' varchar(64) COLLATE utf8_bin NOT NULL DEFAULT '',
5   'email' varchar(64) COLLATE utf8_bin NOT NULL DEFAULT '',

```

```

6  'title' int(2) unsigned NOT NULL DEFAULT '0' COMMENT 'M., Mme, Mlle',
7  'firstName' varchar(32) COLLATE utf8_bin NOT NULL DEFAULT '',
8  'lastName' varchar(32) COLLATE utf8_bin NOT NULL DEFAULT '',
9  'address0' varchar(48) COLLATE utf8_bin NOT NULL DEFAULT '',
10 'address1' varchar(48) COLLATE utf8_bin NOT NULL DEFAULT '',
11 'address2' varchar(48) COLLATE utf8_bin NOT NULL DEFAULT '',
12 'zipCode' varchar(5) COLLATE utf8_bin NOT NULL DEFAULT '',
13 'city' varchar(64) COLLATE utf8_bin NOT NULL DEFAULT '',
14 'country' varchar(32) COLLATE utf8_bin NOT NULL DEFAULT '',
15 'phoneHome' varchar(12) COLLATE utf8_bin NOT NULL DEFAULT '',
16 'phoneGsm' varchar(12) COLLATE utf8_bin NOT NULL DEFAULT '',
17 'birthDate' date NOT NULL DEFAULT '2010-01-01',
18 'balance' double unsigned NOT NULL DEFAULT '0',
19 'ip' varchar(15) COLLATE utf8_bin NOT NULL DEFAULT '',
20 'status' int(2) unsigned NOT NULL DEFAULT '1',
21 'creationTime' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
22 'lastModification' timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
23 'lastConnection' timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
24 'acceptNewsletterSite' tinyint(1) NOT NULL DEFAULT '0',
25 'acceptNewsletterPartners' tinyint(1) NOT NULL DEFAULT '0',
26 PRIMARY KEY ('id'),
27 UNIQUE KEY 'login' ('login')
28 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1 ;
29
30 INSERT INTO 'User' ('id', 'login', 'password', 'email', 'title',
31 'firstName', 'lastName', 'address0', 'address1', 'address2', 'zipCode',
32 'city', 'country', 'phoneHome', 'phoneGsm', 'birthDate', 'balance', 'ip',
33 'status', 'creationTime', 'lastModification', 'lastConnection',
34 'acceptNewsletterSite', 'acceptNewsletterPartners') VALUES
35 (1, 'admin', MD5('admin'), 'admin@site.fr', 0, 'Admin', 'Strateur',
36 'Admin & Co Ltd.', '1, Avenue de la Gestion', '78e étage, bureau n°1',
37 '75000', 'Paris', 'France', '0110203040', '', '1980-05-01', 0, '127.0.0.1',
38 2, '2010-06-01 00:00:00', '2010-06-13 00:00:00', '2010-06-17 11:04:34', 0, 0);

```

### B.1.11 Clés étrangères

```

1 ALTER TABLE 'BankTransaction'
2   ADD CONSTRAINT 'BankTransaction_ibfk_1' FOREIGN KEY ('id')
3     REFERENCES 'PaymentMethod' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;
4
5 ALTER TABLE 'Bet'
6   ADD CONSTRAINT 'Bet_ibfk_1' FOREIGN KEY ('creator')
7     REFERENCES 'User' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;
8
9 ALTER TABLE 'BetChoice'
10  ADD CONSTRAINT 'BetChoice_ibfk_1' FOREIGN KEY ('bet')
11    REFERENCES 'Bet' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;
12
13 ALTER TABLE 'BetInstance'
14  ADD CONSTRAINT 'BetInstance_ibfk_1' FOREIGN KEY ('bet')
15    REFERENCES 'Bet' ('id') ON DELETE NO ACTION ON UPDATE CASCADE,
16  ADD CONSTRAINT 'BetInstance_ibfk_2' FOREIGN KEY ('choice')
17    REFERENCES 'BetChoice' ('id') ON DELETE NO ACTION ON UPDATE CASCADE,
18  ADD CONSTRAINT 'BetInstance_ibfk_3' FOREIGN KEY ('user')
19    REFERENCES 'User' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;

```

```

20
21 ALTER TABLE 'PaymentMethod'
22     ADD CONSTRAINT 'PaymentMethod_ibfk_1' FOREIGN KEY ('id')
23     REFERENCES 'User' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;
24
25 ALTER TABLE 'Transaction'
26     ADD CONSTRAINT 'Transaction_ibfk_1' FOREIGN KEY ('user')
27     REFERENCES 'User' ('id') ON DELETE NO ACTION ON UPDATE CASCADE,
28     ADD CONSTRAINT 'Transaction_ibfk_2' FOREIGN KEY ('siteAccount')
29     REFERENCES 'SiteAccount' ('id') ON DELETE NO ACTION ON UPDATE CASCADE;

```

### B.1.12 Contenu minimal

```

1 INSERT INTO 'User' ('id', 'login', 'password', 'email', 'title',
2 'firstName', 'lastName', 'address0', 'address1', 'address2', 'zipCode',
3 'city', 'country', 'phoneHome', 'phoneGsm', 'birthDate', 'balance', 'ip',
4 'status', 'creationTime', 'lastModification', 'lastConnection',
5 'acceptNewsletterSite', 'acceptNewsletterPartners') VALUES
6 (1, 'admin', MD5('admin'), 'admin@site.fr', 0, 'Admin', 'Strateur',
7 'Admin & Co Ltd.', '1, Avenue de la Gestion', '78e étage, bureau n°1',
8 '75000', 'Paris', 'France', '0110203040', '', '1980-05-01', 0, '127.0.0.1',
9 2, '2010-06-01 00:00:00', '2010-06-13 00:00:00', '2010-06-17 11:04:34', 0, 0);
10
11 INSERT INTO 'SiteAccount' ('id', 'name', 'balance', 'creationTime') VALUES
12 (1, 'Compte Paris', 0, '2010-06-06 04:41:49'),
13 (2, 'Compte Taxe Site', 0, '2010-06-10 18:58:49'),
14 (3, 'Compte Taxe État', 0, '2010-06-10 18:58:49'),
15 (4, 'Compte Bonus', 0, '2010-06-10 15:34:01');

```

## B.2 Couche modèle - Entités

### B.2.1 Bet

```

1 *package com.boc.botv.model;
2
3 import java.text.SimpleDateFormat;
4 import java.util.Date;
5 import java.util.HashSet;
6 import java.util.Set;
7 import javax.persistence.CascadeType;
8 import javax.persistence.Column;
9 import javax.persistence.Entity;
10 import javax.persistence.FetchType;
11 import javax.persistence.GeneratedValue;
12 import static javax.persistence.GenerationType.IDENTITY;
13 import javax.persistence.Id;
14 import javax.persistence.JoinColumn;
15 import javax.persistence.ManyToOne;
16 import javax.persistence.OneToMany;
17 import javax.persistence.Table;
18 import javax.persistence.Temporal;
19 import javax.persistence.TemporalType;
20 import javax.persistence.Transient;
21

```

```

22 /**
23  * Représente en pari
24  * @author Fabien Renaud
25  */
26 @Entity
27 @Table(name = "Bet", catalog = "botv")
28 public class Bet implements java.io.Serializable {
29
30     private static final SimpleDateFormat dateFormatter =
31         new SimpleDateFormat("dd/MM/yyyy");
32     private static final SimpleDateFormat dateLongFormatter =
33         new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
34
35     private Integer id;
36     private User user;
37     private String name;
38     private String description;
39     private Date creationTime;
40     private Date startTime;
41     private Date endTime;
42     private int subsection;
43     private boolean status;
44     private Set<BetChoice> betChoices = new HashSet<BetChoice>(0);
45     private Set<BetInstance> betInstances = new HashSet<BetInstance>(0);
46
47     private String sectionName;
48     /**
49      * Crée un pari par défaut
50      */
51     public Bet() {
52     }
53
54     /**
55      * Crée un pari
56      * @param user Utilisateur ayant créé le pari
57      * @param name Libellé du pari
58      * @param description Description du pari
59      * @param creationTime Date de création du pari
60      * @param startTime Date de début du par
61      * @param endTime Date de fin du pari
62      * @param subsection Sous catégorie à laquelle appartient le pari
63      * @param status Actif (true) ou inactif
64      */
65     public Bet(User user, String name, String description, Date creationTime,
66         Date startTime, Date endTime, int subsection, boolean status) {
67         this.user = user;
68         this.name = name;
69         this.description = description;
70         this.creationTime = creationTime;
71         this.startTime = startTime;
72         this.endTime = endTime;
73         this.subsection = subsection;
74         this.status = status;
75     }
76     /**
77      * Crée un pari

```

```

78  * @param user Utilisateur ayant créé le pari
79  * @param name Libellé du pari
80  * @param description Description du pari
81  * @param creationTime Date de création du pari
82  * @param startTime Date de début du par
83  * @param endTime Date de fin du pari
84  * @param subsection Sous catégorie à laquelle appartient le pari
85  * @param status Actif (true) ou inactif
86  * @param betChoices Les choix associés au pari
87  * @param betInstances
88  */
89  public Bet(User user, String name, String description, Date creationTime,
90      Date startTime, Date endTime, int subsection, boolean status,
91      Set<BetChoice> betChoices, Set<BetInstance> betInstances) {
92      this.user = user;
93      this.name = name;
94      this.description = description;
95      this.creationTime = creationTime;
96      this.startTime = startTime;
97      this.endTime = endTime;
98      this.subsection = subsection;
99      this.status = status;
100     this.betChoices = betChoices;
101     this.betInstances = betInstances;
102 }
103
104 /**
105  * Obtient l'id du pari
106  * @return L'id du pari
107  */
108 @Id
109 @GeneratedValue(strategy = IDENTITY)
110 @Column(name = "id", unique = true, nullable = false)
111 public Integer getId() {
112     return this.id;
113 }
114
115 /**
116  * Modifie l'id du pari
117  * @param id Nouvel id du pari
118  */
119 public void setId(Integer id) {
120     this.id = id;
121 }
122
123 /**
124  * Obtient l'utilisateur ayant créé le pari
125  * @return L'utilisateur ayant créé le pari
126  */
127 @ManyToOne(fetch = FetchType.EAGER)
128 @JoinColumn(name = "creator", nullable = false)
129 public User getUser() {
130     return this.user;
131 }
132
133 /**

```

```

134     * Modifie l'utilisateur ayant créé le pari
135     * @param user Nouvel utilisateur créateur du pari
136     */
137     public void setUser(User user) {
138         this.user = user;
139     }
140
141     /**
142     * Obtient le libellé du pari
143     * @return Libellé du pari
144     */
145     @Column(name = "name", nullable = false, length = 32)
146     public String getName() {
147         return this.name;
148     }
149
150     /**
151     * Modifie le libellé du pari
152     * @param name Nouveau libellé du pari
153     */
154     public void setName(String name) {
155         this.name = name;
156     }
157
158     /**
159     * Obtient la description du pari
160     * @return Description du pari
161     */
162     @Column(name = "description", nullable = false, length = 256)
163     public String getDescription() {
164         return this.description;
165     }
166
167     /**
168     * Modifie la description du pari
169     * @param description Nouvelle description du pari
170     */
171     public void setDescription(String description) {
172         this.description = description;
173     }
174
175     /**
176     * Obtient la date de création du pari parsée en format long.
177     * @return Date de création du pari
178     */
179     @Transient
180     public String getCreationTimeFormatted() {
181         return dateLongFormatter.format(creationTime);
182     }
183
184     /**
185     * Obtient la date de création du pari
186     * @return Date de création du pari
187     */
188     @Temporal(TemporalType.TIMESTAMP)
189     @Column(name = "creationTime", nullable = false, length = 19)

```

```

190 public Date getCreationTime() {
191     return this.creationTime;
192 }
193
194 /**
195  * Modifie la date de création du pari
196  * @param creationTime Nouvelle date de création du pari
197  */
198 public void setCreationTime(Date creationTime) {
199     this.creationTime = creationTime;
200 }
201
202 /**
203  * Obtient la date de début du pari parsée en format long.
204  * @return Date de début du pari
205  */
206 @Transient
207 public String getStartTimeFormatted() {
208     return dateLongFormatter.format(startTime);
209 }
210
211 /**
212  * Obtient la date de début du pari
213  * @return Date de début du pari
214  */
215 @Temporal(TemporalType.TIMESTAMP)
216 @Column(name = "startTime", nullable = false, length = 19)
217 public Date getStartTime() {
218     return this.startTime;
219 }
220
221 /**
222  * Modifie la date de début du pari
223  * @param startTime Date de début du pari
224  */
225 public void setStartTime(Date startTime) {
226     this.startTime = startTime;
227 }
228
229 /**
230  * Obtient la date de fin du pari parsée en format long.
231  * @return Date de fin du pari
232  */
233 @Transient
234 public String getEndTimeFormatted() {
235     return dateLongFormatter.format(endTime);
236 }
237
238 /**
239  * Obtient la date de fin du pari
240  * @return Date de fin du pari
241  */
242 @Temporal(TemporalType.TIMESTAMP)
243 @Column(name = "endTime", nullable = false, length = 19)
244 public Date getEndTime() {
245     return this.endTime;

```

```

246 }
247
248 /**
249  * Modifie la date de fin du pari
250  * @param endTime Nouvelle date de fin du pari
251  */
252 public void setEndTime(Date endTime) {
253     this.endTime = endTime;
254 }
255
256 /**
257  * Obtient la sous catégorie à laquelle appartient ce pari
258  * @return Sous catégorie à laquelle appartient ce pari
259  */
260 @Column(name = "subsection", nullable = false)
261 public int getSubsection() {
262     return this.subsection;
263 }
264
265 /**
266  * Modifie la sous catégorie à laquelle appartient ce pari
267  * @param subsection Nouvelle sous catégorie à laquelle appartient ce pari
268  */
269 public void setSubsection(int subsection) {
270     this.subsection = subsection;
271 }
272
273 /**
274  * Obtient le statut du pari
275  * @return true si le pari est activé
276  */
277 @Column(name = "status", nullable = false)
278 public boolean isStatus() {
279     return this.status;
280 }
281
282 /**
283  * Modifie le statut du pari
284  * @param status true si le pari est activé
285  */
286 public void setStatus(boolean status) {
287     this.status = status;
288 }
289
290 /**
291  * Obtient les choix possibles pour ce pari
292  * @return Choix du pari
293  */
294 @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy = "bet")
295 public Set<BetChoice> getBetChoices() {
296     return this.betChoices;
297 }
298
299 /**
300  * Modifie les choix possibles pour ce pari
301  * @param betChoices Nouveau choix du pari

```

```

302     */
303     public void setBetChoices(Set<BetChoice> betChoices) {
304         this.betChoices = betChoices;
305     }
306
307     /**
308      *
309      * @return
310      */
311     @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
312         mappedBy = "bet")
313     public Set<BetInstance> getBetInstances() {
314         return this.betInstances;
315     }
316
317     /**
318      *
319      * @param betInstances
320      */
321     public void setBetInstances(Set<BetInstance> betInstances) {
322         this.betInstances = betInstances;
323     }
324
325     /**
326      * Récupère le nom correspondant à la catégorie
327      * @return le nom
328      */
329     @Transient
330     public String getSectionName() {
331         return sectionName;
332     }
333
334     /**
335      * Initialise le nom de la catégorie avec la valeur passée en paramètre
336      * @param sectionName
337      */
338     public void setSectionName(String sectionName) {
339         this.sectionName = sectionName;
340     }
341
342
343 }

```

## B.2.2 BetChoice

```

1  *package com.boc.bottv.model;
2
3  import java.util.HashSet;
4  import java.util.Set;
5  import javax.persistence.CascadeType;
6  import javax.persistence.Column;
7  import javax.persistence.Entity;
8  import javax.persistence.FetchType;
9  import javax.persistence.GeneratedValue;
10 import static javax.persistence.GenerationType.IDENTITY;

```

```

11 import javax.persistence.Id;
12 import javax.persistence.JoinColumn;
13 import javax.persistence.ManyToOne;
14 import javax.persistence.OneToMany;
15 import javax.persistence.Table;
16
17 /**
18  * Représente un choix pour un pari
19  * @author Fabien Renaud
20  */
21 @Entity
22 @Table(name = "BetChoice", catalog = "botv")
23 public class BetChoice implements java.io.Serializable {
24
25     private Integer id;
26     private Bet bet;
27     private String label;
28     private boolean isWinning;
29     private Set<BetInstance> betInstances = new HashSet<BetInstance>(0);
30
31     /**
32      * Crée un choix de pari par défaut
33      */
34     public BetChoice() {
35     }
36
37     /**
38      * Crée un choix de pari
39      * @param bet Pari concerné
40      * @param label Nom du choix
41      * @param isWinning Jeu gagnant (true) ou non
42      */
43     public BetChoice(Bet bet, String label, boolean isWinning) {
44         this.bet = bet;
45         this.label = label;
46         this.isWinning = isWinning;
47     }
48
49     /**
50      * Crée un choix de pari
51      * @param bet Pari concerné
52      * @param label Nom du choix
53      * @param isWinning Jeu gagnant (true) ou non
54      * @param betInstances Jeux effectués sur pari
55      */
56     public BetChoice(Bet bet, String label, boolean isWinning,
57         Set<BetInstance> betInstances) {
58         this.bet = bet;
59         this.label = label;
60         this.isWinning = isWinning;
61         this.betInstances = betInstances;
62     }
63
64     /**
65      * Obtient l'id du choix
66      * @return L'id du choix

```

```

67     */
68     @Id
69     @GeneratedValue(strategy = IDENTITY)
70     @Column(name = "id", unique = true, nullable = false)
71     public Integer getId() {
72         return this.id;
73     }
74
75     /**
76      * Modifie l'id du choix
77      * @param id Nouvel id du choix
78      */
79     public void setId(Integer id) {
80         this.id = id;
81     }
82
83     /**
84      * Obtient le pari lié à ce choix
85      * @return Le pari lié à ce choix
86      */
87     @ManyToOne(fetch = FetchType.EAGER)
88     @JoinColumn(name = "bet", nullable = false)
89     public Bet getBet() {
90         return this.bet;
91     }
92
93     /**
94      * Modifie le pari lié à ce choix
95      * @param bet Le nouveau pari lié à ce choix
96      */
97     public void setBet(Bet bet) {
98         this.bet = bet;
99     }
100
101     /**
102      * Obtient le libellé du choix
103      * @return Le libellé du choix
104      */
105     @Column(name = "label", nullable = false, length = 32)
106     public String getLabel() {
107         return this.label;
108     }
109
110     /**
111      * Modifie le libellé du choix
112      * @param label Le nouvel libellé du choix
113      */
114     public void setLabel(String label) {
115         this.label = label;
116     }
117
118     /**
119      * Obtient s'il s'agit d'un jeu gagnant ou non
120      * @return true si le jeu est gagnant
121      */
122     @Column(name = "isWinning", nullable = false)

```

```

123 public boolean isIsWinning() {
124     return this.isWinning;
125 }
126
127 /**
128  * Modifie le fait que le jeu soit gagnant ou non
129  * @param isWinning true pour que le jeu devienne gagnant
130  */
131 public void setIsWinning(boolean isWinning) {
132     this.isWinning = isWinning;
133 }
134
135 /**
136  * Obtient les jeux pris sur ce choix de pari
137  * @return Jeux pris sur ce choix de pari
138  */
139 @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
140     mappedBy = "betChoice")
141 public Set<BetInstance> getBetInstances() {
142     return this.betInstances;
143 }
144
145 /**
146  * Obtient les jeux ayant (eu) lieu sur ce choix de pari
147  * @param betInstances Nouveaux jeux pris sur ce choix de pari
148  */
149 public void setBetInstances(Set<BetInstance> betInstances) {
150     this.betInstances = betInstances;
151 }
152
153 }

```

### B.2.3 BetInstance

```

1  *package com.boc.botv.model;
2
3  import com.boc.botv.common.BetInstanceStatus;
4  import java.text.SimpleDateFormat;
5  import java.util.Date;
6  import javax.persistence.Column;
7  import javax.persistence.Entity;
8  import javax.persistence.FetchType;
9  import javax.persistence.GeneratedValue;
10 import static javax.persistence.GenerationType.IDENTITY;
11 import javax.persistence.Id;
12 import javax.persistence.JoinColumn;
13 import javax.persistence.ManyToOne;
14 import javax.persistence.Table;
15 import javax.persistence.Temporal;
16 import javax.persistence.TemporalType;
17 import javax.persistence.Transient;
18
19 /**
20  * Représente un jeu sur un pari selon un choix défini par un utilisateur pour un montant
21  * @author Fabien Renaud

```

```

22  */
23  @Entity
24  @Table(name = "BetInstance", catalog = "botv")
25  public class BetInstance implements java.io.Serializable {
26      private static final SimpleDateFormat dateLongFormatter =
27          new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
28
29      private Integer id;
30      private BetChoice betChoice;
31      private User user;
32      private Bet bet;
33      private double betAmount;
34      private Date dateOfOperation;
35      private int status;
36      //Transient
37      private double amount;
38
39      /**
40       * Crée un jeu par défaut sur un pari
41       */
42      public BetInstance() {
43      }
44
45      /**
46       * Crée un jeu sur un pari
47       * @param betChoice Choix de jeu
48       * @param user Utilisateur qui joue
49       * @param bet Pari loué
50       * @param betAmount Montant misé
51       * @param dateOfOperation Date de la mise
52       * @param status Statut du pari
53       */
54      public BetInstance(BetChoice betChoice, User user, Bet bet,
55          double betAmount, Date dateOfOperation, int status) {
56          this.betChoice = betChoice;
57          this.user = user;
58          this.bet = bet;
59          this.betAmount = betAmount;
60          this.dateOfOperation = dateOfOperation;
61          this.status = status;
62      }
63
64      /**
65       * Obtient l'id du jeu
66       * @return L'id du jeu
67       */
68      @Id
69      @GeneratedValue(strategy = IDENTITY)
70      @Column(name = "id", unique = true, nullable = false)
71      public Integer getId() {
72          return this.id;
73      }
74
75      /**
76       * Modifie l'id du jeu
77       * @param id Nouvel id du jeu

```

```

78     */
79     public void setId(Integer id) {
80         this.id = id;
81     }
82
83     /**
84      * Obtient le choix du jeu
85      * @return Le choix de jeu
86      */
87     @ManyToOne(fetch = FetchType.EAGER)
88     @JoinColumn(name = "choice", nullable = false)
89     public BetChoice getBetChoice() {
90         return this.betChoice;
91     }
92
93     /**
94      * Modifie le choix du jeu
95      * @param betChoice Nouveau choix du jeu
96      */
97     public void setBetChoice(BetChoice betChoice) {
98         this.betChoice = betChoice;
99     }
100
101     /**
102      * Obtient l'utilisateur jouant
103      * @return L'utilisateur jouant
104      */
105     @ManyToOne(fetch = FetchType.EAGER)
106     @JoinColumn(name = "user", nullable = false)
107     public User getUser() {
108         return this.user;
109     }
110
111     /**
112      * Modifie l'utilisateur jouant
113      * @param user L'utilisateur jouant
114      */
115     public void setUser(User user) {
116         this.user = user;
117     }
118
119     /**
120      * Obtient la pari joué
121      * @return Le pari joué
122      */
123     @ManyToOne(fetch = FetchType.EAGER)
124     @JoinColumn(name = "bet", nullable = false)
125     public Bet getBet() {
126         return this.bet;
127     }
128
129     /**
130      * Modifie le pari joué
131      * @param bet Nouveau pari joué
132      */
133     public void setBet(Bet bet) {

```

```

134     this.bet = bet;
135 }
136
137 /**
138  * Obtient la somme misee
139  * @return La somme misee
140  */
141 @Column(name = "betAmount", nullable = false, precision = 22, scale = 0)
142 public double getBetAmount() {
143     return this.betAmount;
144 }
145
146 /**
147  * Modifie la somme misee
148  * @param betAmount Nouvelle somme à miser
149  */
150 public void setBetAmount(double betAmount) {
151     this.betAmount = betAmount;
152 }
153
154 /**
155  * Obtient la date de première participation pari
156  * @return La date de première participation pari
157  */
158 @Temporal(TemporalType.TIMESTAMP)
159 @Column(name = "dateOfOperation", nullable = false, length = 19)
160 public Date getDateOfOperation() {
161     return this.dateOfOperation;
162 }
163
164 /**
165  * Modifie la date de première participation pari
166  * @param dateOfOperation Nouvelle date
167  */
168 public void setDateOfOperation(Date dateOfOperation) {
169     this.dateOfOperation = dateOfOperation;
170 }
171
172 @Transient
173 public String getDateOfOperationFormatted() {
174     return dateLongFormatter.format(dateOfOperation);
175 }
176
177 public void setDateOfOperationFormatted() {
178
179 }
180
181 /**
182  * Obtient le statut actuel du pari
183  * @return Le statut actuel du pari
184  */
185 @Column(name = "status", nullable = false)
186 public int getStatus() {
187     return this.status;
188 }
189

```

```

190  /**
191   * Modifie le statut actuel du pari
192   * @param status Nouveau statut du pari
193   */
194  public void setStatus(int status) {
195      if (status == BetInstanceStatus.playing
196          || status == BetInstanceStatus.closed
197          || status == BetInstanceStatus.canceled) {
198          this.status = status;
199      }
200  }
201
202
203  public void setAmount(double amount) {
204      this.amount=amount;
205  }
206
207
208  @Transient
209  public double getAmount() {
210      return this.amount;
211  }
212
213
214  }

```

#### B.2.4 User

```

1  *package com.boc.botv.model;
2
3  import com.boc.botv.common.UserStatus;
4  import com.boc.botv.common.UserTitle;
5  import java.text.SimpleDateFormat;
6  import java.util.Date;
7  import java.util.HashSet;
8  import java.util.Set;
9  import javax.persistence.CascadeType;
10 import javax.persistence.Column;
11 import javax.persistence.Entity;
12 import javax.persistence.FetchType;
13 import javax.persistence.GeneratedValue;
14 import static javax.persistence.GenerationType.IDENTITY;
15 import javax.persistence.Id;
16 import javax.persistence.OneToOne;
17 import javax.persistence.Table;
18 import javax.persistence.Transient;
19 import javax.persistence.Temporal;
20 import javax.persistence.TemporalType;
21 import javax.persistence.UniqueConstraint;
22
23 /**
24  * Représente un utilisateur connecté du site
25  * @author Fabien Renaud
26  */
27 @Entity

```

```

28 @Table(name = "User", catalog = "botv", uniqueConstraints = {
29     @UniqueConstraint(columnNames = "login"))
30 public class User implements java.io.Serializable {
31
32     private static final SimpleDateFormat dateFormatter =
33         new SimpleDateFormat("dd/MM/yyyy");
34     private static final SimpleDateFormat dateLongFormatter =
35         new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
36     /**
37      * Id de l'utilisateur
38      */
39     private Integer id;
40     private String login;
41     private String password;
42     private String email;
43     private int title;
44     private String firstName;
45     private String lastName;
46     private String address0;
47     private String address1;
48     private String address2;
49     private String zipCode;
50     private String city;
51     private String country;
52     private String phoneHome;
53     private String phoneGsm;
54     private Date birthDate;
55     private double balance;
56     private String ip;
57     private int status;
58     private Date creationTime;
59     private Date lastModification;
60     private Date lastConnection;
61     private boolean acceptNewsletterSite;
62     private boolean acceptNewsletterPartners;
63     private Set<BetInstance> betInstances = new HashSet<BetInstance>(0);
64     private Set<Transaction> transactions = new HashSet<Transaction>(0);
65     private Set<PaymentMethod> paymentMethods = new HashSet<PaymentMethod>(0);
66     private Set<Bet> bets = new HashSet<Bet>(0);
67     // Transient
68     private String checkPassword;
69     private boolean termsAccepted;
70     private String newPassword;
71     private String checkNewPassword;
72
73     /**
74      * Crée un utilisateur par défaut
75      */
76     public User() {
77     }
78
79     /**
80      * Crée un utilisateur
81      * @param login Pseudo de l'utilisateur
82      * @param password Mot de passe crypté de l'utilisateur
83      * @param email Courriel de l'utilisateur

```

```

84  * @param title   Civilité de l'utilisateur
85  * @param firstName Prénom
86  * @param lastName Nom
87  * @param address0 Première ligne de son adresse
88  * @param address1 Deuxième ligne de son adresse
89  * @param address2 Troisième ligne de son adresse
90  * @param zipCode Code postal
91  * @param city    Ville
92  * @param country Pays
93  * @param phoneHome Téléphone fixe
94  * @param phoneGsm Téléphone mobile
95  * @param birthDate Date de naissance
96  * @param balance Solde du compte utilisateur
97  * @param ip Adresse IP lors de l'inscription
98  * @param status Statut de l'utilisateur
99  * @param creationTime Date de création du compte
100 * @param lastModification Dernière modification des informations du
101 * compte par l'utilisateur
102 * @param lastConnection Dernière date de connexion de l'utilisateur
103 * @param acceptNewsletterSite true si accepte de recevoir les newsletters
104 * du site
105 * @param acceptNewsletterPartners true si accepte de recevoir les newsletters
106 * des partenaires du site
107 */
108 public User(String login, String password, String email, int title,
109             String firstName, String lastName, String address0, String address1,
110             String address2, String zipCode, String city, String country, String phoneHome,
111             String phoneGsm, Date birthDate, double balance, String ip, int status,
112             Date creationTime, Date lastModification, Date lastConnection,
113             boolean acceptNewsletterSite, boolean acceptNewsletterPartners) {
114     this.login = login;
115     this.password = password;
116     this.email = email;
117     this.title = title;
118     this.firstName = firstName;
119     this.lastName = lastName;
120     this.address0 = address0;
121     this.address1 = address1;
122     this.address2 = address2;
123     this.zipCode = zipCode;
124     this.city = city;
125     this.country = country;
126     this.birthDate = birthDate;
127     this.balance = balance;
128     this.status = status;
129     this.creationTime = creationTime;
130     this.lastModification = lastModification;
131     this.lastConnection = lastConnection;
132     this.acceptNewsletterSite = acceptNewsletterSite;
133     this.acceptNewsletterPartners = acceptNewsletterPartners;
134     this.ip = checkIp(ip) ? ip : "0.0.0.0";
135     this.phoneHome = checkPhone(phoneHome) ? phoneHome : "";
136     this.phoneGsm = checkPhone(phoneGsm) ? phoneGsm : "";
137 }
138
139 /**

```

```

140 * Crée un utilisateur
141 * @param login Pseudo de l'utilisateur
142 * @param password Mot de passe crypté de l'utilisateur
143 * @param email Courriel de l'utilisateur
144 * @param title Civilité de l'utilisateur
145 * @param firstName Prénom
146 * @param lastName Nom
147 * @param address0 Première ligne de son adresse
148 * @param address1 Deuxième ligne de son adresse
149 * @param address2 Troisième ligne de son adresse
150 * @param zipCode Code postal
151 * @param city Ville
152 * @param country Pays
153 * @param phoneHome Téléphone fixe
154 * @param phoneGsm Téléphone mobile
155 * @param birthDate Date de naissance
156 * @param balance Solde du compte utilisateur
157 * @param ip Adresse IP lors de l'inscription
158 * @param status Statut de l'utilisateur
159 * @param creationTime Date de création du compte
160 * @param lastModification Dernière modification des informations du compte
161 * par l'utilisateur
162 * @param lastConnection Dernière date de connexion de l'utilisateur
163 * @param acceptNewsletterSite true si accepte de recevoir les newsletters du site
164 * @param acceptNewsletterPartners true si accepte de recevoir les newsletters
165 * des partenaires du site
166 * @param betInstances Paris auxquels à joué l'utilisateur
167 * @param transactions Transactions liées au compte utilisateur
168 * @param bets Paris joués
169 * @param bankTransactions Transactions bancaires
170 */
171 public User(String login, String password, String email, int title,
172 String firstName, String lastName, String address0, String address1,
173 String address2, String zipCode, String city, String country, String phoneHome,
174 String phoneGsm, Date birthDate, double balance, String ip, int status,
175 Date creationTime, Date lastModification, Date lastConnection,
176 boolean acceptNewsletterSite, boolean acceptNewsletterPartners,
177 Set<BetInstance> betInstances, Set<Transaction> transactions,
178 Set<PaymentMethod> paymentMethods, Set<Bet> bets) {
179 this.login = login;
180 this.password = password;
181 this.email = email;
182 this.title = title;
183 this.firstName = firstName;
184 this.lastName = lastName;
185 this.address0 = address0;
186 this.address1 = address1;
187 this.address2 = address2;
188 this.zipCode = zipCode;
189 this.city = city;
190 this.country = country;
191 this.birthDate = birthDate;
192 this.balance = balance;
193 this.status = status;
194 this.creationTime = creationTime;
195 this.lastModification = lastModification;

```

```

196     this.lastConnection = lastConnection;
197     this.acceptNewsletterSite = acceptNewsletterSite;
198     this.acceptNewsletterPartners = acceptNewsletterPartners;
199     this.betInstances = betInstances;
200     this.transactions = transactions;
201     this.paymentMethods = paymentMethods;
202     this.bets = bets;
203     this.ip = checkIp(ip) ? ip : "0.0.0.0";
204     this.phoneHome = checkPhone(phoneHome) ? phoneHome : "";
205     this.phoneGsm = checkPhone(phoneGsm) ? phoneGsm : "";
206 }
207
208 /**
209  * Obtient l'id de l'utilisateur
210  * @return L'id de l'utilisateur
211  */
212 @Id
213 @GeneratedValue(strategy = IDENTITY)
214 @Column(name = "id", unique = true, nullable = false)
215 public Integer getId() {
216     return this.id;
217 }
218
219 /**
220  * Modifie l'id de l'utilisateur
221  * @param id Nouvel id
222  */
223 public void setId(Integer id) {
224     this.id = id;
225 }
226
227 /**
228  * Obtient le pseudo de l'utilisateur
229  * @return Le pseudo de l'utilisateur
230  */
231 @Column(name = "login", unique = true, nullable = false, length = 32)
232 public String getLogin() {
233     return login;
234 }
235
236 /**
237  * Modifie le pseudo de l'utilisateur
238  * @param login Nouveau pseudo
239  */
240 public void setLogin(String login) {
241     this.login = login;
242 }
243
244 /**
245  * Obtient le mot de passe crypté de l'utilisateur
246  * @return Le mot de passe crypté
247  */
248 @Column(name = "password", nullable = false, length = 64)
249 public String getPassword() {
250     return password;
251 }

```

```

252
253 /**
254  * Modifie le mot de passe de l'utilisateur
255  * @param password Nouveau mot de passe crypté
256  */
257 public void setPassword(String password) {
258     this.password = password;
259 }
260
261 /**
262  * Obtient le courriel de l'utilisateur
263  * @return Le courriel de l'utilisateur
264  */
265 @Column(name = "email", nullable = false, length = 64)
266 public String getEmail() {
267     return this.email;
268 }
269
270 /**
271  * Modifie le courriel de l'utilisateur
272  * @param email Nouveau courriel de l'utilisateur
273  */
274 public void setEmail(String email) {
275     this.email = email;
276 }
277
278 /**
279  * Obtient la civilité de l'utilisateur
280  * @return La civilité de l'utilisateur
281  */
282 @Column(name = "title", nullable = false)
283 public int getTitle() {
284     return this.title;
285 }
286
287 /**
288  * Modifie la civilité de l'utilisateur
289  * @param title Nouvelle civilté de l'utilisateur
290  */
291 public void setTitle(int title) {
292     if (title == UserTitle.monsieur
293         || title == UserTitle.madame
294         || title == UserTitle.mademoiselle) {
295         this.title = title;
296     }
297 }
298
299 /**
300  * Obtient le prénom de l'utilisateur
301  * @return Le prénom de l'utilisateur
302  */
303 @Column(name = "firstName", nullable = false, length = 32)
304 public String getFirstName() {
305     return this.firstName;
306 }
307

```

```

308  /**
309   * Modifie le prénom de l'utilisateur
310   * @param firstName Nouveau prénom de l'utilisateur
311   */
312  public void setFirstName(String firstName) {
313      this.firstName = firstName;
314  }
315
316  /**
317   * Obtient le nom de l'utilisateur
318   * @return Le nom de l'utilisateur
319   */
320  @Column(name = "lastName", nullable = false, length = 32)
321  public String getLastName() {
322      return this.lastName;
323  }
324
325  /**
326   * Modifie le nom de l'utilisateur
327   * @param lastName Nouveau nom de l'utilisateur
328   */
329  public void setLastName(String lastName) {
330      this.lastName = lastName;
331  }
332
333  /**
334   * Obtient la première ligne de l'adresse de l'utilisateur
335   * @return La première ligne de l'adresse de l'utilisateur
336   */
337  @Column(name = "address0", nullable = false, length = 48)
338  public String getAddress0() {
339      return this.address0;
340  }
341
342  /**
343   * Modifie la première ligne de l'adresse de l'utilisateur
344   * @param address0 Nouvelle première ligne de l'adresse de l'utilisateur
345   */
346  public void setAddress0(String address0) {
347      this.address0 = address0;
348  }
349
350  /**
351   * Obtient la deuxième ligne de l'adresse de l'utilisateur
352   * @return La deuxième ligne de l'adresse de l'utilisateur
353   */
354  @Column(name = "address1", nullable = false, length = 48)
355  public String getAddress1() {
356      return this.address1;
357  }
358
359  /**
360   * Modifie la deuxième ligne de l'adresse de l'utilisateur
361   * @param address1 Nouvelle deuxième ligne de l'adresse de l'utilisateur
362   */
363  public void setAddress1(String address1) {

```

```

364     this.address1 = address1;
365 }
366
367 /**
368  * Obtient la troisième ligne de l'adresse de l'utilisateur
369  * @return La troisième ligne de l'adresse de l'utilisateur
370  */
371 @Column(name = "address2", nullable = false, length = 48)
372 public String getAddress2() {
373     return this.address2;
374 }
375
376 /**
377  * Modifie la troisième ligne de l'adresse de l'utilisateur
378  * @param address2 Nouvelle troisième ligne de l'adresse de l'utilisateur
379  */
380 public void setAddress2(String address2) {
381     this.address2 = address2;
382 }
383
384 /**
385  * Obtient le code postal de l'utilisateur
386  * @return Le code postal de l'utilisateur
387  */
388 @Column(name = "zipCode", nullable = false, length = 5)
389 public String getZipCode() {
390     return this.zipCode;
391 }
392
393 /**
394  * Modifie le code postal de l'utilisateur
395  * @param zipCode Nouveau code postal de l'utilisateur
396  */
397 public void setZipCode(String zipCode) {
398     this.zipCode = zipCode;
399 }
400
401 /**
402  * Obtient la ville de l'utilisateur
403  * @return La ville de l'utilisateur
404  */
405 @Column(name = "city", nullable = false, length = 64)
406 public String getCity() {
407     return this.city;
408 }
409
410 /**
411  * Modifie la ville de l'utilisateur
412  * @param city Nouvelle ville de l'utilisateur
413  */
414 public void setCity(String city) {
415     this.city = city;
416 }
417
418 /**
419  * Obtient le pays de l'utilisateur

```

```

420     * @return Le pays de l'utilisateur
421     */
422     @Column(name = "country", nullable = false, length = 32)
423     public String getCountry() {
424         return this.country;
425     }
426
427     /**
428     * Modifie le pays de l'utilisateur
429     * @param country Nouveau pays de l'utilisateur
430     */
431     public void setCountry(String country) {
432         this.country = country;
433     }
434
435     /**
436     * Obtient le téléphone fixe de l'utilisateur
437     * @return Le téléphone fixe de l'utilisateur
438     */
439     @Column(name = "phoneHome", nullable = false, length = 12)
440     public String getPhoneHome() {
441         return this.phoneHome;
442     }
443
444     /**
445     * Modifie le téléphone fixe de l'utilisateur
446     * @param phoneHome Nouveau téléphone fixe de l'utilisateur.
447     * Doit être au format 017654321 ou +3347654321
448     */
449     public void setPhoneHome(String phoneHome) {
450         this.phoneHome = checkPhone(phoneHome) ? phoneHome : "";
451     }
452
453     /**
454     * Obtient le téléphone mobile de l'utilisateur
455     * @return Le téléphone mobile de l'utilisateur
456     */
457     @Column(name = "phoneGsm", nullable = false, length = 12)
458     public String getPhoneGsm() {
459         return this.phoneGsm;
460     }
461
462     /**
463     * Modifie le téléphone mobile de l'utilisateur
464     * @param phoneGsm Nouveau téléphone mobile de l'utilisateur.
465     * Doit être au format 067654321 ou +3367654321
466     */
467     public void setPhoneGsm(String phoneGsm) {
468         this.phoneGsm = checkPhone(phoneGsm) ? phoneGsm : "";
469     }
470
471     /**
472     * Obtient la date de naissance de l'utilisateur
473     * @return La date de naissance de l'utilisateur
474     */
475     @Temporal(TemporalType.DATE)

```

```

476 @Column(name = "birthDate", nullable = false, length = 10)
477 public Date getBirthDate() {
478     return this.birthDate;
479 }
480
481 /**
482  * Modifie la date de naissance de l'utilisateur
483  * @param birthDate Nouvelle date de naissance de l'utilisateur
484  */
485 public void setBirthDate(Date birthDate) {
486     this.birthDate = birthDate;
487 }
488
489 /**
490  * Obtient le solde du compte de l'utilisateur
491  * @return Le solde du compte de l'utilisateur
492  */
493 @Column(name = "balance", nullable = false, precision = 22, scale = 0)
494 public double getBalance() {
495     return this.balance > 0 ? this.balance : 0;
496 }
497
498 /**
499  * Modifie le solde du compte de l'utilisateur
500  * @param balance Nouveau solde. Doit être positif.
501  */
502 public void setBalance(double balance) {
503     this.balance = balance > 0 ? balance : 0;
504 }
505
506 /**
507  * Obtient l'adresse IP avec laquelle l'utilisateur s'est inscrit
508  * @return L'adresse IP avec laquelle l'utilisateur s'est inscrit
509  */
510 @Column(name = "ip", nullable = false, length = 15)
511 public String getIp() {
512     return this.ip != null ? ip : "0.0.0.0";
513 }
514
515 /**
516  * Modifie l'adresse IP avec laquelle l'utilisateur s'est inscrit
517  * @param ip Nouvelle adresse IP. Doit être au format 221.78.97.234
518  */
519 public void setIp(String ip) {
520     this.ip = checkIp(ip) ? ip : "";
521 }
522
523 /**
524  * Obtient le statut du compte utilisateur
525  * @return Le statut du compte utilisateur
526  */
527 @Column(name = "status", nullable = false)
528 public int getStatus() {
529     return this.status;
530 }
531

```

```

532  /**
533   * Modifie le statut du compte utilisateur
534   * @param status Nouveau statut
535   */
536  public void setStatus(int status) {
537      if (status == UserStatus.disabled
538          || status == UserStatus.enabled
539          || status == UserStatus.banned) {
540          this.status = status;
541      }
542  }
543
544  /**
545   * Obtient la date de création du compte utilisateur
546   * @return La date de création du compte utilisateur
547   */
548  @Temporal(TemporalType.TIMESTAMP)
549  @Column(name = "creationTime", nullable = false, length = 19)
550  public Date getCreationTime() {
551      return creationTime;
552  }
553
554  /**
555   * Modifie la date de création du compte utilisateur
556   * @param creationTime Nouvelle date
557   */
558  public void setCreationTime(Date creationTime) {
559      this.creationTime = creationTime;
560  }
561
562  /**
563   * Obtient la date de dernière modification des informations du compte
564   * utilisateur par l'utilisateur
565   * @return La date de dernière modification des informations du compte
566   * utilisateur par l'utilisateur
567   */
568  @Temporal(TemporalType.TIMESTAMP)
569  @Column(name = "lastModification", nullable = false, length = 19)
570  public Date getLastModification() {
571      return lastModification;
572  }
573
574  /**
575   * Modifie la date de dernière modification des informations du compte
576   * utilisateur par l'utilisateur
577   * @param lastModification Dernière date de modification
578   */
579  public void setLastModification(Date lastModification) {
580      this.lastModification = lastModification;
581  }
582
583  /**
584   * Obtient la date de dernière connexion de l'utilisateur
585   * @return La date de dernière connexion de l'utilisateur
586   */
587  @Temporal(TemporalType.TIMESTAMP)

```

```

588 @Column(name = "lastConnection", nullable = false, length = 19)
589 public Date getLastConnection() {
590     return lastConnection;
591 }
592
593 /**
594  * Modifie la date de dernière connection de l'utilisateur
595  * @param lastConnection Date de dernière connection
596  */
597 public void setLastConnection(Date lastConnection) {
598     this.lastConnection = lastConnection;
599 }
600
601 /**
602  * Obtient si l'utilisateur accepte de recevoir la newsletter du site ou non
603  * @return true si l'utilistaeur accepte de recevoir la newsletter du site
604  */
605 @Column(name = "acceptNewsletterSite", nullable = false)
606 public boolean isAcceptNewsletterSite() {
607     return this.acceptNewsletterSite;
608 }
609
610 /**
611  * Modifie le fait que l'utilisateur veuille recevoir la newsletter du
612  * site ou non
613  * @param acceptNewsletterSite true si l'utilisateur veut désormais
614  * recevoir la newsletter du site
615  */
616 public void setAcceptNewsletterSite(boolean acceptNewsletterSite) {
617     this.acceptNewsletterSite = acceptNewsletterSite;
618 }
619
620 /**
621  * Obtient si l'utilisateur accepte de recevoir la newsletter des
622  * partenaires du site ou non
623  * @return true si l'utilistaeur accepte de recevoir la newsletter
624  * des partenaires du site
625  */
626 @Column(name = "acceptNewsletterPartners", nullable = false)
627 public boolean isAcceptNewsletterPartners() {
628     return this.acceptNewsletterPartners;
629 }
630
631 /**
632  * Modifie le fait que l'utilisateur veuille recevoir la newsletter
633  * des partenaires du site ou non
634  * @param acceptNewsletterPartners true si l'utilisateur veut désormais
635  * recevoir la newsletter des partenaires du site
636  */
637 public void setAcceptNewsletterPartners(boolean acceptNewsletterPartners) {
638     this.acceptNewsletterPartners = acceptNewsletterPartners;
639 }
640
641 /**
642  * Obtient tous les jeux que l'utilisateur a effectué
643  * @return Les jeux que l'utilisateur a effectué

```

```

644     */
645     @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy = "user")
646     public Set<BetInstance> getBetInstances() {
647         return this.betInstances;
648     }
649
650     /**
651      * Modifie les jeux que l'utilisateur a effectué
652      * @param betInstances Les nouveaux jeux que l'utilisateur a effectué
653      */
654     public void setBetInstances(Set<BetInstance> betInstances) {
655         this.betInstances = betInstances;
656     }
657
658     /**
659      * Obtient toutes les transactions site concernant cet utilisateur
660      * @return Les transactions site concernant cet utilisateur
661      */
662     @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy = "user")
663     public Set<Transaction> getTransactions() {
664         return this.transactions;
665     }
666
667     /**
668      * Modifie les transactions site concernant cet utilisateur
669      * @param transactions Nouvelles transactions site concernant cet utilisateur
670      */
671     public void setTransactions(Set<Transaction> transactions) {
672         this.transactions = transactions;
673     }
674
675     /**
676      * Obtient tous les paris auxquels cet utilisateur a joué
677      * @return Les paris auxquels cet utilisateur a joué
678      */
679     @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy = "user")
680     public Set<Bet> getBets() {
681         return this.bets;
682     }
683
684     @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER, mappedBy = "user")
685     public Set<PaymentMethod> getPaymentMethods() {
686         return this.paymentMethods;
687     }
688
689     public void setPaymentMethods(Set<PaymentMethod> paymentMethods) {
690         this.paymentMethods = paymentMethods;
691     }
692
693     /**
694      * Modifie les paris auxquels cet utilisateur a joué
695      * @param bets Nouveau paris auxquels cet utilisateur a joué
696      */
697     public void setBets(Set<Bet> bets) {
698         this.bets = bets;
699     }

```

```

700
701 /**
702  * Ajoute un montant au solde du compte
703  * @param amount Montant à ajouter au solde du compte
704  */
705 public void increaseBalance(double amount) {
706     this.balance += amount;
707 }
708
709 /**
710  * Retire un montant au solde du compte
711  * @param amount Montant à retirer au solde du compte
712  */
713 public void decreaseBalance(double amount) {
714     this.balance -= amount;
715 }
716
717 public void setBirthDateFormatted(String s) {
718 }
719
720 public void setCreationTimeFormatted(String s) {
721 }
722
723 public void setLastConnectionFormatted(String s) {
724 }
725
726 public void setLastModificationFormatted(String s) {
727 }
728
729 @Transient
730 public String getBirthDateFormatted() {
731     return dateFormatter.format(birthDate);
732 }
733
734 @Transient
735 public String getCreationTimeFormatted() {
736     return dateLongFormatter.format(creationTime);
737 }
738
739 @Transient
740 public String getLastConnectionFormatted() {
741     return dateLongFormatter.format(lastConnection);
742 }
743
744 @Transient
745 public String getLastModificationFormatted() {
746     return dateLongFormatter.format(lastModification);
747 }
748
749 @Transient
750 public PaymentMethod getPaymentMethod() {
751     if (paymentMethods.size() != 1) {
752         return null;
753     }
754     return (PaymentMethod) paymentMethods.toArray()[0];
755 }

```

```

756
757 /**
758  * Obtient la confirmation du mot de passe
759  * @return La confirmation du mot de passe
760  */
761 @Transient
762 public String getCheckPassword() {
763     return checkPassword;
764 }
765
766 /**
767  * Modifie la confirmation du mot de passe
768  * @param checkPassword Nouvelle confirmation du mot de passe
769  */
770 public void setCheckPassword(String checkPassword) {
771     this.checkPassword = checkPassword;
772 }
773
774 /**
775  * Permet de savoir si l'utilisateur a accepté les CGU.
776  * @return true si l'utilisateur a accepté les CGU, false sinon.
777  */
778 @Transient
779 public boolean isTermsAccepted() {
780     return termsAccepted;
781 }
782
783 /**
784  * Modifie l'état d'acceptation des CGU.
785  * @param termsAccepted true si l'utilisateur accepte les CGU, false sinon.
786  */
787 public void setTermsAccepted(boolean termsAccepted) {
788     this.termsAccepted = termsAccepted;
789 }
790
791 /**
792  * Obtient le nouveau mot de passe
793  * @return La confirmation du mot de passe
794  */
795 @Transient
796 public String getNewPassword() {
797     return newPassword;
798 }
799
800 /**
801  * Modifie le nouveau mot de passe
802  * @param password Nouveau mot de passe
803  */
804 public void setNewPassword(String password) {
805     this.newPassword = password;
806 }
807
808 /**
809  * Obtient la confirmation du nouveau mot de passe
810  * @return La confirmation du mot de passe
811  */

```

```

812  @Transient
813  public String getCheckNewPassword() {
814      return checkNewPassword;
815  }
816
817  /**
818   * Modifie la confirmation du nouveau mot de passe
819   * @param checkPassword Nouvelle confirmation du mot de passe
820   */
821  public void setCheckNewPassword(String checkPassword) {
822      this.checkNewPassword = checkPassword;
823  }
824
825  private boolean checkPhone(String phone) {
826      if (phone == null) {
827          return false;
828      }
829      String ph = phone;
830      if (ph.length() == 12) {
831          if (ph.charAt(0) != '+') {
832              return false;
833          }
834      } else if (ph.length() == 10) {
835          if (ph.charAt(0) != '0') {
836              return false;
837          }
838      } else {
839          return false;
840      }
841      ph = ph.substring(1);
842      try {
843          Long.parseLong(ph);
844      } catch (NumberFormatException e) {
845          return false;
846      }
847      return true;
848  }
849
850  private boolean checkIp(String ip) {
851      if (ip == null) {
852          return false;
853      }
854      String[] array = ip.split("\\.");
855      if (array.length != 4) {
856          return false;
857      }
858      try {
859          for (int i = 0; i < 4; i++) {
860              int n = Integer.parseInt(array[i]);
861              if (n < 0 || n > 255) {
862                  return false;
863              }
864          }
865      } catch (NumberFormatException e) {
866          return false;
867      }

```

```

868     return true;
869 }
870 }

```

### B.2.5 UserRememberMe

```

1  *package com.boc.botv.model;
2
3  import com.boc.botv.common.UserTitle;
4
5  /**
6   * Représente un utilisateur côté client (ie. une session utilisateur minimaliste
7   * @author Fabien Renaud
8   */
9  public class UserRememberMe {
10     private Integer id;
11     private String login;
12     private String title;
13     private String firstname;
14     private String lastname;
15     private String balance;
16
17     public UserRememberMe(User user) {
18         this.id = user.getId();
19         this.login = user.getLogin();
20         this.title = UserTitle.toShortString(user.getTitle());
21         this.firstname = user.getFirstName();
22         this.lastname = user.getLastName();
23         this.balance = String.format("%.2f", user.getBalance());
24     }
25
26     /**
27      * Obtient le solde du compte de l'utilisateur
28      * @return Le solde du compte de l'utilisateur
29      */
30     public String getBalance() {
31         return balance;
32     }
33
34     /**
35      * Obtient le prénom de l'utilisateur
36      * @return Le prénom de l'utilisateur
37      */
38     public String getFirstname() {
39         return firstname;
40     }
41
42     /**
43      * Obtient l'id de l'utilisateur
44      * @return L'id de l'utilisateur
45      */
46     public Integer getId() {
47         return id;
48     }
49

```

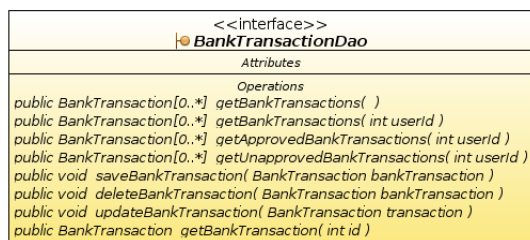
```

50  /**
51   * Obtient le nom de l'utilisateur
52   * @return Le nom de l'utilisateur
53   */
54  public String getLastname() {
55      return lastname;
56  }
57
58  /**
59   * Obtient le pseudo de l'utilisateur
60   * @return Le pseudo de l'utilisateur
61   */
62  public String getLogin() {
63      return login;
64  }
65
66  /**
67   * Obtient la civilité de l'utilisateur
68   * @return La civilité de l'utilisateur
69   */
70  public String getTitle() {
71      return title;
72  }
73
74  public void setBalance(double balance) {
75      this.balance = String.format("%.2f", balance);
76  }
77  }

```

## B.3 Data Access Objects

### B.3.1 BankTransactionDao



Listing 13 – Implémentation du DAO

```

1  *package com.boc.botv.dao.impl;
2
3  import com.boc.botv.common.Database;
4  import com.boc.botv.dao.BankTransactionDao;
5  import com.boc.botv.model.BankTransaction;
6  import java.util.List;
7  import org.springframework.stereotype.Repository;
8
9  @Repository("bankTransactionDao")
10 public class BankTransactionDaoImpl implements BankTransactionDao {
11


```

```

12 public List<BankTransaction> getBankTransactions() {
13     return (List<BankTransaction>) Database.getSession().createQuery("from " +
14         "BankTransaction").list();
15 }
16
17 public List<BankTransaction> getBankTransactions(int userId) {
18     return (List<BankTransaction>) Database.getSession().createQuery("from " +
19         "BankTransaction as bktr where bktr.user =" + userId + "'").list();
20 }
21
22 public List<BankTransaction> getApprovedBankTransactions(int userId) {
23     String req = "from BankTransaction as bktr where (bktr.user =" +
24         + userId + "' ) and (bktr.isApproved = 'true')";
25     return (List<BankTransaction>) Database.getSession().createQuery(req).list();
26 }
27
28 public List<BankTransaction> getUnapprovedBankTransactions(int userId) {
29     String req = "from BankTransaction as bktr where (bktr.user =" +
30         + userId + "' ) and (bktr.isApproved = 'false')";
31     return (List<BankTransaction>) Database.getSession().createQuery(req).list();
32 }
33
34 public void saveBankTransaction(BankTransaction bankTransaction) {
35     Database.getSession().save(bankTransaction);
36 }
37
38 public void deleteBankTransaction(BankTransaction bankTransaction) {
39     Database.getSession().delete(bankTransaction);
40 }
41
42 public void updateBankTransaction(BankTransaction banktransaction) {
43     Database.getSession().update(banktransaction);
44 }
45
46 public BankTransaction getBankTransaction(int id) {
47     return (BankTransaction) Database.getSession().load(
48         BankTransaction.class, id);
49 }
50 }

```

### B.3.2 BetDao

<<interface>>
 <b>BetDao</b>
Attributes
Operations
<pre> public Bet[0..*] getBets( ) public Bet getBet( Integer betid ) public void deleteBet( Bet bet ) public void saveBet( Bet bet ) public void updateBet( Bet bet ) public Bet[0..*] getBetsBySubsection( int subsection ) public Bet[0..*] getBetsByCreationTime( Date firstDate, Date lastDate ) public Bet[0..*] getBetsByStartTime( Date firstDate, Date lastDate ) public Bet[0..*] getBetsByEndTime( Date firstDate, Date lastDate ) public Bet[0..*] getBetsPlayedBetween( Date firstDate, Date lastDate ) public Bet[0..*] getBetsNotPlayedBetween( Date firstDate, Date lastDate ) public Bet[0..*] getBetsByCloserEndDate( int numberOfBet, int subsection ) public Bet[0..*] getCurrentBetsBySubsection( int subsection ) </pre>

```

1  *package com.boc.botv.dao.impl;
2
3  import com.boc.botv.common.Database;
4  import com.boc.botv.dao.BetDao;
5  import com.boc.botv.model.Bet;
6  import java.text.SimpleDateFormat;
7  import java.util.Date;
8  import java.util.List;
9  import org.springframework.stereotype.Repository;
10
11  @Repository("betDao")
12  public class BetDaoImpl implements BetDao {
13
14      private static final SimpleDateFormat dateFormatter =
15          new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
16
17      public List<Bet> getBets() {
18          return (List<Bet>) Database.getSession().createQuery("from Bet").list();
19      }
20
21      public Bet getBet(Integer betId) {
22          return (Bet) Database.getSession().load(Bet.class, betId);
23      }
24
25      public void deleteBet(Bet bet) {
26          Database.getSession().delete(bet);
27      }
28
29      public void saveBet(Bet bet) {
30          Database.getSession().save(bet);
31      }
32
33      public void updateBet(Bet bet) {
34          Database.getSession().update(bet);
35      }
36
37      public List<Bet> getBetsBySubsection(int subsection) {
38          return (List<Bet>) Database.getSession().createQuery("from Bet as b "
39              + "where b.subsection=" + subsection).list();
40      }
41
42      public List<Bet> getBetsByCreationTime(Date firstDate, Date lastDate) {
43          String t1 = dateFormatter.format(firstDate);
44          String t2 = dateFormatter.format(lastDate);
45          return (List<Bet>) Database.getSession().createQuery("from Bet as bet " +
46              "where bet.creationTime between '" + t1 + "' and '" + t2 + "'").list();
47      }
48
49      public List<Bet> getBetsByStartTime(Date firstDate, Date lastDate) {
50          String t1 = dateFormatter.format(firstDate);
51          String t2 = dateFormatter.format(lastDate);
52          return (List<Bet>) Database.getSession().createQuery("from Bet as bet " +
53              "where bet.startTime between '" + t1 + "' and '" + t2 + "'").list();
54      }
55

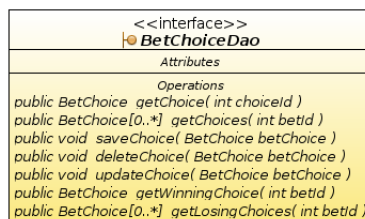
```

```

56 public List<Bet> getBetsByEndTime(Date firstDate , Date lastDate) {
57     String t1 = dateFormatter.format(firstDate);
58     String t2 = dateFormatter.format(lastDate);
59     return (List<Bet>) Database.getSession().createQuery("from Bet as bet " +
60         "where bet.endTime between '" + t1 + "' and '" + t2 + "'").list();
61 }
62
63 public List<Bet> getBetsPlayedBetween(Date firstDate , Date lastDate) {
64     String t1 = dateFormatter.format(firstDate);
65     String t2 = dateFormatter.format(lastDate);
66     // TODO
67     return null;
68 }
69
70 public List<Bet> getBetsNotPlayedBetween(Date firstDate , Date lastDate) {
71     String t1 = dateFormatter.format(firstDate);
72     String t2 = dateFormatter.format(lastDate);
73     // TODO
74     return null;
75 }
76
77 public List<Bet> getBetsByCloserEndDate(int numberOfBet , int subsection) {
78     return Database.getSession().createQuery("from Bet as b where b.subsection="
79         + subsection + " and b.endTime > NOW() order by b.endTime asc ").list();
80 }
81
82 public List<Bet> getCurrentBetsBySubsection(int subsection) {
83     return Database.getSession().createQuery("from Bet as b where b.subsection="
84         + subsection + " and b.endTime > NOW()").list();
85 }
86 }

```

### B.3.3 BetChoiceDao



Listing 15 – Implémentation du DAO

```

1 *package com.boc.botv.dao.impl;
2
3 import com.boc.botv.common.Database;
4 import com.boc.botv.dao.BetChoiceDao;
5 import com.boc.botv.model.BetChoice;
6 import java.util.List;
7 import org.springframework.stereotype.Repository;
8
9 @Repository("betChoiceDao")
10 public class BetChoiceDaoImpl implements BetChoiceDao {


```

```

11
12 public BetChoice getChoice(int choiceId) {
13     return (BetChoice) Database.getSession().load(BetChoice.class, choiceId);
14 }
15
16 public List<BetChoice> getChoices(int betId) {
17     return (List<BetChoice>) Database.getSession().createQuery("from BetChoice "
18         + "as bc where bc.bet=" + betId).list();
19 }
20
21 public void saveChoice(BetChoice betChoice) {
22     Database.getSession().save(betChoice);
23 }
24
25 public void deleteChoice(BetChoice betChoice) {
26     Database.getSession().delete(betChoice);
27 }
28
29 public void updateChoice(BetChoice betChoice) {
30     Database.getSession().update(betChoice);
31 }
32
33 public BetChoice getWinningChoice(int betId) {
34     List<BetChoice> bcl = (List<BetChoice>) Database.getSession().createQuery(
35         "from BetChoice as bc where bc.isWinning=1 and bc.bet=" + betId);
36     if (bcl.size() != 1) {
37         return null;
38     }
39     return bcl.get(0);
40 }
41
42 public List<BetChoice> getLosingChoices(int betId) {
43     return (List<BetChoice>) Database.getSession().createQuery(
44         "from BetChoice as bc where bc.isWinning=0 and bc.bet=" + betId).list();
45 }
46 }

```

### B.3.4 BetInstanceDao

<<interface>>  <b>BetInstanceDao</b>	
Attributes	
Operations	
<i>public void</i>	<i>updateGame( BetInstance betInstance )</i>
<i>public void</i>	<i>saveGame( BetInstance betInstance )</i>
<i>public void</i>	<i>deleteGame( BetInstance betInstance )</i>
<i>public BetInstance</i>	<i>getGame( int betInstanceid )</i>
<i>public BetInstance[0..*]</i>	<i>getGames( )</i>
<i>public BetInstance[0..*]</i>	<i>getGamesByStatus( int status )</i>
<i>public BetInstance[0..*]</i>	<i>getGamesByChoice( int choiceid )</i>
<i>public BetInstance[0..*]</i>	<i>getGamesByUser( int userid )</i>
<i>public BetInstance[0..*]</i>	<i>getGamesByUserAndStatus( int userid, int status )</i>
<i>public BetInstance[0..*]</i>	<i>getGamesByBet( int betid )</i>

Listing 16 – Implémentation du DAO

```

1 *package com.boc.botv.dao.impl;
2
3 import com.boc.botv.common.Database;

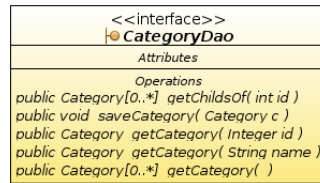
```

```

4 import com.boc.bottv.dao.BetInstanceDao;
5 import com.boc.bottv.model.BetInstance;
6 import java.util.List;
7 import org.springframework.stereotype.Repository;
8
9 @Repository("betInstanceDao")
10 public class BetInstanceDaoImpl implements BetInstanceDao {
11
12     public void updateGame(BetInstance betInstance) {
13         Database.getSession().update(betInstance);
14     }
15
16     public void saveGame(BetInstance betInstance) {
17         Database.getSession().save(betInstance);
18     }
19
20     public void deleteGame(BetInstance betInstance) {
21         Database.getSession().delete(betInstance);
22     }
23
24     public BetInstance getGame(int betInstanceId) {
25         return (BetInstance) Database.getSession().load(
26             BetInstance.class, betInstanceId);
27     }
28
29     public List<BetInstance> getGames() {
30         return (List<BetInstance>) Database.getSession().
31             createQuery("from BetInstance").list();
32     }
33
34     public List<BetInstance> getGamesByStatus(int status) {
35         return (List<BetInstance>) Database.getSession().createQuery("from " +
36             "BetInstance as beti where beti.status=" + status).list();
37     }
38
39     public List<BetInstance> getGamesByChoice(int choiceId) {
40         return (List<BetInstance>) Database.getSession().createQuery("from " +
41             "BetInstance as beti where beti.choice=" + choiceId).list();
42     }
43
44     public List<BetInstance> getGamesByUser(int userId) {
45         return (List<BetInstance>) Database.getSession().createQuery("from " +
46             "BetInstance as beti where beti.user=" + userId).list();
47     }
48
49     public List<BetInstance> getGamesByUserAndStatus(int userId, int status) {
50         return (List<BetInstance>) Database.getSession().createQuery("from " +
51             "BetInstance as beti where beti.user=" + userId +
52             " and beti.status=" + status).list();
53     }
54
55     public List<BetInstance> getGamesByBet(int betId) {
56         return (List<BetInstance>) Database.getSession().createQuery("from " +
57             "BetInstance as beti where beti.bet=" + betId).list();
58     }
59 }

```


### B.3.5 CategoryDao



Listing 17 – Implémentation du DAO

```
1  *package com.boc.botv.dao.impl;  
2  
3  import com.boc.botv.common.Database;  
4  import com.boc.botv.dao.CategoryDao;  
5  import com.boc.botv.model.Category;  
6  import java.util.List;  
7  import org.hibernate.Query;  
8  import org.springframework.stereotype.Repository;  
9  
10 @Repository("categoryDao")  
11 public class CategoryDaoImpl implements CategoryDao {  
12  
13     public List<Category> getChildsOf(int id) {  
14         return (List<Category>) Database.getSession().createQuery("from " +  
15             "Category as c where c.parentId=" + id).list();  
16     }  
17  
18     public void saveCategory(Category c) {  
19         Database.getSession().save(c);  
20     }  
21  
22     public Category getCategory(Integer id) {  
23         return (Category) Database.getSession().load(Category.class, id);  
24     }  
25  
26     public Category getCategory(String name) {  
27         Query q = Database.getSession().createQuery("from Category as c " +  
28             "where c.name=:name");  
29         q.setString("name", name);  
30         List<Category> list = q.list();  
31         if (list.isEmpty()) {  
32             return null;  
33         }  
34         return list.get(0);  
35     }  
36  
37     public List<Category> getCategory() {  
38         return (List<Category>) Database.getSession().  
39             createQuery("from Category").list();  
40     }  
41 }
```

### B.3.6 PaymentMethodDao

<<interface>>
 <b>PaymentMethodDao</b>
Attributes
Operations
public PaymentMethod[0..*] getPaymentMethods( )
public void savePaymentMethod( PaymentMethod paymentMethod )
public void deletePaymentMethod( PaymentMethod paymentMethod )
public void updatePaymentMethod( PaymentMethod paymentMethod )
public PaymentMethod getPaymentMethod( int paymentId )
public void deletePaymentMethod( int id )

Listing 18 – Implémentation du DAO

```

1  *package com.boc.botv.dao.impl;
2
3  import com.boc.botv.common.Database;
4  import com.boc.botv.dao.PaymentMethodDao;
5  import com.boc.botv.model.PaymentMethod;
6  import java.util.List;
7  import org.springframework.stereotype.Repository;
8
9  @Repository("paymentMethodDao")
10 public class PaymentMethodDaoImpl implements PaymentMethodDao {
11
12     public List<PaymentMethod> getPaymentMethods() {
13         return (List<PaymentMethod>) Database.getSession().createQuery("from " +
14             "PaymentMethod").list();
15     }
16
17     public void savePaymentMethod(PaymentMethod paymentMethod) {
18         Database.getSession().save(paymentMethod);
19     }
20
21     public void deletePaymentMethod(PaymentMethod paymentMethod) {
22         Database.getSession().delete(paymentMethod);
23     }
24
25     public void updatePaymentMethod(PaymentMethod paymentMethod) {
26         Database.getSession().update(paymentMethod);
27     }
28
29     public PaymentMethod getPaymentMethod(int paymentId) {
30         return (PaymentMethod) Database.getSession().load(PaymentMethod.class, paymentId);
31     }
32
33     public void deletePaymentMethod(int id) {
34         Database.getSession().createQuery("delete PaymentMethod as pm where pm.id="
35             + id).executeUpdate();
36     }
37 }

```

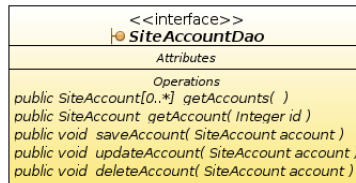
### B.3.7 SiteAccountDao

Listing 19 – Implémentation du DAO

```

1  *package com.boc.botv.dao.impl;
2
3  import com.boc.botv.common.Database;

```

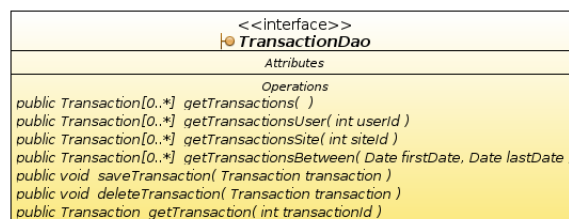


```

4 import com.boc.botv.dao.SiteAccountDao;
5 import com.boc.botv.model.SiteAccount;
6 import java.util.List;
7 import org.springframework.stereotype.Repository;
8
9 @Repository("siteAccountDao")
10 public class SiteAccountDaoImpl implements SiteAccountDao {
11
12     public List<SiteAccount> getAccounts() {
13         return (List<SiteAccount>) Database.getSession().createQuery("from " +
14             "SiteAccount").list();
15     }
16
17     public SiteAccount getAccount(Integer id) {
18         return (SiteAccount) Database.getSession().load(SiteAccount.class, id);
19     }
20
21     public void saveAccount(SiteAccount account) {
22         Database.getSession().save(account);
23     }
24
25     public void updateAccount(SiteAccount account) {
26         Database.getSession().update(account);
27     }
28
29     public void deleteAccount(SiteAccount account) {
30         Database.getSession().delete(account);
31     }
32 }

```

### B.3.8 TransactionDao



Listing 20 – Implémentation du DAO

```

1 *package com.boc.botv.dao.impl;
2
3 import com.boc.botv.common.Database;

```

```

4 import com.boc.botv.dao.TransactionDao;
5 import com.boc.botv.model.Transaction;
6 import java.util.Date;
7 import java.util.List;
8 import org.springframework.stereotype.Repository;
9 import java.sql.Timestamp;
10
11
12 @Repository("transactionDao")
13 public class TransactionDaoImpl implements TransactionDao {
14
15     public List<Transaction> getTransactions() {
16         return (List<Transaction>) Database.getSession().createQuery("from " +
17             "Transaction").list();
18     }
19
20     public List<Transaction> getTransactionsUser(int userId) {
21         return (List<Transaction>) Database.getSession().createQuery("from " +
22             "Transaction as t where t.user=" + userId).list();
23     }
24
25     public List<Transaction> getTransactionsSite(int siteId) {
26         return (List<Transaction>) Database.getSession().createQuery("from " +
27             "Transaction as t where t.siteAccount = " + siteId).list();
28     }
29
30     public List<Transaction> getTransactionsBetween(
31         Date firstDate, Date lastDate) {
32         Timestamp t1 = new Timestamp(firstDate.getTime());
33         Timestamp t2 = new Timestamp(lastDate.getTime());
34         return (List<Transaction>) Database.getSession().createQuery("from " +
35             "Transaction as t where t.dateOfOperation between '" + t1 + "' and '" + t2 + "'").list();
36     }
37
38     public void saveTransaction(Transaction transaction) {
39         Database.getSession().save(transaction);
40     }
41
42     public void deleteTransaction(Transaction transaction) {
43         Database.getSession().delete(transaction);
44     }
45
46     public Transaction getTransaction(int transactionId) {
47         return (Transaction) Database.getSession().load(
48             Transaction.class, transactionId);
49     }
50 }

```

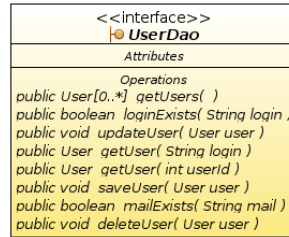
### B.3.9 UserDao

Listing 21 – Implémentation du DAO

```

1 *package com.boc.botv.dao.impl;
2
3 import com.boc.botv.common.Database;
4 import com.boc.botv.dao.UserDao;

```



```

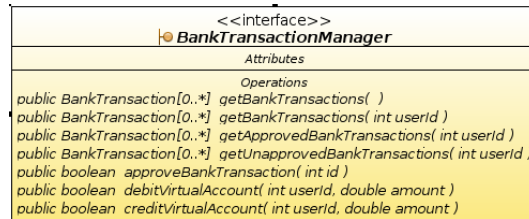
5  import com.boc.btv.model.User;
6  import java.util.List;
7  import org.springframework.stereotype.Repository;
8
9  @Repository("userDao")
10 public class UserDaoImpl implements UserDao {
11
12     public List<User> getUsers() {
13         return (List<User>) Database.getSession().createQuery("from User").list();
14     }
15
16     public boolean loginExists(String login) {
17         return !Database.getSession().createQuery("from User as user where " +
18             "user.login = '" + login + "'").list().isEmpty();
19     }
20
21     public void updateUser(User user) {
22         Database.getSession().update(user);
23     }
24
25     public User getUser(String login) {
26         List<User> lu = (List<User>) Database.getSession().createQuery("from " +
27             "User as user where user.login = '" + login + "'").list();
28         if (lu.size() != 1) {
29             return null;
30         }
31         User u = lu.get(0);
32         return u;
33     }
34
35     public User getUser(int userId) {
36         return (User) Database.getSession().load(User.class, userId);
37     }
38
39     public void saveUser(User user) {
40         Database.getSession().save(user);
41     }
42
43     public boolean mailExists(String mail) {
44         return !Database.getSession().createQuery("from User as user where " +
45             "user.email = '" + mail + "'").list().isEmpty();
46     }
47
48     public void deleteUser(User user) {
49         Database.getSession().delete(user);
50     }

```

51 }

## B.4 Services

### B.4.1 BankTransactionManager



Listing 22 – Implémentation du Service

```
1  *package com.boc.botv.service.impl;
2
3  import com.boc.botv.common.Database;
4  import com.boc.botv.dao.BankTransactionDao;
5  import com.boc.botv.dao.UserDao;
6  import com.boc.botv.model.BankTransaction;
7  import com.boc.botv.model.PaymentMethod;
8  import com.boc.botv.model.User;
9  import com.boc.botv.service.BankTransactionManager;
10 import java.util.Date;
11 import java.util.List;
12 import org.apache.commons.logging.Log;
13 import org.apache.commons.logging.LogFactory;
14 import org.hibernate.Session;
15 import org.springframework.beans.factory.annotation.Autowired;
16 import org.springframework.stereotype.Service;
17
18 @Service("bankTransactionManager")
19 public class BankTransactionManagerImpl implements BankTransactionManager {
20
21     private final Log logger = LogFactory.getLog(BankTransactionManagerImpl.class);
22     private BankTransactionDao bankTransactionDao;
23     private UserDao userDao;
24
25     @Autowired
26     public void setDao(UserDao userDao, BankTransactionDao bankTransactionDao) {
27         this.userDao = userDao;
28         this.bankTransactionDao = bankTransactionDao;
29     }
30
31     public List<BankTransaction> getBankTransactions() {
32         logger.info("[BankTransactionManager] Getting bank transactions");
33         return bankTransactionDao.getBankTransactions();
34     }
35
36     public List<BankTransaction> getBankTransactions(int userId) {
37         logger.info("[BankTransactionManager] Getting bank transaction #"
38             + userId);
39         return bankTransactionDao.getBankTransactions(userId);

```

```

40     }
41
42     public List<BankTransaction> getApprovedBankTransactions(int userId) {
43         logger.info("[BankTransactionManager] Getting approved bank transactions #"
44             + userId);
45         return bankTransactionDao.getApprovedBankTransactions(userId);
46     }
47
48     public List<BankTransaction> getUnapprovedBankTransactions(int userId) {
49         logger.info("[BankTransactionManager] Getting unapproved bank transactions #"
50             + userId);
51         return bankTransactionDao.getUnapprovedBankTransactions(userId);
52     }
53
54     public boolean approveBankTransaction(int id) {
55         if (id < 0) {
56             return false;
57         }
58         logger.info("[BankTransactionManager] Approving bank transactions #"
59             + id);
60
61         BankTransaction banktransaction = bankTransactionDao.getBankTransaction(id);
62         if (banktransaction == null) {
63             return false;
64         }
65         banktransaction.setIsApproved(true);
66         Date now = new Date();
67         banktransaction.setTransactionTime(now);
68
69         return true;
70     }
71
72     public boolean debitVirtualAccount(int userId, double amount) {
73         if (userId < 1 || amount <= 0) {
74             return false;
75         }
76         logger.info("[BankTransactionManager] Debiting " + amount
77             + " euros bank account #" + userId);
78
79         User user = this.userDao.getUser(userId);
80         if (user == null || user.getPaymentMethods().isEmpty()) {
81
82             return false;
83         }
84         Date now = new Date();
85         PaymentMethod pm = ( (List<PaymentMethod>) user.getPaymentMethods() )
86             .get(0);
87         BankTransaction banktransaction = new BankTransaction(pm, amount,
88             0, now, now, false);
89         this.bankTransactionDao.saveBankTransaction(banktransaction);
90
91         return true;
92     }
93
94     public boolean creditVirtualAccount(int userId, double amount) {
95         if (userId < 1 || amount <= 0) {

```

```

96         return false;
97     }
98     logger.info("[BankTransactionManager] Crediting " + amount
99         + " euros bank account #" + userId);
100
101     User user = this.userDao.getUser(userId);
102     if (user == null || user.getPaymentMethods().isEmpty()) {
103
104         return false;
105     }
106     Date now = new Date();
107     PaymentMethod pm = ( (List<PaymentMethod>) user.getPaymentMethods() )
108         .get(0);
109     BankTransaction banktransaction = new BankTransaction(pm, amount,
110         0, now, now, false);
111     this.bankTransactionDao.saveBankTransaction(banktransaction);
112
113     return true;
114 }
115 }

```

#### B.4.2 BetManager

<<interface>>	
BetManager	
Attributes	
Operations	
public Bet getBet( int betid )	
public Bet[0..*] getBets( )	
public Bet[0..*] getBetsByCreationTime( Date firstDate, Date lastDate )	
public Bet[0..*] getBetsByStartTime( Date firstDate, Date lastDate )	
public Bet[0..*] getBetsByEndTime( Date firstDate, Date lastDate )	
public Bet[0..*] getBetsPlayedBetween( Date firstDate, Date lastDate )	
public Bet[0..*] getBetsNotPlayedBetween( Date firstDate, Date lastDate )	
public BetChoice[0..*] getChoices( int betid )	
public BetInstance[0..*] getGamesByBet( int betid )	
public BetInstance[0..*] getGamesByChoice( int choiceid )	
public void createBet( Bet bet )	
public void updateBet( Bet bet )	
public void deleteBet( Bet bet )	
public boolean changeName( int betid, String name )	
public boolean changeDescription( int betid, String description )	
public boolean addChoice( Bet bet, String label )	
public boolean changeChoice( int choiceid, String label )	
public boolean deleteChoice( int choiceid )	
public boolean addGamer( int userid, int choiceid, double amount )	
public boolean setWinningChoice( int choiceid )	
public User[0..*] getWinners( int betid )	
public User[0..*] getLosers( int betid )	
public Bet[0..*] getBetBySubsection( int subsection )	
public Bet[0..*] getCurrentBetsBySubsection( int subsection )	
public Bet[0..*] getBetsByCloserEndDate( int numberOfBet, int subsection )	
public boolean saveOrUpdate( User admin, int id, String name, String desc,	

Listing 23 – Implémentation du Service

```

1  *package com.boc.botv.service.impl;
2
3  import com.boc.botv.common.BetInstanceStatus;
4  import com.boc.botv.dao.BetChoiceDao;
5  import com.boc.botv.dao.BetDao;
6  import com.boc.botv.dao.UserDao;
7  import com.boc.botv.dao.BetInstanceDao;
8  import com.boc.botv.dao.SiteAccountDao;
9  import com.boc.botv.dao.TransactionDao;
10 import com.boc.botv.model.Bet;

```

```

11 import com.boc.bottv.model.BetChoice;
12 import com.boc.bottv.model.BetInstance;
13 import com.boc.bottv.model.SiteAccount;
14 import com.boc.bottv.model.Transaction;
15 import com.boc.bottv.model.User;
16 import com.boc.bottv.model.UserRememberMe;
17 import com.boc.bottv.service.BetManager;
18 import java.util.ArrayList;
19 import java.util.Date;
20 import java.util.List;
21 import org.apache.commons.logging.Log;
22 import org.apache.commons.logging.LogFactory;
23 import org.springframework.beans.factory.annotation.Autowired;
24 import org.springframework.security.core.Authentication;
25 import org.springframework.security.core.context.SecurityContextHolder;
26 import org.springframework.stereotype.Service;
27
28 @Service("betManager")
29 public class BetManagerImpl implements BetManager {
30
31     private static final double taxForCountry = 0.075;
32     private static final double taxForSite = 0.1;
33     private final Log logger = LogFactory.getLog(BetManagerImpl.class);
34     private UserDao userDao;
35     private BetDao betDao;
36     private BetChoiceDao betChoiceDao;
37     private BetInstanceDao betInstanceDao;
38     private SiteAccountDao siteAccountDao;
39     private TransactionDao transactionDao;
40
41     @Autowired
42     public void setDao(UserDao userDao, BetDao betDao, BetInstanceDao betInstanceDao,
43         SiteAccountDao siteAccountDao, TransactionDao transactionDao,
44         BetChoiceDao betChoiceDao) {
45         this.userDao = userDao;
46         this.betDao = betDao;
47         this.betInstanceDao = betInstanceDao;
48         this.siteAccountDao = siteAccountDao;
49         this.transactionDao = transactionDao;
50         this.betChoiceDao = betChoiceDao;
51     }
52
53     public Bet getBet(int betId) {
54         logger.info("[BetManager] Getting bet by id #" + betId);
55         return betDao.getBet(new Integer(betId));
56     }
57
58     public List<Bet> getBets() {
59         logger.info("[BetManager] Getting bets");
60         return betDao.getBets();
61     }
62
63     public List<Bet> getBetsByCreationTime(Date firstDate, Date lastDate) {
64         logger.info("[BetManager] Getting bets by creation time between "
65             + firstDate + " and " + lastDate);
66         return betDao.getBetsByCreationTime(firstDate, lastDate);

```

```

67 }
68
69 public List<Bet> getBetsByStartTime(Date firstDate, Date lastDate) {
70     logger.info("[BetManager] Getting bets by start time between "
71         + firstDate + " and " + lastDate);
72     return betDao.getBetsByStartTime(firstDate, lastDate);
73 }
74
75 public List<Bet> getBetsByEndTime(Date firstDate, Date lastDate) {
76     logger.info("[BetManager] Getting bets by end time between "
77         + firstDate + " and " + lastDate);
78     return betDao.getBetsByEndTime(firstDate, lastDate);
79 }
80
81 public List<Bet> getBetsPlayedBetween(Date firstDate, Date lastDate) {
82     logger.info("[BetManager] Getting bets with instances between : "
83         + firstDate + " and " + lastDate);
84     return betDao.getBetsPlayedBetween(firstDate, lastDate);
85 }
86
87 public List<Bet> getBetsNotPlayedBetween(Date firstDate, Date lastDate) {
88     logger.info("[BetManager] Getting bets with instances between : "
89         + firstDate + " and " + lastDate);
90     return betDao.getBetsNotPlayedBetween(firstDate, lastDate);
91 }
92
93 public List<BetChoice> getChoices(int betId) {
94     logger.info("[BetManager] Getting choices for bet #" + betId);
95     return betChoiceDao.getChoices(betId);
96 }
97
98 public List<BetInstance> getGamesByBet(int betId) {
99     logger.info("[BetManager] Getting bet instances for bet #" + betId);
100    return betInstanceDao.getGamesByBet(betId);
101 }
102
103 public List<BetInstance> getGamesByChoice(int choiceId) {
104     logger.info("[BetManager] Getting bet instances for bet bind with choice #"
105         + choiceId);
106     return betInstanceDao.getGamesByChoice(choiceId);
107 }
108
109 public List<User> getWinners(int betId) {
110     logger.info("[BetManager] Getting winners for bet #" + betId);
111     BetChoice choice = betChoiceDao.getWinningChoice(betId);
112     if (choice != null) {
113         return this.getUserWhoChoosed(choice.getId());
114     }
115     return null;
116 }
117
118 public List<User> getLosers(int betId) {
119     logger.info("[BetManager] Getting losers for bet #" + betId);
120     List<BetChoice> choices = betChoiceDao.getLosingChoices(betId);
121     if (choices == null) {
122         return null;

```

```

123     }
124     List<User> users = new ArrayList<User>();
125     for (BetChoice bc : choices) {
126         List<User> ul = this.getUserWhoChooed(bc.getId());
127         if (ul != null) {
128             users.addAll(ul);
129         }
130     }
131     return users;
132 }
133
134 public List<User> getUserWhoChooed(int choiceId) {
135     logger.info("[BetManager] Getting users who choosed choice : " + choiceId);
136     List<BetInstance> bil = betInstanceDao.getGamesByChoice(choiceId);
137     List<User> users = new ArrayList<User>();
138     for (BetInstance bi : bil) {
139         users.add(bi.getUser());
140     }
141     return users;
142 }
143
144 public void createBet(Bet bet) {
145     logger.info("[BetManager] Creating bet #" + bet.getId());
146     betDao.saveBet(bet);
147 }
148
149 public void updateBet(Bet bet) {
150     logger.info("[BetManager] Updating bet #" + bet.getId());
151     betDao.updateBet(bet);
152 }
153
154 public void deleteBet(Bet bet) {
155     logger.info("[BetManager] Deleting bet #" + bet.getId());
156     betDao.deleteBet(bet);
157 }
158
159 public boolean changeName(int betId, String name) {
160     if (betId < 1) {
161         return false;
162     }
163     logger.info("[BetManager] Changing name " + name + " for bet " + betId);
164
165     Bet bet = betDao.getBet(betId);
166     bet.setName(name);
167     return true;
168 }
169
170 public boolean changeDescription(int betId, String description) {
171     if (betId < 1) {
172         return false;
173     }
174     logger.info("[BetManager] Changing description " + description + " for bet "
175         + betId);
176
177     Bet bet = betDao.getBet(betId);
178     bet.setDescription(description);

```

```

179     return true;
180 }
181
182 public boolean addChoice(Bet bet, String name) {
183     if (bet == null || name == null || name.length() < 1) {
184         return false;
185     }
186     logger.info("[BetManager] Adding choice " + name + "' for bet "
187         + bet.getId());
188
189     BetChoice choice = new BetChoice(bet, name, false);
190     betChoiceDao.saveChoice(choice);
191     return true;
192 }
193
194 public boolean changeChoice(int choiceId, String label) {
195     if (choiceId < 1 || label == null || label.length() < 1) {
196         return false;
197     }
198     logger.info("[BetManager] Changing choice " + choiceId + "'");
199
200     BetChoice choice = betChoiceDao.getChoice(choiceId);
201     choice.setLabel(label);
202     return true;
203 }
204
205 public boolean deleteChoice(int choiceId) {
206     if (choiceId < 1) {
207         return false;
208     }
209     logger.info("[BetManager] Deleting choice " + choiceId + "'");
210
211     BetChoice bc = betChoiceDao.getChoice(choiceId);
212     if (bc != null) {
213         return false;
214     }
215     betChoiceDao.deleteChoice(bc);
216     return true;
217 }
218
219 public boolean addGamer(int userId, int choiceId, double amount) {
220     if (userId < 1 || choiceId < 1 || amount < 1) {
221         return false;
222     }
223     logger.info("[BetManager] Adding choice " + choiceId + " with an amount of "
224         + amount + " for user " + userId);
225
226     User user = userDao.getUser(userId);
227     if (user == null || user.getBalance() < amount) {
228         return false;
229     }
230     BetChoice bc = betChoiceDao.getChoice(choiceId);
231     if (bc == null) {
232         return false;
233     }
234     Date now = new Date();

```

```

235     Bet b = bc.getBet();
236     if (now.before(b.getStartTime()) || now.after(b.getEndTime())) {
237         return false;
238     }
239
240     String label = String.format("PRÉLÈVEMENT PARI\n%1$s | CHOIX : '%2$s'",
241         b.getName(), bc.getLabel());
242     SiteAccount sa = siteAccountDao.getAccount(1);
243
244     logger.info("[BetManager] Crediting (output) client account");
245     user.decreaseBalance(amount);
246     Authentication a = SecurityContextHolder.getContext().getAuthentication();
247     ((UserRememberMe) a.getPrincipal()).setBalance(user.getBalance());
248
249     logger.info("[BetManager] Establishing transaction");
250     Transaction t = new Transaction(user, sa, label, now, amount, 0);
251     transactionDao.saveTransaction(t);
252
253     logger.info("[BetManager] Creating the bet instance");
254     BetInstance bi = new BetInstance(bc, user, b, amount, now,
255         BetInstanceStatus.playing);
256     betInstanceDao.saveGame(bi);
257     return true;
258 }
259
260 public boolean setWinningChoice(int choiceId) {
261     if (choiceId < 1) {
262         return false;
263     }
264     logger.info("[BetManager] Setting bet choice #" + choiceId
265         + " to the 'win' status.");
266
267     BetChoice winningChoice = betChoiceDao.getChoice(choiceId);
268     if (winningChoice == null) {
269         return false;
270     }
271     winningChoice.setIsWinning(true);
272
273     logger.info("[BetManager] Updating bet parameters...");
274     logger.info("[BetManager] Calculating amounts.");
275     double totalAmount = 0;
276     double totalWinnersAmount = 0;
277     int betId = winningChoice.getBet().getId();
278     List<BetInstance> games = this.getGamesByBet(betId);
279     List<BetInstance> winningGames = new ArrayList<BetInstance>();
280     for (BetInstance g : games) {
281         totalAmount += g.getBetAmount();
282         if (g.getBetChoice().getId() == choiceId) {
283             totalWinnersAmount += g.getBetAmount();
284             winningGames.add(g);
285         }
286     }
287
288     // aucun parieur n'a gagné
289     if (totalWinnersAmount == 0) {
290         // montant taxe Etat déduite

```

```

291     double stateTax = 0;
292     double amountForSite = 0;
293     SiteAccount sa1 = siteAccountDao.getAccount(1); // Compte Paris
294     SiteAccount sa2 = siteAccountDao.getAccount(2); // Compte Taxe Site
295     SiteAccount sa3 = siteAccountDao.getAccount(3); // Compte Taxe État
296     User admin = userDao.getUser("admin");
297     String betName = winningChoice.getBet().getName();
298     String choiceName = winningChoice.getLabel();
299     String label1 = String.format(
300         "GAIN DU SITE POUR LE PARI\n'%1$s' | CHOIX '%2$s'",
301         betName, choiceName);
302     String label2 = "PRÉLÈVEMENT TAXE POUR LE SITE";
303     String label3 = "PRÉLÈVEMENT TAXE POUR L'ÉTAT";
304     Date now = new Date();
305
306     // on crédite le site de l'admin
307     Transaction t = new Transaction(admin, sa1, label1, now, 0, totalAmount);
308     transactionDao.saveTransaction(t);
309     admin.increaseBalance(amountForSite);
310
311     // on prélève les taxes pour le site
312     amountForSite = totalAmount * (1 - taxForCountry);
313     t = new Transaction(admin, sa2, label2, now, amountForSite, 0);
314     transactionDao.saveTransaction(t);
315     admin.decreaseBalance(amountForSite);
316
317     // on prélève les taxes pour l'état
318     stateTax = totalAmount * taxForCountry;
319     t = new Transaction(admin, sa3, label3, now, stateTax, 0);
320     transactionDao.saveTransaction(t);
321     admin.decreaseBalance(stateTax);
322     return true;
323 }
324
325 double totalRemainder = 0;
326 boolean doTaxSite = true;
327 boolean doTaxCountry = true;
328 if ((totalRemainder = totalAmount * (1 - (taxForSite + taxForCountry)))
329     < totalWinnersAmount) {
330     if ((totalRemainder = totalAmount * (1 - (taxForCountry)))
331         < totalWinnersAmount) {
332         doTaxCountry = false;
333         if ((totalRemainder = totalAmount * (1 - (taxForSite)))
334             < totalWinnersAmount) {
335             doTaxSite = false;
336             totalRemainder = totalAmount;
337         }
338     }
339 }
340
341 logger.info("[BetManager] Calculating and applying winnings");
342 double baseratio = totalAmount / totalWinnersAmount;
343 double ratio = totalRemainder / totalWinnersAmount;
344 double amount = 0;
345 Date now = new Date();
346 String betName = winningChoice.getBet().getName();

```

```

347 String choiceName = winningChoice.getLabel();
348 String label1 = String.format("GAIN PARI\n'%1$s' | CHOIX '%2$s'",
349     betName, choiceName);
350 String label2 = "PRÉLÈVEMENT TAXE POUR LE SITE";
351 String label3 = "PRÉLÈVEMENT TAXE POUR L'ÉTAT";
352 SiteAccount sa1 = siteAccountDao.getAccount(1); // Compte Paris
353 SiteAccount sa2 = siteAccountDao.getAccount(2); // Compte Taxe Site
354 SiteAccount sa3 = siteAccountDao.getAccount(3); // Compte Taxe État
355 for (BetInstance w : winningGames) {
356     User u = w.getUser();
357     // Du compte des paris vers le compte utilisateur : tous les gains non taxés.
358     amount = baseratio * w.getBetAmount();
359     Transaction t = new Transaction(w.getUser(), sa1, label1, now, 0, amount);
360     transactionDao.saveTransaction(t);
361     u.increaseBalance(amount);
362
363     // Prélèvement des taxes.
364     if (doTaxSite) {
365         amount = w.getBetAmount() * taxForSite;
366         t = new Transaction(w.getUser(), sa1, label2, now, amount, 0);
367         transactionDao.saveTransaction(t);
368         u.decreaseBalance(amount);
369     }
370
371     if (doTaxCountry) {
372         amount = w.getBetAmount() * taxForCountry;
373         t = new Transaction(w.getUser(), sa1, label3, now, amount, 0);
374         transactionDao.saveTransaction(t);
375         u.decreaseBalance(amount);
376     }
377 }
378
379 return true;
380 }
381
382 public List<Bet> getBetBySubsection(int subsection) {
383     logger.info("[BetManager] Getting bets by subsection");
384     return betDao.getBetsBySubsection(subsection);
385 }
386
387 public List<Bet> getBetsByCloserEndDate(int numberOfBet, int subsection) {
388     logger.info("[BetManager] Getting bets by subsection and closest end time");
389     return betDao.getBetsByCloserEndDate(numberOfBet, subsection);
390 }
391
392 public boolean saveOrUpdate(User admin, int id, String name, String desc,
393     Date creationDate, Date startDate, Date endDate, int section, boolean actif)
394     // boolean changed = false;
395     // création d'un nouveau pari
396     if (id == -1) {
397         Bet bet = new Bet(admin, name, desc, creationDate, startDate, endDate,
398             section, actif);
399         betDao.saveBet(bet);
400     } else {
401         Bet bet = betDao.getBet(id);
402         if (!bet.getName().equals(name)) {

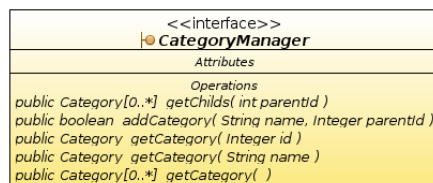
```

```

403         bet.setName(name);
404     }
405     if (!bet.getDescription().equals(desc)) {
406         bet.setDescription(desc);
407     }
408     if (!bet.getStartTime().equals(startDate)) {
409         bet.setStartTime(startDate);
410     }
411     if (!bet.getEndTime().equals(endDate)) {
412         bet.setEndTime(endDate);
413     }
414     if (!(bet.getSubsection() == section)) {
415         bet.setSubsection(section);
416     }
417     if (!(bet.isStatus() == actif)) {
418         bet.setStatus(actif);
419     }
420
421     betDao.updateBet(bet);
422 }
423 return true;
424 }
425
426 public List<Bet> getCurrentBetsBySubsection(int subsection) {
427     logger.info("[BetManager] Getting playable bets by subsection");
428     return betDao.getCurrentBetsBySubsection(subsection);
429 }
430 }

```

### B.4.3 CategoryManager



Listing 24 – Implémentation du Service

```

1  *package com.boc.btv.service.impl;
2
3  import com.boc.btv.common.Database;
4  import com.boc.btv.dao.CategoryDao;
5  import com.boc.btv.model.Category;
6  import com.boc.btv.service.CategoryManager;
7  import java.util.List;
8  import org.apache.commons.logging.Log;
9  import org.apache.commons.logging.LogFactory;
10 import org.hibernate.Session;
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.stereotype.Service;
13 import org.springframework.transaction.annotation.Transactional;
14

```

```

15 @Service("categoryManager")
16 public class CategoryManagerImpl implements CategoryManager {
17
18     private final Log logger = LoggerFactory.getLog(SiteAccountManagerImpl.class);
19     private CategoryDao categoryDao;
20
21     @Autowired
22     public void setCategoryDao(CategoryDao categoryDao) {
23         this.categoryDao = categoryDao;
24     }
25
26     public List<Category> getChilds(int parentId) {
27         logger.info("[CategoryManager] Getting childs of #" + parentId);
28         List<Category> cl = categoryDao.getChildsOf(parentId);
29
30         return cl;
31     }
32
33     public Category getCategory(Integer id) {
34         logger.info("[CategoryManager] Looking for category #" + id);
35         Category c = categoryDao.getCategory(id);
36         List<Category> sub = categoryDao.getChildsOf(c.getId());
37         if (sub != null) {
38             c.setCountOfChilds(sub.size());
39         }
40
41         return c;
42     }
43
44     public Category getCategory(String name) {
45         logger.info("[CategoryManager] Looking for category '" + name + "'");
46         Category c = categoryDao.getCategory(name);
47         List<Category> sub = categoryDao.getChildsOf(c.getId());
48         if (sub != null) {
49             c.setCountOfChilds(sub.size());
50         }
51
52         return c;
53     }
54
55     public List<Category> getCategory(){
56         logger.info("[CategoryManager] getting all the Category");
57         List<Category> cl = categoryDao.getCategory();
58
59         return cl;
60     }
61
62     public boolean addCategory(String name, Integer parentId) {
63         logger.info("[CategoryManager] Adding category '" + name
64             + "' as a child of #" + parentId);
65         if (parentId == null || parentId < 0 || name == null || name.length() < 2) {
66             return false;
67         }
68         Category c = new Category(name, parentId);
69         categoryDao.saveCategory(c);
70

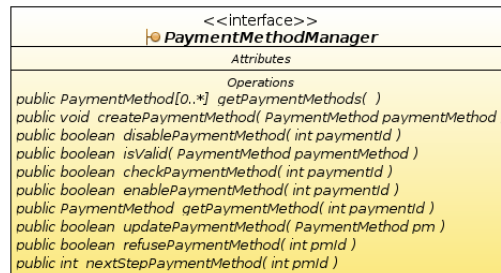
```

```

71     return true;
72 }
73 }

```

#### B.4.4 PaymentMethodManager



Listing 25 – Implémentation du Service

```

1  *package com.boc.botv.service.impl;
2
3  import com.boc.botv.common.PaymentMethodStatus;
4  import com.boc.botv.dao.PaymentMethodDao;
5  import com.boc.botv.model.PaymentMethod;
6  import com.boc.botv.service.PaymentMethodManager;
7  import java.util.List;
8  import org.apache.commons.logging.Log;
9  import org.apache.commons.logging.LogFactory;
10 import org.springframework.beans.factory.annotation.Autowired;
11 import org.springframework.stereotype.Service;
12
13 @Service("paymentMethodManager")
14 public class PaymentMethodManagerImpl implements PaymentMethodManager {
15
16     private final Log logger = LogFactory.getLog(PaymentMethodManagerImpl.class);
17     private PaymentMethodDao paymentMethodDao;
18
19     @Autowired
20     public void setPaymentMethodDao(PaymentMethodDao paymentMethodDao) {
21         this.paymentMethodDao = paymentMethodDao;
22     }
23
24     public List<PaymentMethod> getPaymentMethods() {
25         List<PaymentMethod> paymentMethods = paymentMethodDao.getPaymentMethods();
26         logger.info("[PaymentMethod] Length of the list : " + paymentMethods.size());
27
28         return paymentMethods;
29     }
30
31     public boolean isValid(PaymentMethod paymentMethod) {
32         logger.info("[PaymentMethod] Check status of payment method #" +
33             + paymentMethod + " ");
34         int status = paymentMethod.getStatus();
35
36         return status == 2;

```

```

37 }
38
39 public PaymentMethod getPaymentMethod(int paymentId) {
40     logger.info("[PaymentMethod] Getting Status");
41     return paymentMethodDao.getPaymentMethod(paymentId);
42 }
43
44 public void createPaymentMethod(PaymentMethod paymentMethod) {
45     logger.info("[PaymentMethod] Creation of the payment method : "
46         + paymentMethod.getId());
47     paymentMethodDao.savePaymentMethod(paymentMethod);
48 }
49
50 public boolean disablePaymentMethod(int paymentId) {
51     logger.info("[PaymentMethod] Disable payment method of #" + paymentId);
52     PaymentMethod pm = paymentMethodDao.getPaymentMethod(paymentId);
53     pm.setStatus(PaymentMethodStatus.disabled);
54
55     return true;
56 }
57
58 public boolean enablePaymentMethod(int paymentId) {
59     logger.info("[PaymentMethod] Enable payment method of #" + paymentId);
60     PaymentMethod pm = paymentMethodDao.getPaymentMethod(paymentId);
61     pm.setStatus(PaymentMethodStatus.enabled);
62
63     return true;
64 }
65
66 public boolean updatePaymentMethod(PaymentMethod pm) {
67     logger.info("[PaymentMethod] Updating PaymentMethod");
68     if (pm == null) {
69         return false;
70     }
71     paymentMethodDao.updatePaymentMethod(pm);
72
73     return true;
74 }
75
76 public int nextStepPaymentMethod(int pmId) {
77     logger.info("[PaymentMethod] Updating payment method status #" + pmId);
78     if (pmId < 0) {
79         return -1;
80     }
81     PaymentMethod pm = paymentMethodDao.getPaymentMethod(pmId);
82     if (pm == null) {
83
84         return -1;
85     }
86     int status = pm.getStatus();
87     if (status == PaymentMethodStatus.disabled) {
88         status = PaymentMethodStatus.checking;
89     } else {
90         status = PaymentMethodStatus.enabled;
91     }
92     pm.setStatus(status);

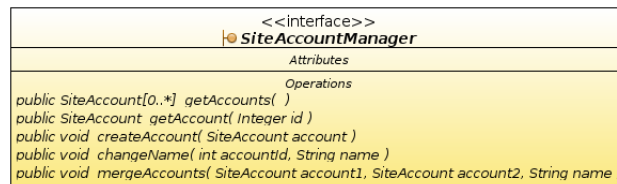
```

```

93
94     return status;
95 }
96
97 public boolean checkPaymentMethod(int paymentId) {
98     logger.info("[PaymentMethod] Starting verification for payment method of #"
99         + paymentId);
100     PaymentMethod pm = paymentMethodDao.getPaymentMethod(paymentId);
101     pm.setStatus(PaymentMethodStatus.checking);
102
103     return true;
104 }
105
106 public boolean refusePaymentMethod(int paymentId) {
107     logger.info("[PaymentMethod] Refusing payment method #" + paymentId);
108     PaymentMethod pm = paymentMethodDao.getPaymentMethod(paymentId);
109     pm.setStatus(PaymentMethodStatus.refused);
110
111     return true;
112 }
113 }

```

#### B.4.5 SiteAccountManager



Listing 26 – Implémentation du Service

```

1  *package com.boc.botv.service.impl;
2
3  import com.boc.botv.common.Database;
4  import com.boc.botv.dao.SiteAccountDao;
5  import com.boc.botv.model.SiteAccount;
6  import com.boc.botv.service.SiteAccountManager;
7  import java.util.Date;
8  import java.util.List;
9  import org.apache.commons.logging.Log;
10 import org.apache.commons.logging.LogFactory;
11 import org.hibernate.Session;
12 import org.springframework.beans.factory.annotation.Autowired;
13 import org.springframework.stereotype.Service;
14 import org.springframework.transaction.annotation.Transactional;
15
16 @Service("siteAccountManager")
17 public class SiteAccountManagerImpl implements SiteAccountManager {
18
19     private final Log logger = LogFactory.getLog(SiteAccountManagerImpl.class);
20     private SiteAccountDao siteAccountDao;
21

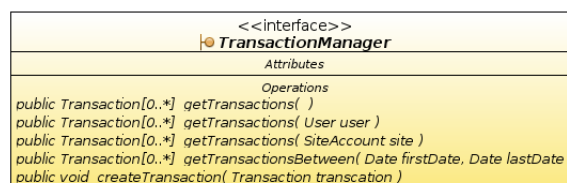
```

```

22  @Autowired
23  public void setSiteAccountDao(SiteAccountDao siteAccountDao) {
24      this.siteAccountDao = siteAccountDao;
25  }
26
27  public List<SiteAccount> getAccounts() {
28      logger.info("[SiteAccountManager] Getting all SiteAccount");
29      return siteAccountDao.getAccounts();
30  }
31
32  public SiteAccount getAccount(Integer id) {
33      logger.info("[SiteAccountManager] Getting SiteAccount #" + id);
34      return siteAccountDao.getAccount(id);
35  }
36
37  public void createAccount(SiteAccount account) {
38      logger.info("[SiteAccountManager] Creating SiteAccount with account");
39      siteAccountDao.saveAccount(account);
40  }
41  }
42
43  public void changeName(int accountId, String name) {
44      logger.info("[SiteAccountManager] Changing siteAccount name to " + name + " ");
45      SiteAccount account = siteAccountDao.getAccount(accountId);
46      account.setName(name);
47  }
48
49  public void mergeAccounts(SiteAccount account1, SiteAccount account2,
50      String name) {
51      logger.info(String.format("[SiteAccountManager] Merging accounts #%1$d " +
52          "and %2$d under name '%3$s'", account1.getId(), account2.getId(), name));
53      SiteAccount sa = new SiteAccount();
54      sa.setName(name);
55      sa.setBalance(account1.getBalance() + account2.getBalance());
56      sa.setCreationTime(new Date());
57      siteAccountDao.deleteAccount(account1);
58      siteAccountDao.deleteAccount(account2);
59      siteAccountDao.saveAccount(sa);
60  }
61  }

```

#### B.4.6 TransactionManager



Listing 27 – Implémentation du Service

```

1  *package com.boc.botv.service.impl;
2

```

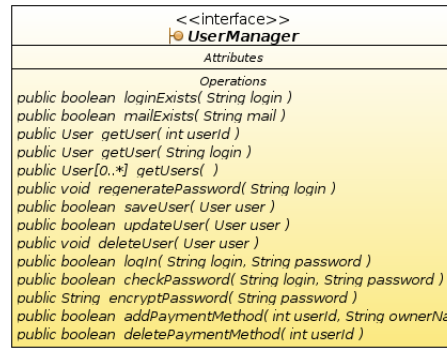
```

3  import com.boc.botv.dao.TransactionDao;
4  import com.boc.botv.model.SiteAccount;
5  import com.boc.botv.model.Transaction;
6  import com.boc.botv.model.User;
7  import com.boc.botv.service.TransactionManager;
8  import java.util.Date;
9  import java.util.List;
10 import org.apache.commons.logging.Log;
11 import org.apache.commons.logging.LogFactory;
12 import org.springframework.beans.factory.annotation.Autowired;
13 import org.springframework.stereotype.Service;
14
15 @Service("transactionManager")
16 public class TransactionManagerImpl implements TransactionManager {
17
18     private final Log logger = LogFactory.getLog(TransactionManagerImpl.class);
19     private TransactionDao transactionDao;
20
21     @Autowired
22     public void setTransactionDao(TransactionDao transactionDao) {
23         this.transactionDao = transactionDao;
24     }
25
26     public List<Transaction> getTransactions() {
27         logger.info("[TransactionManager] Getting transactions");
28         return transactionDao.getTransactions();
29     }
30
31     public List<Transaction> getTransactions(User user) {
32         logger.info("[TransactionManager] Getting transactions for user "
33             + user.getLogin() + " ");
34         return transactionDao.getTransactionsUser(user.getId());
35     }
36
37     public List<Transaction> getTransactions(SiteAccount siteAccount) {
38         logger.info("[TransactionManager] Getting transactions for site account #"
39             + siteAccount.getId());
40         return transactionDao.getTransactionsSite(siteAccount.getId());
41     }
42
43     public List<Transaction> getTransactionsBetween(Date firstDate, Date lastDate) {
44         logger.info("[TransactionManager] Getting transactions between " + firstDate
45             + " and " + lastDate);
46         return transactionDao.getTransactionsBetween(firstDate, lastDate);
47     }
48
49     public void createTransaction(Transaction transaction) {
50         logger.info("[TransactionManager] Creating transactions");
51         transactionDao.saveTransaction(transaction);
52     }
53 }

```

#### B.4.7 UserManager

Listing 28 – Implémentation du Service



```

1  *package com.boc.btv.service.impl;
2
3  import com.boc.btv.common.PaymentMethodStatus;
4  import com.boc.btv.dao.BetChoiceDao;
5  import com.boc.btv.dao.BetInstanceDao;
6  import com.boc.btv.dao.PaymentMethodDao;
7  import com.boc.btv.dao.SiteAccountDao;
8  import com.boc.btv.dao.TransactionDao;
9  import com.boc.btv.dao.UserDao;
10 import com.boc.btv.model.BetChoice;
11 import com.boc.btv.model.BetInstance;
12 import com.boc.btv.model.PaymentMethod;
13 import com.boc.btv.model.SiteAccount;
14 import com.boc.btv.model.Transaction;
15 import com.boc.btv.model.User;
16 import com.boc.btv.model.UserRememberMe;
17 import com.boc.btv.service.UserManager;
18 import java.util.ArrayList;
19 import java.util.Date;
20 import java.util.List;
21 import org.apache.commons.codec.digest.DigestUtils;
22 import org.apache.commons.logging.Log;
23 import org.apache.commons.logging.LogFactory;
24 import org.springframework.beans.factory.annotation.Autowired;
25 import org.springframework.security.authentication.RememberMeAuthenticationToken;
26 import org.springframework.security.core.Authentication;
27 import org.springframework.security.core.GrantedAuthority;
28 import org.springframework.security.core.authority.GrantedAuthorityImpl;
29 import org.springframework.security.core.context.SecurityContextHolder;
30 import org.springframework.stereotype.Service;
31
32 @Service("userManager")
33 public class UserManagerImpl implements UserManager {
34
35     private final Log logger = LogFactory.getLog(UserManagerImpl.class);
36     private UserDao userDao;
37     private BetChoiceDao betChoiceDao;
38     private BetInstanceDao betInstanceDao;
39     private PaymentMethodDao paymentMethodDao;
40     private SiteAccountDao siteAccountDao;
41     private TransactionDao transactionDao;
42
43     @Autowired

```

```

44 public void setDao(UserDao userDao, BetChoiceDao betChoiceDao,
45     BetInstanceDao betInstanceDao, PaymentMethodDao paymentMethodDao,
46     SiteAccountDao siteAccountDao, TransactionDao transactionDao) {
47     this.userDao = userDao;
48     this.betChoiceDao = betChoiceDao;
49     this.betInstanceDao = betInstanceDao;
50     this.paymentMethodDao = paymentMethodDao;
51     this.siteAccountDao = siteAccountDao;
52     this.transactionDao = transactionDao;
53 }
54
55 public List<User> getUsers() {
56     logger.info("[UserManager] Getting all users");
57     return userDao.getUsers();
58 }
59
60 public boolean loginExists(String login) {
61     logger.info("[UserManager] Testing if login " + login + " exists");
62     return userDao.loginExists(login);
63 }
64
65 public boolean mailExists(String mail) {
66     logger.info("[UserManager] Testing if mail " + mail + " exists");
67     return userDao.mailExists(mail);
68 }
69
70 public User getUser(int userId) {
71     logger.info("[UserManager] Getting user #" + userId);
72     return userDao.getUser(userId);
73 }
74
75 public User getUser(String login) {
76     logger.info("[UserManager] Getting user " + login + "");
77     return userDao.getUser(login);
78 }
79
80 public void regeneratePassword(String login) {
81     logger.info("[UserManager] Regenerating password for " + login + "");
82     User user = userDao.getUser(login);
83     String pwd = generatePassword();
84     logger.info("[UserManager] Generated password " + pwd + "");
85     user.setPassword(encryptPassword(pwd));
86     // TODO envoi d'un mail
87     user.setLastModification(new Date());
88     userDao.updateUser(user);
89 }
90
91 public boolean saveUser(User user) {
92     logger.info("[UserManager] Saving user " + user.getLogin() + "");
93     if (!user.getPassword().equals(user.getCheckPassword())) {
94         return false;
95     }
96     user.setPassword(encryptPassword(user.getPassword()));
97     Date now = new Date();
98     user.setCreationTime(now);
99     user.setLastModification(now);

```

```

100     user.setLastConnection(now);
101     userDao.saveUser(user);
102
103     return true;
104 }
105
106 public boolean updateUser(User user) {
107     logger.info("[userManager] Updating user " + user.getLogin() + "");
108     user.setLastModification(new Date());
109     userDao.updateUser(user);
110
111     return true;
112 }
113
114 public void deleteUser(User user) {
115     logger.info("[userManager] Deleting user " + user.getLogin() + "");
116     userDao.deleteUser(user);
117 }
118
119 public boolean logIn(String login, String password) {
120     logger.info("[userManager] Try to log in " + login + "");
121     User user = userDao.getUser(login);
122     if (user == null || !encryptPassword(password).equals(user.getPassword())) {
123         return false;
124     }
125     List<GrantedAuthority> ga = new ArrayList<GrantedAuthority>();
126     ga.add(new GrantedAuthorityImpl("ROLE_USER"));
127     if (user.getId() == 1) {
128         ga.add(new GrantedAuthorityImpl("ROLE_ADMIN"));
129     }
130     Authentication result = new RememberMeAuthenticationToken(login,
131         new UserRememberMe(user), ga);
132     SecurityContextHolder.getContext().setAuthentication(result);
133     user.setLastConnection(new Date());
134
135     return true;
136 }
137
138 public List<User> getWinners(int betId) {
139     List<BetInstance> betInstances = this.getGamesByBetId(betId);
140     List<User> winners = new ArrayList<User>();
141     for (BetInstance betInstance : betInstances) {
142         if (betInstance.getStatus() == 1) {
143             User user = userDao.getUser(betInstance.getUser().getId());
144             winners.add(user);
145         }
146     }
147
148     return winners;
149 }
150
151 public List<User> getLosers(int betId) {
152     List<BetInstance> betInstances = this.getGamesByBetId(betId);
153     List<User> losers = new ArrayList<User>();
154     for (BetInstance betInstance : betInstances) {
155         if (betInstance.getStatus() == 0) {

```

```

156         User user = userDao.getUser(betInstance.getUser().getId());
157         losers.add(user);
158     }
159 }
160
161 return losers;
162 }
163
164 public List<User> getUserWhoChooosed(int choiceId) {
165     List<BetInstance> betInstances = betInstanceDao.getGamesByChoice(choiceId);
166     List<User> whoChooosed = new ArrayList<User>();
167     for (BetInstance betInstance : betInstances) {
168         User user = userDao.getUser(betInstance.getUser().getId());
169         whoChooosed.add(user);
170     }
171
172     return whoChooosed;
173 }
174
175 public boolean checkPassword(String login, String password) {
176     User user = userDao.getUser(login);
177     if (user == null) {
178         return false;
179     }
180
181     return user.getPassword().equals(encryptPassword(password));
182 }
183
184 public boolean addPaymentMethod(int userId, String ownerName, String cardNumber
185     String cardName, Date cardExpirationDate, String cardCryptogram) {
186     if (userId < 2) {
187         return false;
188     } else if (ownerName.trim().length() < 2) {
189         return false;
190     } else if (cardName.trim().length() < 2) {
191         return false;
192     } else if (cardNumber.trim().length() != 16) {
193         return false;
194     } else if (cardCryptogram.trim().length() != 3) {
195         return false;
196     }
197     User u = userDao.getUser(userId);
198
199     u.increaseBalance(100);
200     Authentication a = SecurityContextHolder.getContext().getAuthentication();
201     ((UserRememberMe) a.getPrincipal()).setBalance(u.getBalance());
202
203     SiteAccount sa = siteAccountDao.getAccount(4);
204     Transaction t = new Transaction(u, sa, "Bonus de création de compte bancaire"
205         new Date(), 0, 100);
206     transactionDao.saveTransaction(t);
207
208     PaymentMethod pm = new PaymentMethod(u.getId(), u, new Date(), ownerName,
209         cardNumber, cardExpirationDate, cardName, cardCryptogram,
210         PaymentMethodStatus.disabled);
211     paymentMethodDao.savePaymentMethod(pm);

```

```

212     return true;
213 }
214
215 public boolean deletePaymentMethod(int userId) {
216     if (userId < 2) {
217         return false;
218     }
219
220     paymentMethodDao.deletePaymentMethod(userId);
221     return true;
222 }
223
224 private List<BetInstance> getGamesByBetId(int betId) {
225     List<BetChoice> choices = betChoiceDao.getChoices(betId);
226     List<BetInstance> list = new ArrayList<BetInstance>();
227     if (choices != null) {
228         for (BetChoice c : choices) {
229             List<BetInstance> l = betInstanceDao.getGamesByChoice(c.getId());
230             if (l != null) {
231                 list.addAll(l);
232             }
233         }
234     }
235
236     return list;
237 }
238
239 public String encryptPassword(String password) {
240     return DigestUtils.md5Hex(password);
241 }
242
243 private String generatePassword() {
244     String pwd = "aaaaaaaaaa";
245     //TODO
246     return pwd;
247 }
248 }

```

## B.5 Controllers

### B.5.1 HomeController

```

1  *package com.boc.btv.web.controller;
2
3  import com.boc.btv.common.Database;
4  import com.boc.btv.model.User;
5  import com.boc.btv.service.UserManager;
6  import java.util.HashMap;
7  import java.util.Map;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.stereotype.Controller;
10 import org.springframework.ui.Model;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RequestMethod;
13 import org.springframework.web.bind.annotation.RequestParam;
14

```

```

15 @Controller
16 @RequestMapping("/")
17 public class HomeController {
18     @Autowired
19     private UserManager manager;
20
21     private Map<String, Object> map = new HashMap<String, Object>();
22
23     public HomeController() {
24         map.put("menu", "home");
25         map.put("path", "");
26     }
27
28     @RequestMapping("index")
29     public void indexHandler(Model model) {
30         model.addAllAttributes(map);
31         model.addAttribute("title",
32             "BetonTv : Paris sportifs, sur la TV et l'actualité.");
33         model.addAttribute("currentUrl", "/index.htm");
34     }
35
36     @RequestMapping("contact")
37     public void contactHandler(Model model) {
38         model.addAllAttributes(map);
39         model.addAttribute("title", "Contacter les webmestres BetOnTV");
40         model.addAttribute("currentUrl", "/contact.htm");
41     }
42
43     @RequestMapping("lost-password")
44     public void lostPasswordHandler(Model model) {
45         model.addAllAttributes(map);
46         model.addAttribute("title", "Mot de passe oublié");
47         model.addAttribute("currentUrl", "/index.htm");
48     }
49
50     @RequestMapping(value = "/lost-password", method = RequestMethod.POST)
51     public String lostPasswordHandler(
52         @RequestParam(value = "login", required = true) String login,
53         @RequestParam(value = "email", required = true) String email) {
54         Database.startTransaction(false);
55         User user = manager.getUser(login);
56         if (user == null) {
57             Database.closeTransaction();
58             return "lost-password";
59         }
60         if (!user.getEmail().equals(email)) {
61             Database.closeTransaction();
62             return "lost-password";
63         }
64         manager.regeneratePassword(login);
65         Database.closeTransaction();
66         return "redirect:/index.htm";
67     }
68
69     @RequestMapping("cgu")
70     public void cguHandler(Model model) {

```

```

71     model.addAllAttributes(map);
72     model.addAttribute("title", "Conditions Générales d'Utilisation de BetonTV");
73     model.addAttribute("currentUrl", "/cgu.htm");
74 }
75
76 @RequestMapping("mentions")
77 public void mentionsHandler(Model model) {
78     model.addAllAttributes(map);
79     model.addAttribute("title", "Mentions Légales de BetonTV");
80     model.addAttribute("currentUrl", "/mentions.htm");
81 }
82
83 @RequestMapping(value="checklogin", method=RequestMethod.POST)
84 public String checkLoginHandler(
85     @RequestParam(value="login", required=true) String login,
86     Model model) {
87     Database.startTransaction(true);
88     boolean r = manager.loginExists(login);
89     model.addAttribute("result", r ? "1" : "0");
90     Database.closeTransaction();
91     return "xml/boolean";
92 }
93
94 @RequestMapping(value="checkemail", method=RequestMethod.POST)
95 public String checkEmailHandler(
96     @RequestParam(value="email", required=true) String email,
97     Model model) {
98     Database.startTransaction(true);
99     boolean r = manager.mailExists(email);
100    model.addAttribute("result", r ? "1" : "0");
101    Database.closeTransaction();
102    return "xml/boolean";
103 }
104 }

```

### B.5.2 RegistrationController

```

1  *package com.boc.btv.web.controller;
2
3  import com.boc.btv.common.Database;
4  import com.boc.btv.common.UserStatus;
5  import com.boc.btv.model.User;
6  import com.boc.btv.service.UserManager;
7  import com.boc.btv.web.validator.UserValidator;
8  import java.text.SimpleDateFormat;
9  import java.util.Date;
10 import org.springframework.stereotype.Controller;
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.beans.propertyeditors.CustomDateEditor;
13 import org.springframework.ui.Model;
14 import org.springframework.validation.BindingResult;
15 import org.springframework.web.bind.WebDataBinder;
16 import org.springframework.web.bind.annotation.InitBinder;
17 import org.springframework.web.bind.annotation.RequestMapping;
18 import org.springframework.web.bind.annotation.RequestMethod;

```

```

19 import org.springframework.web.bind.annotation.SessionAttributes;
20 import org.springframework.web.bind.support.SessionStatus;
21
22 @Controller
23 @RequestMapping("/registration")
24 @SessionAttributes("user")
25 public class RegistrationController {
26
27     @Autowired
28     private UserManager userManager;
29     @Autowired
30     private UserValidator validator;
31
32     @InitBinder
33     public void initBinder(WebDataBinder dataBinder) {
34         dataBinder.setDisallowedFields(new String[]{"id"});
35         dataBinder.setRequiredFields(
36             new String[]{"login", "password", "checkPassword", "email", "title",
37                 "firstName", "lastName", "address0", "zipCode", "city", "country",
38                 "termsAccepted"});
39
40         SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
41         dateFormat.setLenient(true);
42         dataBinder.registerCustomEditor(Date.class,
43             new CustomDateEditor(dateFormat, true));
44     }
45
46     @RequestMapping(method = RequestMethod.GET)
47     public User setUpForm(Model model) {
48         model.addAttribute("menu", "home");
49         model.addAttribute("path", "");
50         model.addAttribute("title", "Formulaire d'inscription");
51         model.addAttribute("currentUrl", "/index.htm");
52         return new User();
53     }
54
55     @RequestMapping(method = RequestMethod.POST)
56     public String processForm(
57         User user,
58         BindingResult result,
59         SessionStatus status,
60         Model model) {
61         Database.startTransaction(false);
62         validator.validate(user, result);
63         if (result.hasErrors()) {
64             Database.closeTransaction();
65             return "registration";
66         }
67         user.setStatus(UserStatus.disabled);
68         userManager.saveUser(user);
69         status.setComplete();
70         Database.closeTransaction();
71         return "redirect:/index.htm";
72     }
73 }

```

### B.5.3 SectionController

```
1  *package com.boc.botv.web.controller;
2
3  import com.boc.botv.common.Database;
4  import com.boc.botv.model.Bet;
5  import com.boc.botv.model.Category;
6  import com.boc.botv.service.BetManager;
7  import com.boc.botv.service.CategoryManager;
8  import java.util.ArrayList;
9  import java.util.Date;
10 import java.util.List;
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.stereotype.Controller;
13 import org.springframework.ui.Model;
14 import org.springframework.web.bind.annotation.PathVariable;
15 import org.springframework.web.bind.annotation.RequestMapping;
16 import org.springframework.web.bind.annotation.RequestMethod;
17
18 @Controller
19 @RequestMapping("/")
20 public class SectionController {
21
22     @Autowired
23     private BetManager manager;
24     @Autowired
25     private CategoryManager categoryManager;
26
27     @RequestMapping(value = "sport/{subcat}", method = RequestMethod.GET)
28     public String sportHandler(
29         @PathVariable(value = "subcat") String subcat,
30         Model model) {
31         if (subcat == null || subcat.equals("")) {
32             return "redirect:/index.htm";
33         }
34         setPage("Sport", subcat, model);
35         return "section/index";
36     }
37
38     @RequestMapping(value = "television/{subcat}", method = RequestMethod.GET)
39     public String tvHandler(
40         @PathVariable(value = "subcat") String subcat,
41         Model model) {
42         if (subcat == null || subcat.equals("")) {
43             return "redirect:/index.htm";
44         }
45         setPage("Télévision", subcat, model);
46         return "section/index";
47     }
48
49     @RequestMapping(value = "actualites/{subcat}", method = RequestMethod.GET)
50     public String newsHandler(
51         @PathVariable(value = "subcat") String subcat,
52         Model model) {
53         if (subcat == null || subcat.equals("")) {
54             return "redirect:/index.htm";
55         }
56     }
```

```

55     }
56     setPage("Actualités", subcat, model);
57     return "section/index";
58 }
59
60 private void setPage(String section, String subsection, Model model) {
61     model.addAttribute("menu", section.toLowerCase());
62     model.addAttribute("submenu", subsection.toLowerCase());
63     model.addAttribute("path", "../");
64
65     Database.startTransaction(false);
66     List<Category> categories = new ArrayList<Category>();
67     if (subsection.equals("index")) {
68         subsection = section;
69         Category c = categoryManager.getCategory(subsection);
70         categories = categoryManager.getChilds(c.getId());
71         categories.add(c);
72         model.addAttribute("title", section + " :: Tous les paris");
73     } else {
74         subsection = Character.toUpperCase(subsection.charAt(0))
75             + subsection.substring(1);
76         categories.add(categoryManager.getCategory(subsection));
77         model.addAttribute("title", section + " > " + subsection
78             + " :: Tous les paris");
79     }
80
81     List<Bet> allBets = new ArrayList<Bet>();
82     List<Bet> closestBets = new ArrayList<Bet>();
83     for (Category c : categories) {
84         allBets.addAll(manager.getCurrentBetsBySubsection(c.getId()));
85         closestBets.addAll(manager.getBetsByCloserEndDate(2, c.getId()));
86     }
87
88     model.addAttribute("bet", allBets);
89     model.addAttribute("betsoon", closestBets);
90
91     Database.closeTransaction();
92 }
93 }

```

## B.6 Java Server Pages

### B.6.1 Header

```

1 <%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
2 <%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
3 <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
4 <!DOCTYPE html>
5 <html>
6 <head>
7     <title><c:out value="${title}" /></title>
8     <meta http-equiv="content-type" content="text/html; charset=UTF-8">
9     <link href="/css/design.css" type="text/css" rel="stylesheet" media="screen" />
10    <script type="text/javascript" src="/js/jquery.js"></script>
11    <script type="text/javascript" src="/js/jquery.tools.js"></script>
12    <script type="text/javascript" src="/js/vars.js"></script>

```

```

13     <script type="text/javascript" src="/js/anonymous.js"></script>
14     <script type="text/javascript" src="/js/user.js"></script>
15     <script type="text/javascript" src="/js/animations.js"></script>
16     <script type="text/javascript" src="/js/actions.js"></script>
17     <script type="text/javascript" src="/js/init.js"></script>
18 </head>
19 <body>
20     <header>
21         <sec:authorize ifAllGranted="ROLE_USER">
22             <%@include file="/WEB-INF/jsp/in-logout.jsp" %>
23         </sec:authorize>
24         <sec:authorize ifNotGranted="ROLE_USER">
25             <%@include file="/WEB-INF/jsp/in-login.jsp" %>
26         </sec:authorize>
27     </header>
28     <%@include file="/WEB-INF/jsp/menu/index.jsp" %>
29     <div id="content">
30         <div>

```

### B.6.2 Footer

```

1     </div>
2 </div>
3 <footer>
4     <nav id="info">
5         <ul>
6             <li><a href="/cgu.htm">Conditions Générales d'Utilisation</a></li>
7             <li><a href="/mentions.htm">Mentions Légales</a></li>
8             <li><a href="/contact.htm">Contact</a></li>
9         </ul>
10    </nav>
11 </footer>
12 </body>
13 </html>

```

### B.6.3 Index

```

1 <%@ page contentType="text/html" pageEncoding="UTF-8" language="java"%>
2 <%@include file="/WEB-INF/jsp/header.jsp" %>
3
4 <section id="home">
5     <div>
6         <div class="container">
7             <canvas id="stadium" class="loading h300"></canvas>
8             <article class="h300 animation">
9                 <h1>Coupe du monde de football</h1>
10                <p>Jusqu'où ira votre équipe ? Pariez dès aujourd'hui !</p>
11            </article>
12        </div>
13        <div class="box container w400">
14            <canvas id="tennis" class="loading h150"></canvas>
15            <article class="h150 animation">
16                <h1>Sport &rsquo; Rolland Garros</h1>
17                <p>Nadal ou Söderling ?</p>

```

```

18     </article>
19 </div>
20 <div class="box container w400">
21     <canvas id="tbbt" class="loading h150"></canvas>
22     <article class="h150 animation">
23         <h1>Télévision &rsquo;</h1>
24         <p>Pariez sur l'audience de vos séries préférées.</p>
25     </article>
26 </div>
27 <div class="box container wend">
28     <canvas id="jt" class="loading h150"></canvas>
29     <article class="h150 animation">
30         <h1>Actualités &rsquo;</h1>
31         <p>Pariez sur l'actualité.</p>
32     </article>
33 </div>
34 </div>
35 </section>
36 <%@include file="/WEB-INF/jsp/footer.jsp" %>

```

#### B.6.4 In-Login

```

1 <%@ page contentType="text/html" pageEncoding="UTF-8" language="java"%>
2 <nav id="userlinks">
3     <ul>
4         <li>
5             <a href="/lost-password.htm" rel="nofollow">Mot de passe oublié ?</a>
6         </li>
7         <li>
8             <a href="/registration.htm" rel="nofollow">Inscrivez vous !</a>
9         </li>
10    </ul>
11 </nav>
12 <form id="login-area" action="/login.htm" method="post">
13     <div>
14         <input type="text" required="required" name="username" value="" >
15         <input type="password" required="required" name="password" value="" >
16         <input type="hidden" name="from" value="<c:url value='${currentUrl}'/>">
17         <div class="submit-button">
18             <div class="top"></div>
19             <div class="content">Connexion</div>
20         </div>
21     </div>
22 </form>

```

#### B.6.5 Out-Login

```

1 <%@ page contentType="text/html" pageEncoding="UTF-8"%>
2 <nav id="userlinks">
3     <ul>
4         <sec:authorize ifNotGranted="ROLE_ADMIN">
5             <li>
6                 Bienvenue
7                 <sec:authentication property="principal.title"/>

```

```

8         <sec:authentication property="principal.firstname"/>
9         <sec:authentication property="principal.lastname"/>
10    </li>
11    <li>Solde : <sec:authentication property="principal.balance"/> euros</li>
12    <li>
13        <a href="/profil/<sec:authentication property="principal.login"/>
14            /index.htm">Mon profil</a>
15    </li>
16 </sec:authorize>
17 <sec:authorize ifAllGranted="ROLE_ADMIN">
18     <li>Bonjour Monsieur l' Administrateur</li>
19     <li><a href="/admin/index.htm" rel="nofollow">Administration</a></li>
20 </sec:authorize>
21 <li>
22     <a href="/logout.htm?from=<c:url value="{currentUrl}"/>">Déconnexion</a>
23 </li>
24 </ul>
25 </nav>

```

### B.6.6 Registration

```

1 <%@ page contentType="text/html" pageEncoding="UTF-8" language="java"%>
2 <%@include file="/WEB-INF/jsp/header.jsp" %>
3 <section id="registration">
4     <div>
5         <div id="errors">Coucou</div>
6         <form:form modelAttribute="user" action="/registration.htm" method="post">
7             <div class="items">
8                 <fieldset>
9                     <legend>Identifiants</legend>
10                    <p>
11                        <label>Pseudo <span class="required">*</span> </label>
12                        <form:input path="login" required="required"/>
13                        <form:errors path="login" cssClass="errors"/>
14                    </p>
15                    <p>
16                        <label>Mot de passe <span class="required">*</span> </label>
17                        <form:password path="password" required="required"/>
18                        <form:errors path="password" cssClass="errors"/>
19                    </p>
20                    <p>
21                        <label>
22                            Confirmation du mot de passe <span class="required">*</span> :
23                        </label>
24                        <form:password path="checkPassword" required="required"
25                            data-equals="password"/>
26                    </p>
27                    <p>
28                        <label>Courriel <span class="required">*</span> </label>
29                        <input name="email" type="email" required="required" value="" />
30                        <form:errors path="email" cssClass="errors"/>
31                    </p>
32                    <div class="browse next submit-button">
33                        <div class="top"></div>
34                        <div class="content">Étape suivante &rsaquo;</div>

```

```

35     </div>
36 </fieldset>
37 <fieldset>
38     <legend>Données personnelles</legend>
39     <p>
40         <label>Civilité <span class="required">*</span> :</label>
41         <form:radiobutton path="title" value="0" />
42         <label>Monsieur</label>
43         <form:radiobutton path="title" value="1" />
44         <label>Madame</label>
45         <form:radiobutton path="title" value="2" />
46         <label>Mademoiselle</label>
47         <form:errors path="title" cssClass="errors" />
48     </p>
49     <p>
50         <label>Prénom <span class="required">*</span> :</label>
51         <form:input path="firstName" required="required" />
52         <form:errors path="firstName" cssClass="errors" />
53     </p>
54     <p>
55         <label>Nom <span class="required">*</span> :</label>
56         <form:input path="lastName" required="required" />
57         <form:errors path="lastName" cssClass="errors" />
58     </p>
59     <p>
60         <label>Adresse <span class="required">*</span> :</label>
61         <form:input path="address0" required="required" />
62         <form:errors path="address0" cssClass="errors" />
63     </p>
64     <p>
65         <label></label>
66         <form:input path="address1" />
67     </p>
68     <p>
69         <label></label>
70         <form:input path="address2" />
71     </p>
72     <p>
73         <label>Code Postal <span class="required">*</span> :</label>
74         <form:input path="zipCode" required="required" />
75         <form:errors path="zipCode" cssClass="errors" />
76     </p>
77     <p>
78         <label>Ville <span class="required">*</span> :</label>
79         <form:input path="city" required="required" />
80         <form:errors path="city" cssClass="errors" />
81     </p>
82     <p>
83         <label>Pays <span class="required">*</span> :</label>
84         <form:input path="country" required="required" />
85         <form:errors path="country" cssClass="errors" />
86     </p>
87     <p>
88         <label>
89             Date de naissance <span class="required">*</span> :
90         </label>

```

```

91         <input name="birthDate" type="date" required="required"
92             value="1970-01-01"/>
93         <form:errors path="birthDate" cssClass="errors"/>
94     </p>
95     <p>
96         <label>Téléphone fixe :</label>
97         <form:input path="phoneHome"/>
98     </p>
99     <p>
100        <label>Téléphone portable :</label>
101        <form:input path="phoneGsm"/>
102    </p>
103    <div class="browse prev submit-button">
104        <div class="top"></div>
105        <div class="content">&lsaquo; Étape précédente</div>
106    </div>
107    <div class="browse next submit-button">
108        <div class="top"></div>
109        <div class="content">Étape suivante &rsaquo;</div>
110    </div>
111 </fieldset>
112 <fieldset>
113     <legend>Confirmation</legend>
114     <p>
115         <label>Conditions Générales d'Utilisation</label>
116     </p>
117     <p>
118         <textarea name="cgu" cols="80" rows="13" readonly="readonly">
119             — CGU ici —
120         </textarea>
121     </p>
122     <p>
123         <form:checkbox path="termsAccepted" value="1" required="required"/>
124         <label>
125             J'ai lu et j'accepte les conditions générales d'utilisation
126         </label>
127         <form:errors path="termsAccepted" cssClass="errors"/>
128     </p>
129     <p>
130         <form:checkbox path="acceptNewsletterSite" value="1"/>
131         <label>J'accepte de recevoir la newsletter du site</label>
132     </p>
133     <p>
134         <form:checkbox path="acceptNewsletterPartners" value="1"/>
135         <label>
136             J'accepte de recevoir les newsletters des partenaires de bet-on-tv.com
137         </label>
138     </p>
139     <div class="browse prev submit-button">
140         <div class="top"></div>
141         <div class="content">&lsaquo; Étape précédente</div>
142     </div>
143     <div class="next submit-button">
144         <div class="top"></div>
145         <div class="content">Terminer l'inscription ! &rsaquo;</div>
146     </div>

```

```
147         </fieldset>
148     </div>
149 </form:form>
150
151 </div>
152 </section>
153 <%@include file="/WEB-INF/jsp/footer.jsp" %>
```

---