|  |  |
| --- | --- |
| BobbyStone  Hearthstone Like | Résumé  Nous avons créé un jeu de cartes de type Hearthstone réalisé avec Django Rest Framework côté serveur et Angular côté client. 2 joueurs doivent se connecter à une même partie afin que celle-ci se lance ; le projet n’intègre pas d’intelligence artificielle.  Auteur  PYCKAERT Valerian, DELAUNAY Simon, CORNAND Enzo, MIRABILE Thomas |

Table des matières

[1 Installation du projet 3](#_Toc507621794)

[1.1 API 3](#_Toc507621795)

[1.2 Client 3](#_Toc507621796)

[1.3 Jouer 3](#_Toc507621797)

[2 Administration 4](#_Toc507621798)

[2.1 Gestion des utilisateurs 4](#_Toc507621799)

[2.2 Gestion des cartes, type de cartes, des decks et des effets 4](#_Toc507621800)

[2.3 Gestion des règles 5](#_Toc507621801)

[2.4 Gestion des logs 5](#_Toc507621802)

[3 Fonctionnalités joueur 6](#_Toc507621803)

[4 Livrables 7](#_Toc507621804)

[4.1 Gestion de la partie Admin 7](#_Toc507621805)

[4.2 Gestion des comptes utilisateurs 7](#_Toc507621806)

[4.3 Gestion des cartes & decks 7](#_Toc507621807)

[4.4 Gestion des règles 8](#_Toc507621808)

[4.5 Gestion des logs 8](#_Toc507621809)

[4.6 Consulter ses decks 8](#_Toc507621810)

[4.7 Rejoindre une partie 9](#_Toc507621811)

[4.8 Jouer la partie 9](#_Toc507621812)

[5 Documentation technique et fonctionnelle 10](#_Toc507621813)

[5.1 Processus de navigation 10](#_Toc507621814)

[5.2 Architecture 10](#_Toc507621815)

[5.2.1 Légende & Hyperlien 10](#_Toc507621816)

[5.2.2 Schéma de BDD 11](#_Toc507621817)

[5.3 Description des écrans 11](#_Toc507621818)

[5.3.1 Connexion 11](#_Toc507621819)

[5.3.2 Accueil 12](#_Toc507621820)

[5.3.3 Partie 12](#_Toc507621821)

[5.3.4 Deck 13](#_Toc507621822)

[5.3.5 Créer un deck 13](#_Toc507621823)

[5.3.6 Règles 14](#_Toc507621824)

[5.3.7 Plateau de jeu 15](#_Toc507621825)

[5.3.8 Action en jeu 16](#_Toc507621826)

[5.3.9 Fin de la partie 16](#_Toc507621827)

[5.3.10 Historique des parties (Game Log) 17](#_Toc507621828)

[5.3.11 Historique des actions lors d’une partie (Game Log Actions) 17](#_Toc507621829)

# Installation du projet

Pour faire un setup complet du projet il faut dans le dossier client procéder comme suit :

## API

* python manage.py makemigrations app
* python manage.py migrate
* Supprimer la base nouvellement créée dans le dossier api et remplacer par la base fournis à la racine du fichier

## Client

* Npm install
* Npm start

## Jouer

* Ouvrir Google Chrome !important ;
* Ouvrir une nouvelle fenêtre en navigation privé (CTRL + MAJ + N)

1ère fenêtre

**ID :** thomi

**MDP :** thomi thomi

2nd fenêtre

**ID :** bobby

**MDP :** bobby bobby

**Avec un des comptes cliquer sur :**

* JOUER
* Créer une partie
* Choisir, cartes + règles
* Cliquer sur OK

**Avec le compte restant**

* JOUER
* Rejoindre la partie créer

Have Fun !

# Administration

La partie administration de notre projet propose les gestions suivantes :

* Utilisateurs
* Cartes & Deck
* Type de cartes
* Règles
* Logs
* Groupes d’utilisateurs

## Gestion des utilisateurs

La gestion des utilisateurs est gérée par le module d’administration de Django Rest Framework, nous avons surchargé l’utilisateur pour qu’il puisse être associé à une partie et qu’il ait un éventail de decks disponibles. De plus, nous avons mis en place l’authentification en JWT pour les utilisateurs ainsi que l’association des utilisateurs inscrits depuis le client au groupe Player, si ce dernier existe. Le groupe Player empêche aux joueurs de se connecter à l’interface d’administration de Django.

## Gestion des cartes, type de cartes, des decks et des effets

Notre application propose aux joueurs de constituer leurs decks à l’aide des différentes cartes de jeu.

Les cartes ont un nom, un type (monstre ou sort), un coût, de la vie (si monstre), de la force (si monstre) et une image. Ils peuvent aussi avoir un effet.

Les effets sont créés séparément et disposent de ces propriétés : nom, description, joueur ciblé, type de sort (dégât ou soin) et le nombre de dégât ou de soin.

Il existe des decks liés à aucun utilisateur, c’est normal, ils sont disponibles à tous les utilisateurs et seulement les admins peuvent les modifier. Sinon un utilisateur peut se constituer son deck depuis le client en ayant accès à toutes les données des cartes.

## Gestion des règles

Ce jeu de carte se joue tour par tour. Durant son tour, le joueur peut poser des monstres sur sa partie de terrain ou lancer des sorts.

Cependant, chacune des cartes jouées a un coût en mana.

Les monstres invoqués ne peuvent pas attaquer qu'au tour d'après et ils peuvent attaquer soit les monstres de l'adversaire soit l'adversaire lui-même et ainsi lui faire baisser ses points de vie. <br>

Le perdant est le joueur n'ayant plus de points de vie en premier.

De plus il y a la possibilité de créer ses propres variantes en créant de nouvelles règles qui permettront de modifier certaines valeurs des règles :

* Nombre points de vie des joueurs
* Nombre de cartes dans le deck
* Nombre de cartes dans la première main
* Nombre de cartes à tirer au début de son tour

## Gestion des logs

Afin de sauvegarder les informations des différentes parties et éviter les problèmes de triche, nous avons mis en place un système de logs.

D’autres part, ces logs nous permettront éventuellement dans une seconde version de l’application de faire des statistiques sur les performances des joueurs.

Les logs sont divisés en deux parties :

* L’entité GameLog qui est lié à une Game a les informations standard de la partie : début de la partie, fin de la partie, nombre de rounds, gagnant et perdant.
* L’entité GameLogAction qui est lié un GameLog a les informations détaillées de la partie : la date de l’action, le tour, l’utilisateur, l’action qui s’est déroulée et les données brutes qui ont transitées entre le client et le serveur.

# Fonctionnalités joueur

* Créer un compte
* Se connecter et se déconnecter
* Voir toutes les règles
* Voir les decks disponibles à tous les utilisateurs, créer et modifier ses propres decks
* Créer/Rejoindre/Jouer une partie

# Livrables

## Gestion de la partie Admin

Concepteur : Thomas Mirabile

Responsable : Thomas Mirabile

Plan d’action : 14/12/2017 - 15/12/2017

Architecture & concepts : Utilisation de la partie Back office de Django

Fonctionnalités liées : Avoir des fields organisées afin de gérer plus facilement la création de carte, deck, user etc.

Retour d’expérience : Le backoffice natif de Django est une bonne découverte, il nous a fait gagner un temps considérable pour la création des différentes entités.

## Gestion des comptes utilisateurs

Concepteur : Thomas Mirabile

Responsable : Valérian Pyckaert

Plan d’action : 14/12/2017 - 15/12/2017

Architecture & concepts : Utilisation des utilisateurs Django avec la surcouche Django Rest Framework, mise en place d’une authentification par token.

Fonctionnalités liées : Création des decks, Créer une partie, rejoindre une partie.

Retour d’expérience : Pas de difficultés particulières sauf pour la mise en place de l’authentification par token.

## Gestion des cartes & decks

Concepteur : Valérian Pyckaert

Responsable : Simon Delaunay

Plan d’action : 14/12/2017 – 15/12/2017

Architecture & concepts : Modélisation des données (Models.py), relations entre entités, sérialisations des models.

Fonctionnalités liées : Rejoindre une partie, Jouer la partie, Créer une partie, Créer son deck, modifier son deck, poser une carte.

Retour d’expérience : Découverte du mapping de données avec Django et des « serializers ».

## Gestion des règles

Concepteur : Enzo Cornand

Responsable : Thomas Mirabile

Plan d’action : 11/01/2018 – 12/01/2018

Architecture & concepts : Mise en place d’une entité de paramétrage.

Fonctionnalités liées : Rejoindre / créer la partie, jouer.

Retour d’expérience : Mises en place en dernier, ce fut quelque chose de plutôt simple à avoir, il n’y avait que quelques conditions à rajouter dans la gestion de la partie.

## Gestion des logs

Concepteur : Valerian Pyckaert

Responsable : Enzo Cornand

Plan d’action : 01/02/2018 – 02/02/2018

Architecture & concepts : Une entité pour le log général de partie et une entité pour le log de chaque action de joueur.

Fonctionnalités liées : Jouer la partie, consultation des statistiques (optionnel).

Retour d’expérience : Les logs sont arrivés en dernier et de même que les règles, ils ont été très simples à mettre en place. Simplement un objet à créer durant les actions de la partie et à sauvegarder.

## Consulter ses decks

Concepteur : Simon Delaunay

Responsable : Thomas Mirabile

Plan d’action : 21/12/2017 – 22/12/2017

Architecture & concepts :

Fonctionnalités liées : Créer son deck, modifier son deck.

Retour d’expérience : Utilisation des serializers dans les vues.

## Rejoindre une partie

Concepteur : Simon Delaunay

Responsable : Enzo Cornand

Plan d’action : 18/01/2018 – 19/01/2018

Architecture & concepts : WebSocket

Fonctionnalités liées : Jouer la partie

Retour d’expérience : La documentation de Django Channels était au top et m’a permis de créer ce que je cherchais à faire, des groupes d’utilisateurs communiquant en temps réel avec le serveur.

## Jouer la partie

Concepteur : Enzo Cornand

Responsable : Enzo Cornand

Plan d’action : 25/01/2018 – 28/02/2018

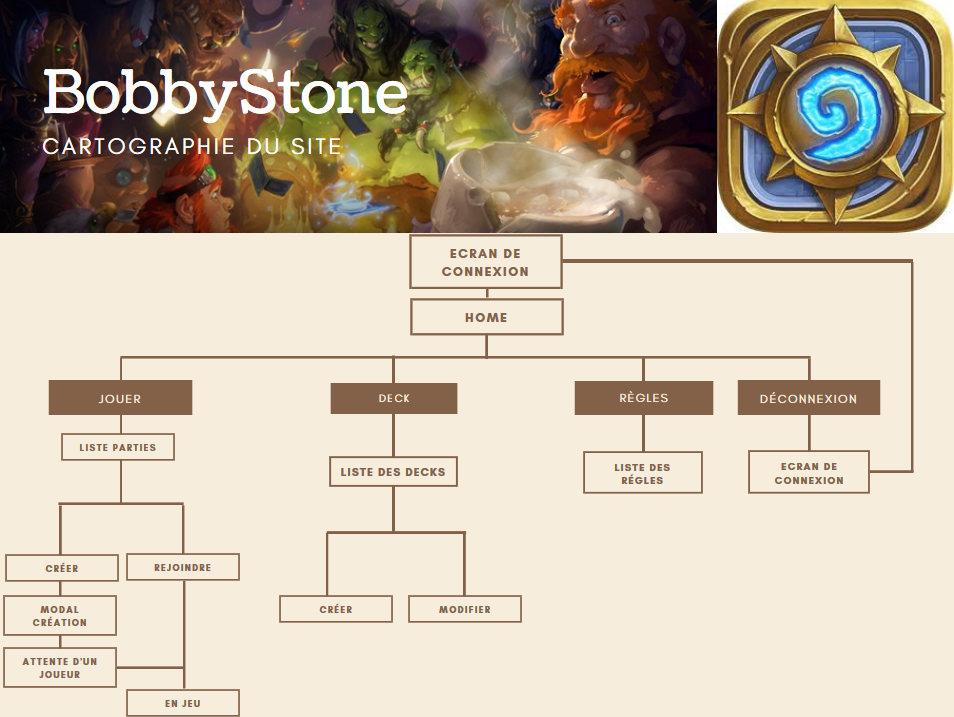
Architecture & concepts : Architecture d’envois de messages avec des données entre les clients et le serveur en WebSocket.

Fonctionnalités liées : Log

Retour d’expérience : Nous avons un système de message ayant des données transitant entre les clients et le serveur, à la réception sur le serveur, nous avons toutes les données nécessaires pour savoir quoi faire : la partie, l’émetteur, l’action et les données nécessaires à l’action.   
De là, nous vérifions que les données envoyées concordent avec les données de la partie enregistrées en base de données. Si les données sont bonnes, nous effectuons l’action et enregistrons le nouvel état de la partie en base de données. Du côté du client, l’émetteur aura déjà vu l’action se déroulait, si le message de retour du serveur dit que l’action s’est bien déroulé alors rien ne se passe, c’est seulement si l’action s’est mal déroulée alors nous récupérons les données de la partie depuis le serveur ; l’autre joueur quant à lui ne verra l’action se déroulait que si le serveur dit que l’action s’est bien déroulée, il recevra les mêmes données qu’a reçu le client et refera la même partie de code qu’a déjà fait l’émetteur.   
Ainsi très peu de données se transmettent en Websocket et nous avons une optimisation du temps de réponse de l’application et des cas d’erreur.  
Tout ceci est géré entre 4 actions : l’initialisation de la partie quand 2 joueurs sont présents, poser une carte, attaquer et terminer son tour.

# Documentation technique et fonctionnelle

## Processus de navigation



## Architecture

Schéma interactif, possibilité de voir les relations en passant la souris sur chaque lien entre deux tables. Cliquer sur **Fork** dans la barre de navigation pour pouvoir éditer. A droite du nom des tables, cliquer sur le logo  afin de voir le type de relation.

### Légende & Hyperlien

: **User**

: **Game**

: **Cartes**

: **Deck**

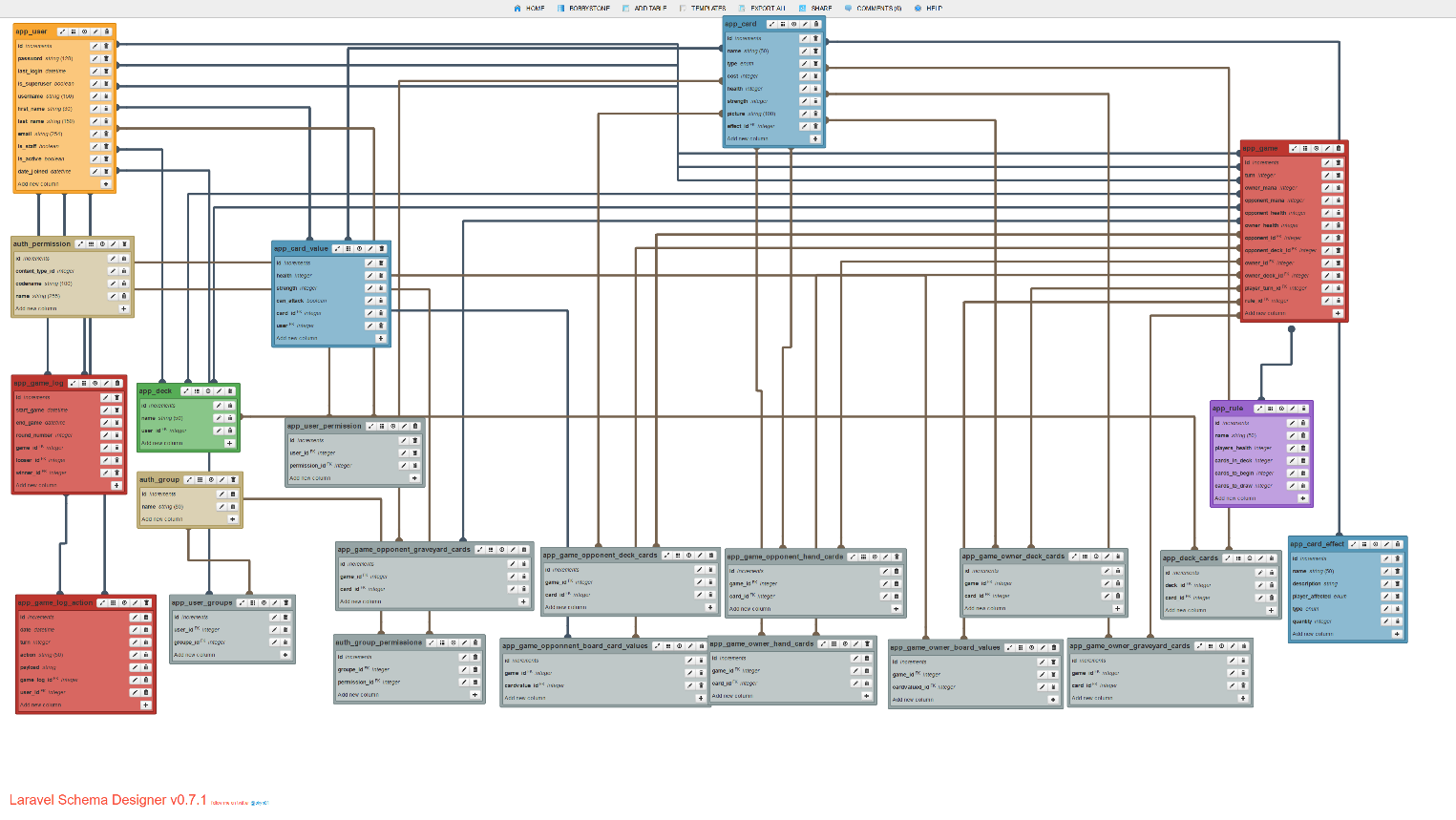
: **Règles**

: **Permissions/groupes**

: **Table Pivot**

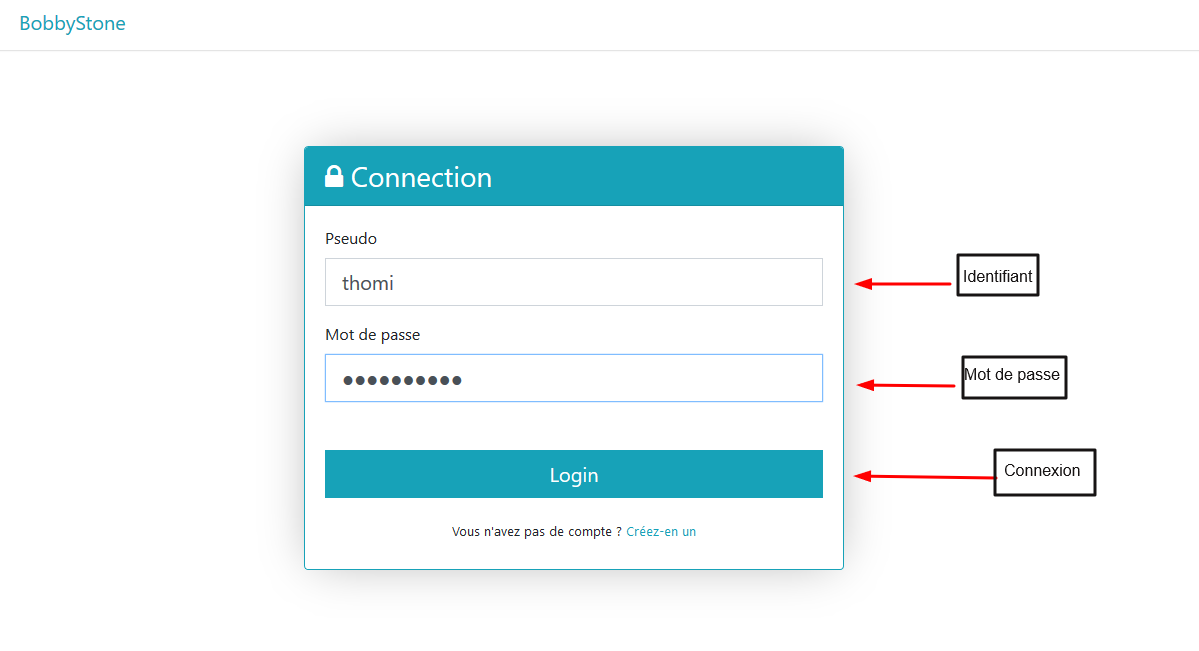
* <http://www.laravelsd.com/share/zcPrRg>

### **Schéma de BDD**



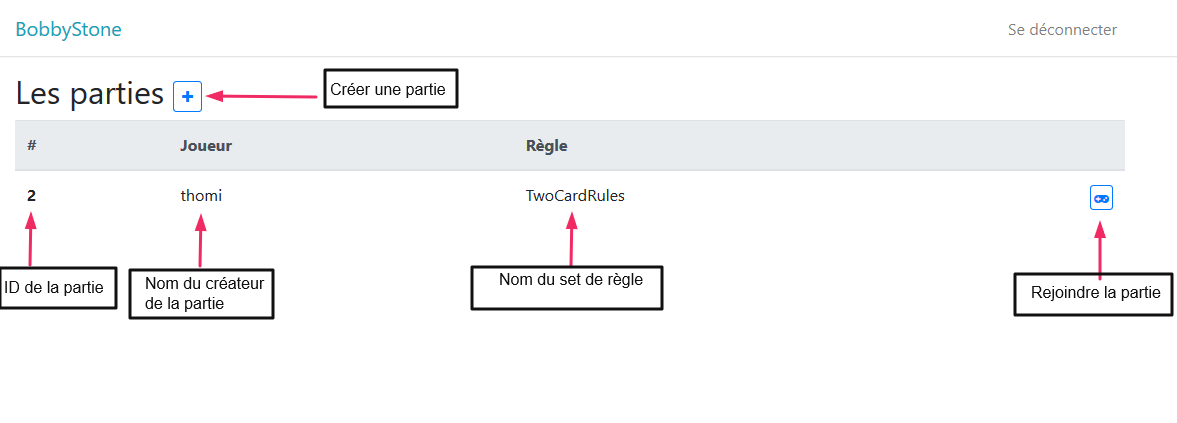
## Description des écrans

### Connexion



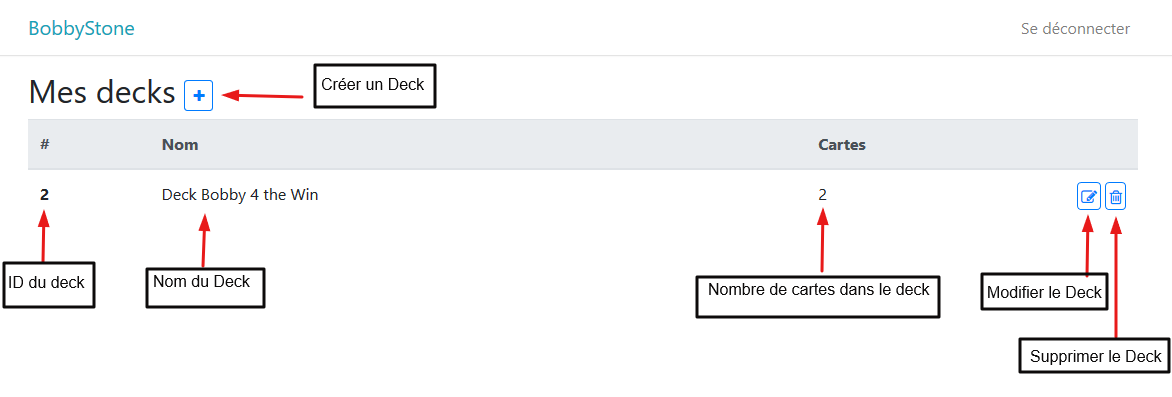
### Accueil

### Partie

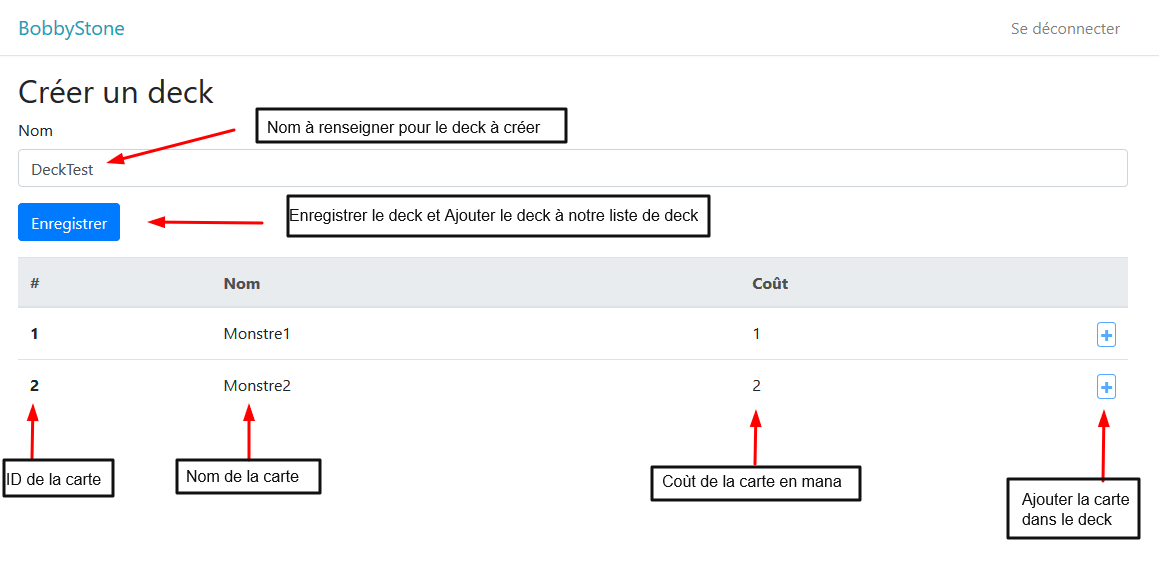


#### Création de parties

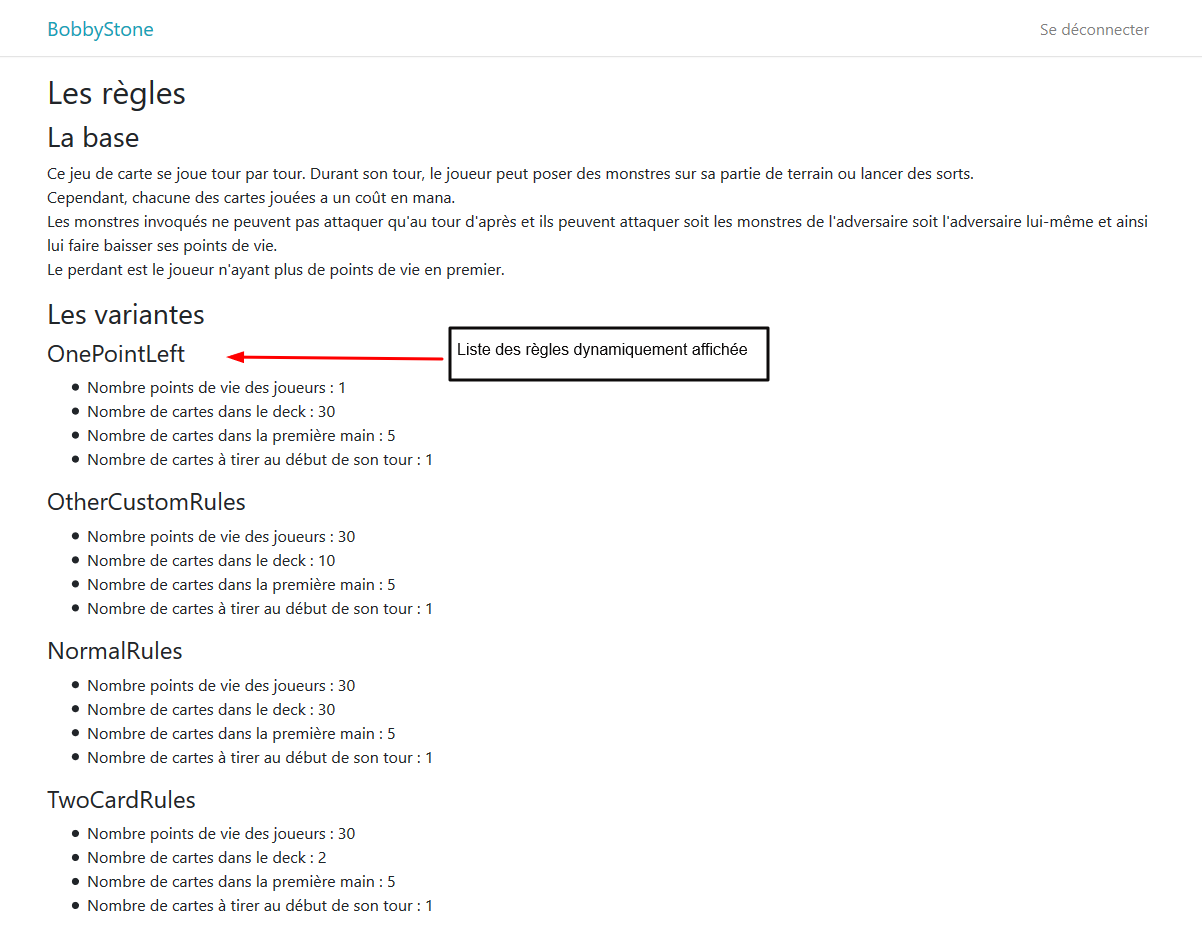
### Deck



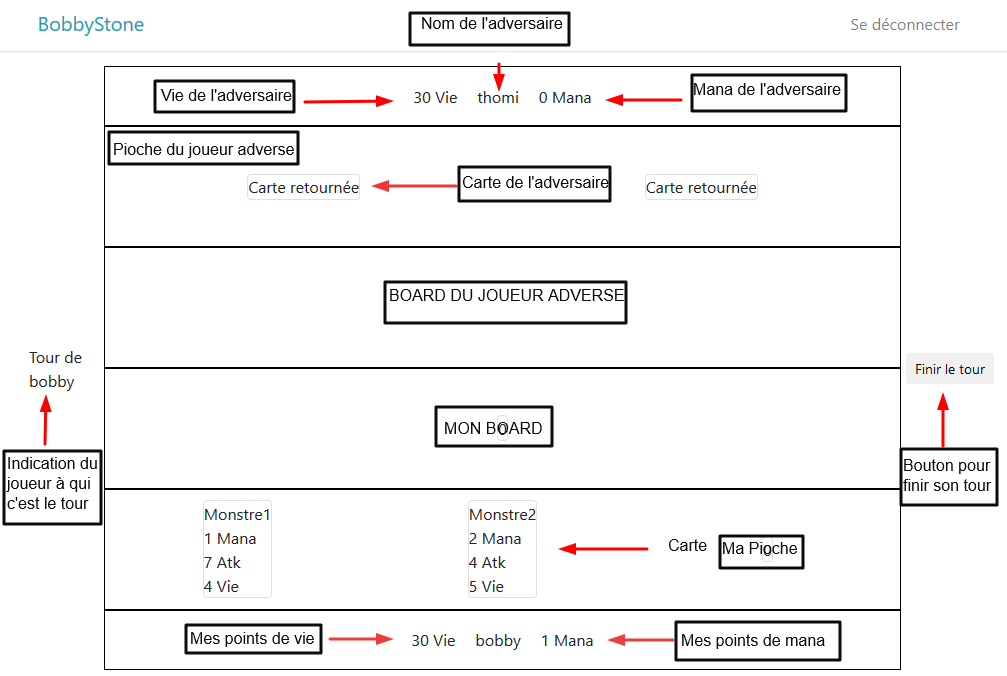
### Créer un deck



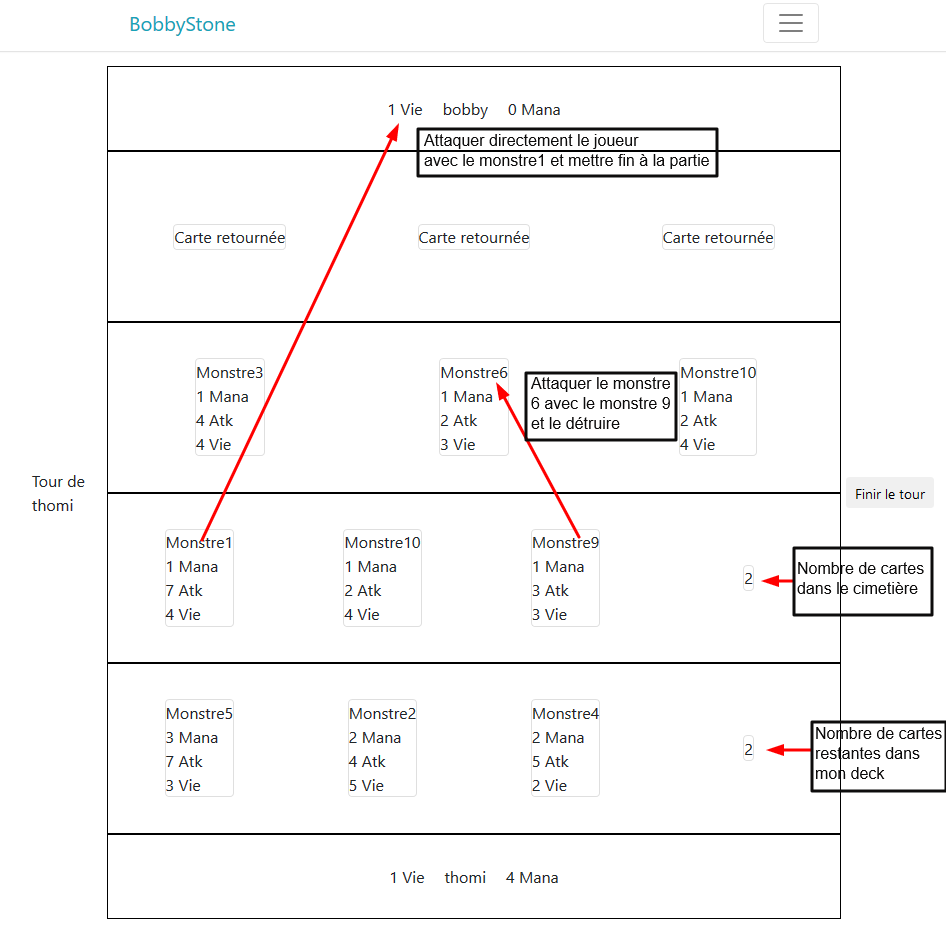
### Règles



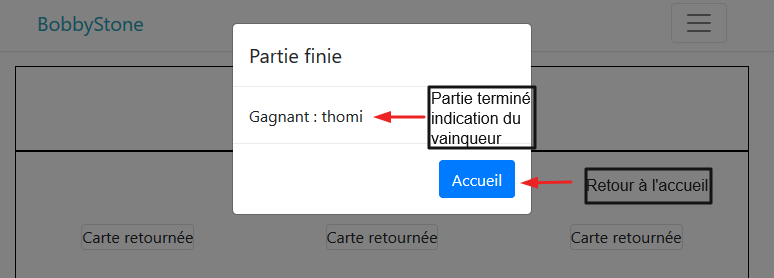
### Plateau de jeu



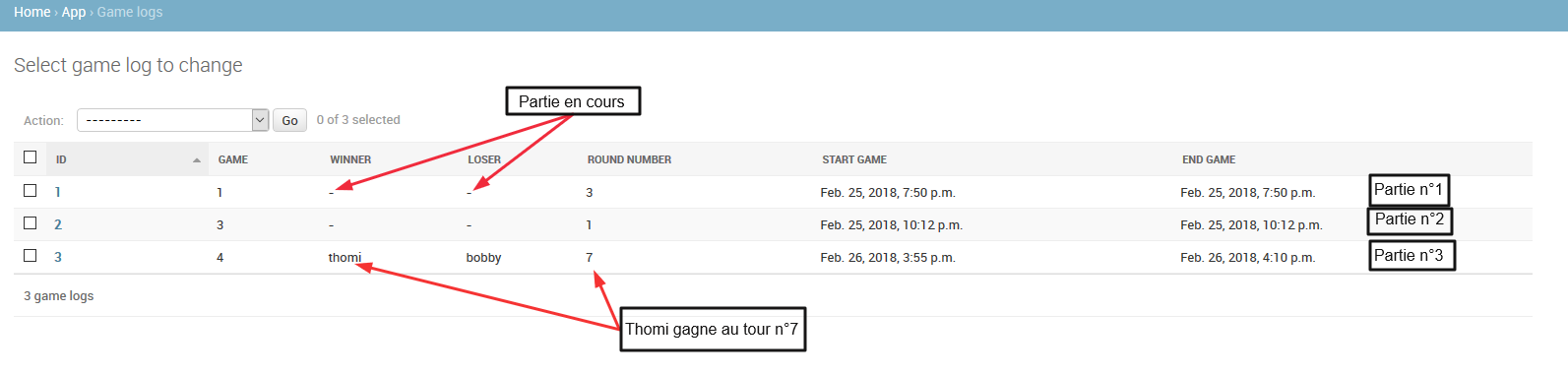
### Action en jeu



### Fin de la partie



### Historique des parties (Game Log)



### Historique des actions lors d’une partie (Game Log Actions)

