Grégoire Boiron Florian Vidal Théo Le Donne

Bases de données distribuées

8INF803

Travaux Pratiques 1

Le sort du dernier recours.
Page Ranking.

Table des matières

Introduction

Ce rapport a pour but d'améliorer la compréhension de l'ensemble du projet, les choix techniques faits ainsi que la navigation dans le code.

- **Les parties 1 et 2** sont écrites pour comprendre la structure du projet et celle du code. Elles faciliteront la navigation dans le code.

- **La partie 3** permet de mieux comprendre la base de données SQLite utilisée. Cette base de donnée conditionne l'application et oblige a faire certaines requêtes et jointures en plus.

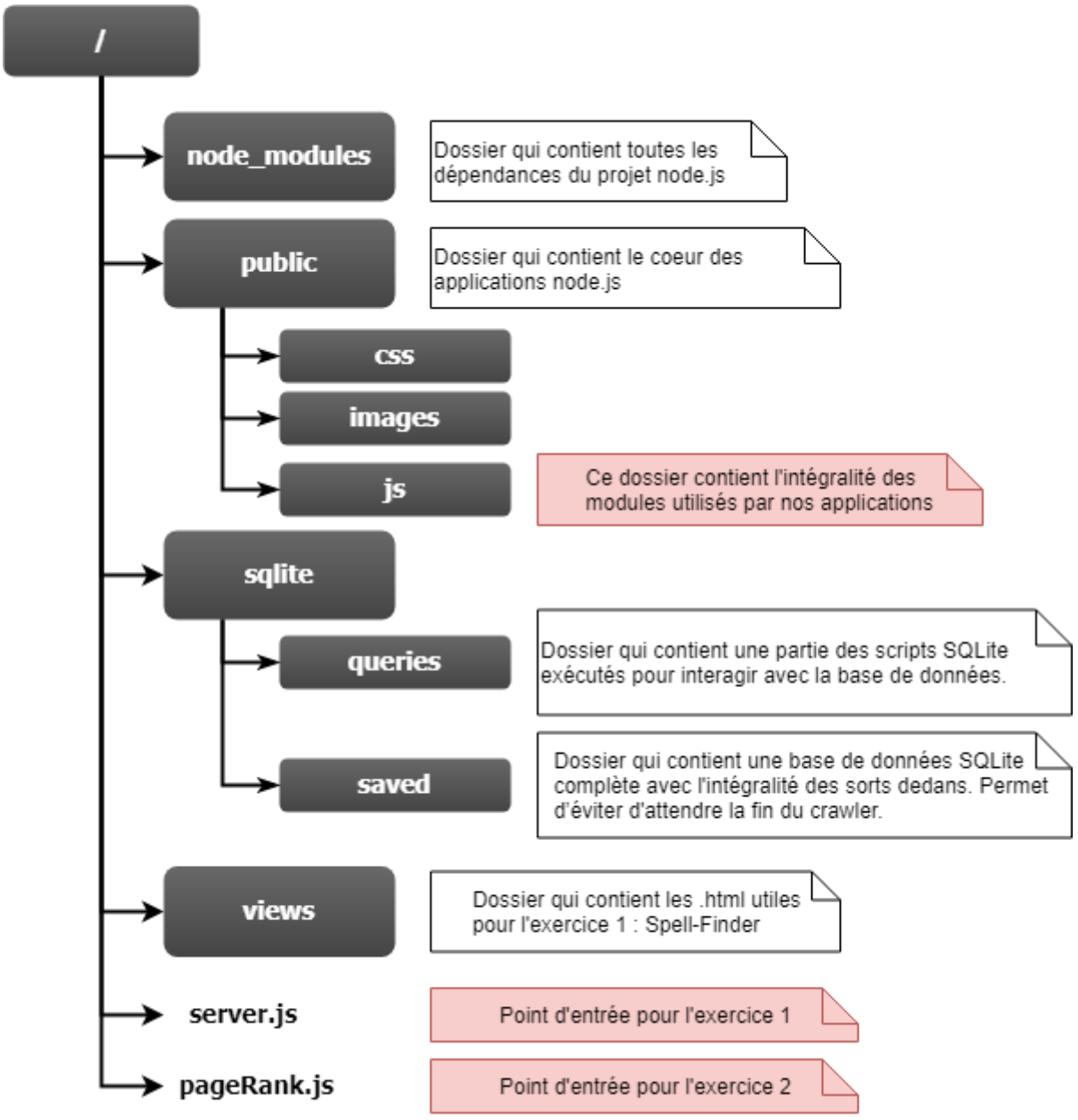
- Enfin, **la partie 4** permet de lancer les scripts et de vérifier les résultats. De brèves explications ainsi que des screenshots sont donnés pour arriver à des résultats cohérents.

I - L’arborescence du git

L’ensemble des fichiers des applications sont présents sur GitHub à l'adresse suivante : https://github.com/Graygzou/Spells-Finder

Schéma

Pour comprendre rapidement l’ensemble du git, voici une brève description des dossiers et fichiers présents. Les rectangles gris représentent des dossiers et les textes seulement des fichiers. Une note est écrite pour les dossiers importants.



II - Schéma des modules js

Pour se retrouver dans le code de l'exercice 1, il est possible de représenter les relations entre les modules par un schéma. Ce schéma est le suivant :

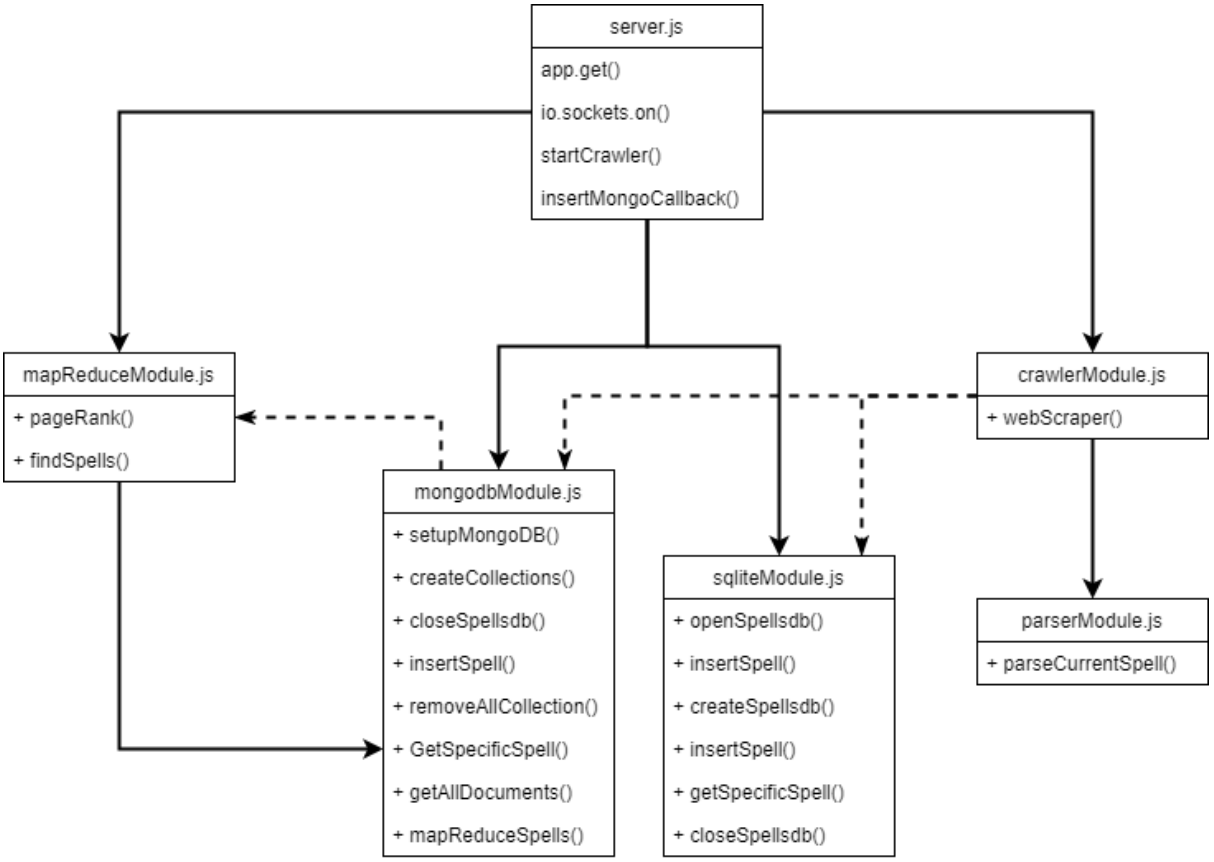


Illustration des dépendances entre modules et avec le point d'entrée.

Légende :

- Les flèches pleines correspondent à une dépendance concrète ("*require(xxxx)*" en haut du fichier + appels des méthodes du module) t
- Les flèche en pointillées correspondent aux appels à travers des callbacks (insertion des données dans les BDD, appels des fonctions map et reduce, etc.).

On peut voir, que la partie droite du schéma correspond à la première partie de l'exercice. Et celle de gauche la deuxième partie.

III - Présentation du schéma SQLite

Pour répondre à la question 3, nous avons décidé de respecter le schéma ci-dessous. La représentation de ce schéma correspond à un Modèle Conceptuel des Données (MCD) de la méthode Merise.

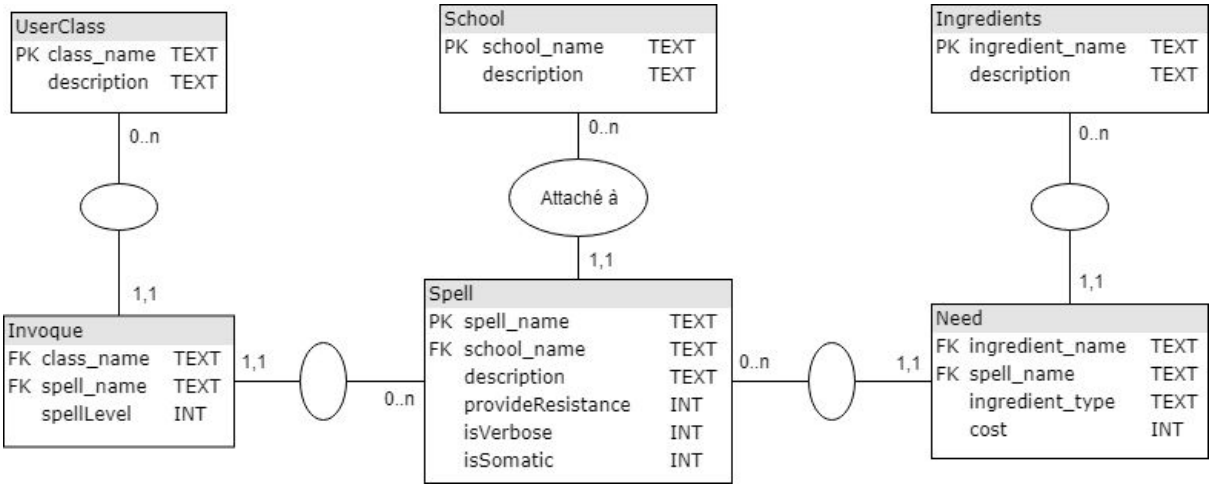
Explications

Les relations entre les différentes tables se lisent de la façon suivante :

Relation Spell → *School* :
un *Spell* est attaché à **1,1** *School*

Relation *School* → *Spell* :
une *School* attache **0,n** *Spell*

En d'autre mots, une école peut posséder aucun ou plusieurs sorts (on peut imaginer qu'une nouvelle école arrive et n'a pas encore conçu de sorts..). Et un sort n'appartient qu'à une et une seule école : on ne peut pas avoir un sort de l'école "*Necromancy*" et "*Conjuration*" à la fois !



MCD de la base de données des spells.

L'avantage d'une telle représentation est que l'on peut facilement récupérer tous les sorts d'une classe donnée, avec ou sans un filtre sur le niveau du sort. De même pour les écoles.

La table ingrédients devait regrouper les "*Materials"* utiles pour les components Material/ Focus / Divine Focus. Les éléments de ces tables n'ont finalement pas été insérés par manque de temps.

Remarques

En ce qui concerne la partie droite du diagramme, (tables *Need* / *Ingredients*), elles ne sont pour l'instant pas utilisées par les algorithmes. En effet, elles n'étaient pas indispensables pour répondre à la problématique de Pito.
Cependant,sans cette table, aucun Focus / Materials ne peut être pris en compte. Ce qui peut être une feature intéressante.

IV - Utilisation des scripts

Note : Il faut qu'une instance de base de données MongoDB soit lancé pour que les deux applications se déroulent correctement. Si ce n'est pas le cas, les applications crasheront.

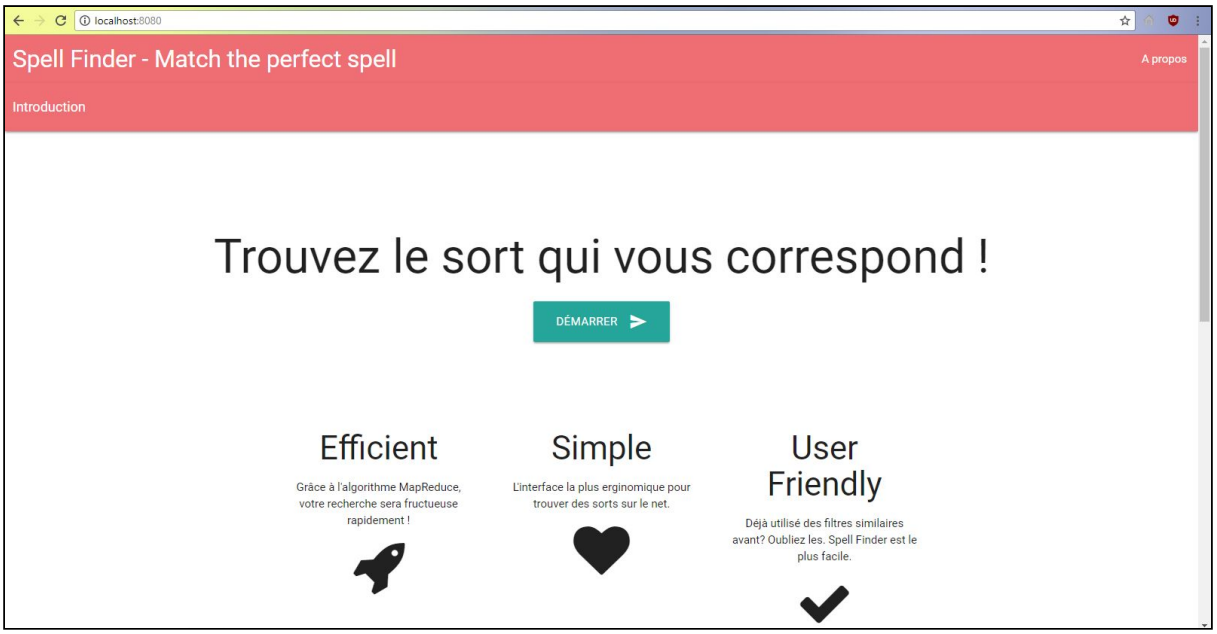
Exercice 2 - Page Rank

Pour tester l'exercice 2, il suffit de lancer le script suivant, localisé à la racine du projet : **pageRank.js.**
Une fois ce script lancé, le page rank s'effectuera et affichera les résultats à la console. Les résultats obtenus correspondent à ceux de l'énoncé.
La liste de liens sortants d'un noeud ("outlink_list") contient le noeud en question afin d'effectuer la fonction *reduce()* correctement mais ce dernier est ignoré dans le calcul du pagerank.

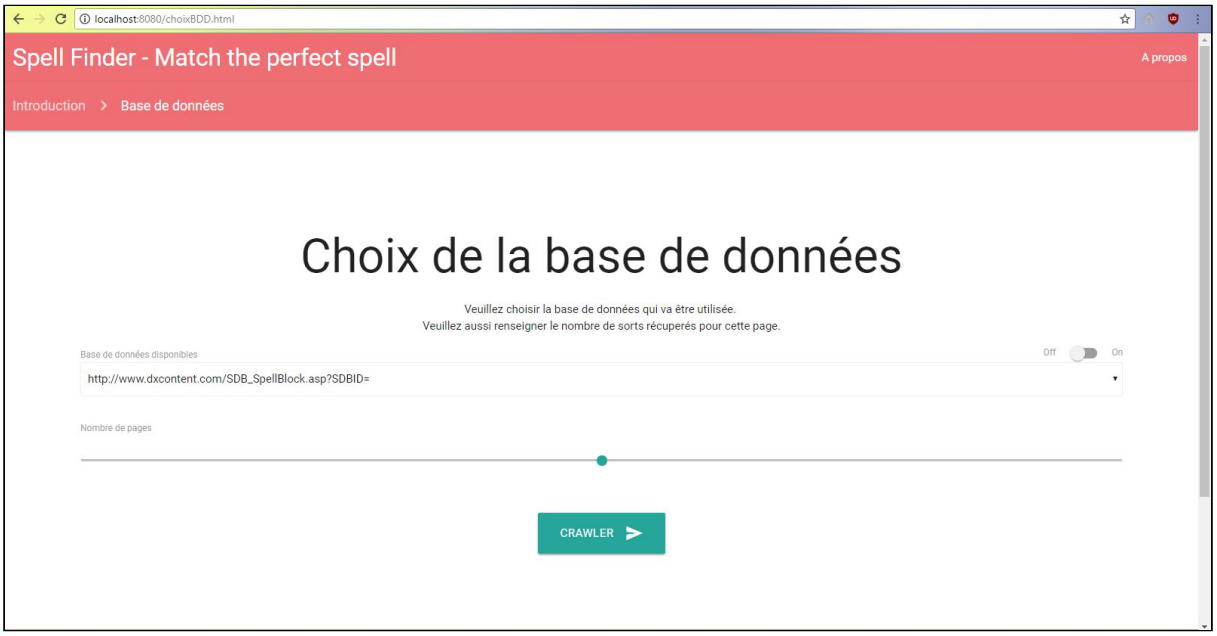
```
Iteration n° 18

{ _id: 'B',
  value: { pagerank: 0.7833688246340563, outlink_list: [ 'B', 'C' ] } }
{ _id: 'A',
  value:
   { pagerank: 1.4900124031066637,
     outlink_list: [ 'A', 'B', 'C' ] } }
{ _id: 'C',
  value: { pagerank: 1.57661877225928, outlink_list: [ 'C', 'A' ] } }
{ _id: 'D',
  value: { pagerank: 0.15000000000000002, outlink_list: [ 'D', 'C' ] } }


Iteration n° 19

{ _id: 'B',
  value: { pagerank: 0.7832552713203321, outlink_list: [ 'B', 'C' ] } }
{ _id: 'A',
  value:
   { pagerank: 1.4901259564203881,
     outlink_list: [ 'A', 'B', 'C' ] } }
{ _id: 'C',
  value: { pagerank: 1.57661877225928, outlink_list: [ 'C', 'A' ] } }
{ _id: 'D',
  value: { pagerank: 0.15000000000000002, outlink_list: [ 'D', 'C' ] } }


Iteration n° 20

{ _id: 'A',
  value:
   { pagerank: 1.4901259564203881,
     outlink_list: [ 'A', 'B', 'C' ] } }
{ _id: 'B',
  value: { pagerank: 0.7833035314786649, outlink_list: [ 'B', 'C' ] } }
{ _id: 'C',
  value: { pagerank: 1.5765705121009472, outlink_list: [ 'C', 'A' ] } }
{ _id: 'D',
  value: { pagerank: 0.15000000000000002, outlink_list: [ 'D', 'C' ] } }
End
```

Exercice 1 - Spell-Finder

Pour tester l'exercice 1 il faut, dans une premier temps, lancé le script localisé à la racine du projet : **server.js.**
Quand ce script est lancé, il suffit ensuite d'ouvrir une page web à l'adresse suivante : **localhost:8080.** La page suivante devrait être affichée :



Il suffit de cliquer sur le bouton démarrer pour continuer. La page suivante s'affichera :



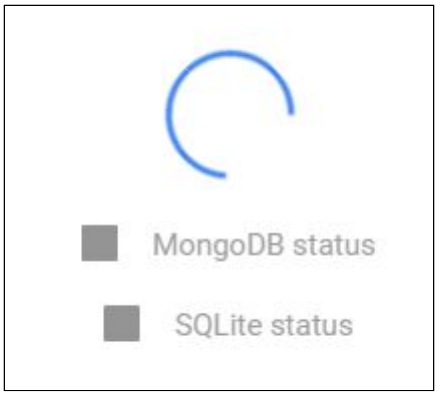
Cette page demande si l'on veut extraire les données d'une site. Pour l'instant, il n'est possible que de crawler les données du site http://www.dxcontent.com/ .

Pour crawler les données, il suffit d'activer le "*toggle button*" à droite, de choisir le nombre de pages voulues et de cliquer sur le bouton Crawler. Une fois le bouton cliqué, le crawler s'exécutera et une page de chargement sera affichée. Cette page se mettra à jour lorsqu'une base de données sera pleine.

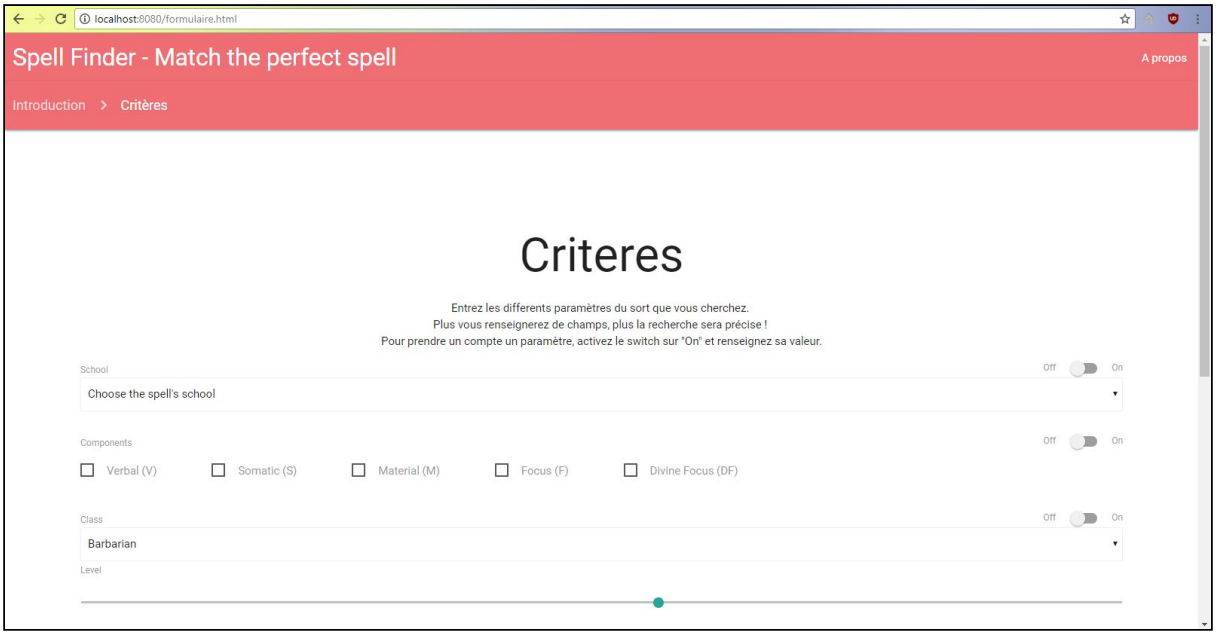
Chaque page crawlé sera converti en format JSON. Un exemple est donné ci-dessous. Il s'agit du sort "***Path Of Glory***".

```
{ name: 'Path Of Glory',
  school: 'conjuration',
  subschool: 'healing',
  descriptor: '',
  levels:
   [ { class: 'bard', level: '2' },
     { class: 'cleric', level: '2' },
     { class: 'oracle', level: '2' } ],
  CastingTime: '1 standard action',
  components: [ 'V', 'S' ],
  range: 'touch',
  duration: '1 round/level',
  SavingThrow: 'none; ',
  SpellResistance: 'no',
  description: 'You cause four 5-foot squares (one of which must be your space) to glow with dim
illumination. Starting on your next turn, as a swift action you can extend the glowing area by an
 additional four 5-foot squares; each new square must be adjacent to a square that was previously
 glowing. Allies that end their turns on a glowing square (including one who falls unconscious in
 the square) are healed of 1 point of damage.',
  effect: '' }
```


Note : La base de données MongoDB est remplie bien plus rapidement que la SQLite. Il est normal que la base de données SQLite mette plus de temps, puisqu'elle doit effectuer bien plus de *queries* que la MongoDB (à cause des liaisons multiples la table **Invoque**).



Lorsque le crawler à fini, ou dans le cas où le toggle button n'a pas été sélectionnée, on arrive directement sur le formulaire suivant :



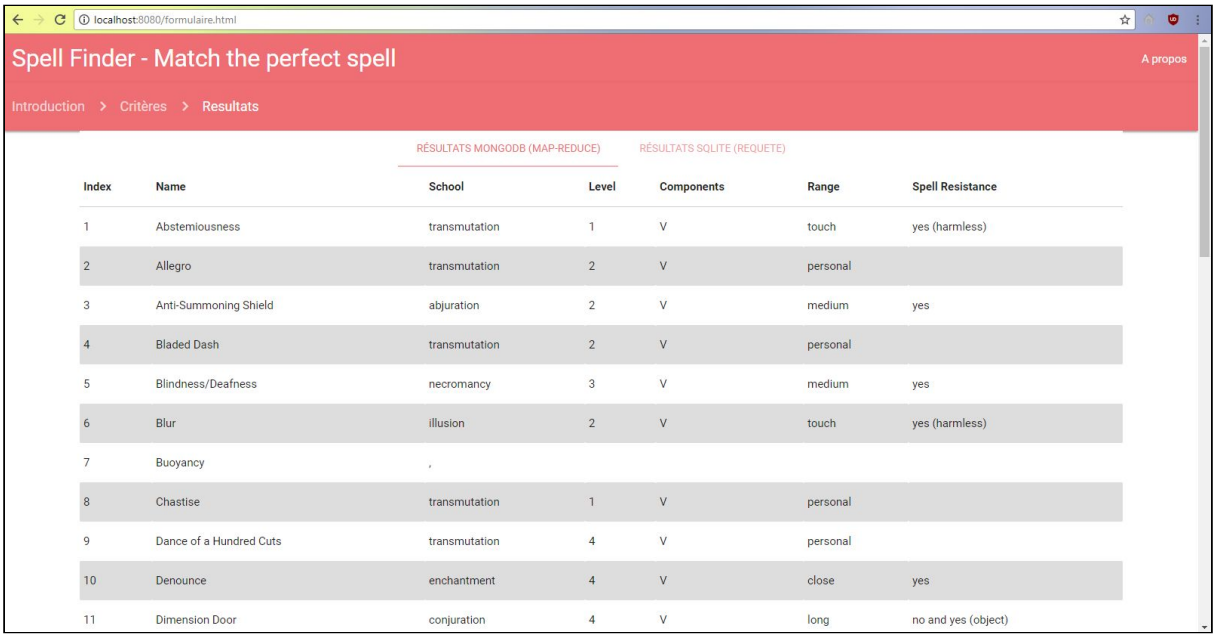
Cette page permet de saisir les valeurs pour filtrer les sorts. Pour prendre en compte une valeur, il faut activer le "*toggle button*" associé et saisir la valeur souhaitée. C'est ici que les critères vont rentrés pour sauver Pito !

Note 1 : De temps en temps, suivant les paramètres, il y a une différence entre les sorts issues de MongoDB et ceux de SQLite. Cela est dû à certaines liaisons qui ne sont pas ajoutées dans la base de données SQLite. **L'application n'est donc pas encore entièrement fonctionnelle mais pourra largement sauver Pito !**

Note 2 : A l'heure actuelle, il n'est pas possible de filtrer le niveau seul. Il faudra renseigner la classe que l'on souhaite en plus du niveau. Pito devra donc connaître sa classe (wizard ou sorcerer à priori) s'il veut avoir uniquement ses sorts disponibles de niveau 4 !

Note 3 : Certaines valeurs dans le formulaire sont inutiles. Par exemple, la range n'est pas encore liée dans l'algorithme. Il n'est pas donc possible de filtrer avec cette valeur.

Enfin, après avoir choisi les filtres, et avoir cliqué sur le bouton Chercher, une liste de sorts s'affichera. Elle devrait correspondre à celle que l'on attend



Index	Name	School	Level	Components	Range	Spell Resistance
1	Abstemiousness	transmutation	1	V	touch	yes (harmless)
2	Allegro	transmutation	2	V	personal	
3	Anti-Summoning Shield	abjuration	2	V	medium	yes
4	Bladed Dash	transmutation	2	V	personal	
5	Blindness/Deafness	necromancy	3	V	medium	yes
6	Blur	illusion	2	V	touch	yes (harmless)
7	Buoyancy	,				
8	Chastise	transmutation	1	V	personal	
9	Dance of a Hundred Cuts	transmutation	4	V	personal	
10	Denounce	enchantment	4	V	close	yes
11	Dimension Door	conjuration	4	V	long	no and yes (object)

On peut voir en scrollant l'intégralité des sorts obtenus grâce aux filtres. Il est possible de visualiser les résultats obtenus avec la requête SQLite en cliquant sur l'onglet SQLite en haut de la page.

Note 1 : En ce qui concerne Pito, s'il choisit les sorts Verbaux seulement, il n'aura pas les sorts Verbaux avec Focus par exemple ! Seul les sorts avec 1 seul component qui est 'V' seront sélectionnés.