
,

TER - Jeu des différences

**POINTEAU Gabrielle
(FANUS Ludovic)**

LAFOURCADE Mathieu



1^{ère} année de master informatique mention IASD (et GL)
Département informatique
Université des Sciences

2022 - 2023

Table des matières

Introduction	4
Notre groupe	4
Support	4
Répartition des tâches	4
Répartition au cours du temps	4
Présentation du projet	5
But	5
Jeux de mots	5
Interêt dans le monde de l'informatique	6
Choix du langage de programmation	6
HTML & CSS	6
PHP	6
JavaScript	6
1 Connexion / Inscription	9
1.1 Inscription	9
1.2 Connexion	10
1.3 Session	10
2 Accueil	12
2.1 Barre de menu	12
2.2 Carrousel	13
2.3 Affichage en fonction de la connexion	13
2.4 Autres	13
3 Création des parties	14
3.1 Entités	14
3.1.1 Lien avec jeux de mots	14
3.1.2 Base de données	15
3.1.3 Session	15
3.1.4 Affichage suivant	15
3.2 Différences	15
3.2.1 Base de données	16
3.2.2 Affichage suivant	16
4 Jouer	18
4.1 Entités	18
4.1.1 Session	18

4.2	Différences	19
4.2.1	Affichage départ	19
4.2.2	Base de données	19
4.2.3	Affichage suivant	19
4.2.4	Session	19
4.3	Calcul des points	19
4.3.1	Base de données	20
5	Parties / Classement	21
5.1	Classement	21
5.1.1	Ajout pour plus tard	21
5.2	Parties (en cours)	21
6	Compte	22
6.1	Modification	22
	Conclusion	24
	Difficultés	24
	Bénéfices	24
	Améliorations possibles	25
7	Annexes	27
7.1	checkTermeReseau.php	27
7.2	Classement	27
7.3	CreerParties.php	28
7.4	Jouer.php	29
7.5	connexion.php	29
7.6	Compte.php	29
7.7	index.php	30
7.8	Organisation	30
	Bibliographie	31

Introduction

Notre groupe

Notre groupe était composé de 2 membres :

- FANUS Ludovic
- POINTEAU Gabrielle

Notre encadrant est Monsieur LAFOURCADE Mathieu.

Gestion de groupe

Support

Au départ, nous avons plusieurs plateformes de discussion. On utilisait Discord comme moyen de communication avec un serveur de TER où l' on mettait les informations telles que les sites utiles ou encore pour se poser des questions.

Nous avons gitHub comme moyen de stockage du projet. Puis nous utilisons Trello, un site d'organisation des tâches entre nous. On utilisait également un google doc auquel notre encadrant avait accès pour pouvoir noter toutes nos réflexions sur le projet.

De plus, nous devons utiliser aussi également Overleaf pour notre rapport et sûrement un google doc pour faire les diaporamas de la présentation finale, mais que nous n'avions pas encore fixés.

Au niveau des supports, nous avons de quoi s'organiser de façon optimale.

Répartition des tâches

Cependant, la répartition des tâches s'est révélée assez compliquée, car il fallait que chacun prenne ses marques. Ensuite à l'aide de Trello nous avons réussi à se répartir les tâches équitablement (figure 7.10 page 30).

Répartition au cours du temps

De plus, il est vite apparu que l'on était un peu en retard pour pouvoir mettre le site sur internet et que de nombreux joueurs puissent l'utiliser pour avoir une étude concrète sur laquelle se baser dans notre rapport. Puis ensuite, on a réalisé le projet à partir de zéro en moins d'1 mois pour pouvoir rendre quelque chose malgré le silence d'un des membres.

En conséquence, les problèmes de dernière minute que nous avons rencontrés ne nous ont pas permis d'effectuer un bilan plus approfondi et constructif du jeu au travers des joueurs extérieurs, par leurs avis ainsi que par les données qu'ils nous auraient pu nous transmettre. Par exemple, nous aurions pu et dû analyser plus précisément les termes et les différences apportés pour améliorer notre jeu. De même, les retours d'expérience auraient probablement pu nous permettre d'envisager des améliorations, qu'elles soient techniques ou relatives au domaine du ressenti.

Présentation du projet

But

Notre projet s'intitule : **Jeu de différences**. Il consiste en but premier à récolter des données sur les différences qu'on peut trouver entre deux mots assez proches comme un "chat" et un "chien" ou une "orange" et un "pamplemousse".

On doit donc partir de deux termes et trouver les différences entre ces deux mots de telle manière à ce qu'on entre un domaine de différence par exemple un "lieu" ; le chat et le chien ne vive peut-être pas au même endroit. Puis on doit donner une valeur pour différencier à partir du domaine les deux mots, de telle sorte qu'elles soient vraies pour un des mots et fausses pour l'autre. Par exemple, un chat vit principalement plus à l'intérieur qu'un chien. On mettra donc comme domaine "lieu" et comme valeurs : "extérieur" comme étant vrai pour le chien et donc faux pour le chat et "intérieur" comme étant vrai pour le chat et faux pour le chien.

Ainsi, pour rendre ludique la récolte de données, nous devons le proposer sous forme de jeu et notamment de jeu web pour pouvoir l'avoir en ligne et le proposer facilement à notre entourage.

Cependant, comme il est difficile de créer des parties par un ordinateur en récoltant des mots au hasard, l'idée est que ce soit les utilisateurs qui créent des parties pour les faire jouer aux autres utilisateurs du site. On a donc un site web qui permet en premier lieu à jouer à des parties et qui proposera par la suite aux joueurs de créer des parties.

Par ailleurs, ce projet n'est pas un quiz, il n'y a pas de réponse juste ou de réponse fausse. On fait simplement confiance aux utilisateurs dans leurs données qu'ils rentrent en ajoutant des poids sur une relation. Plus la réponse sera considérée comme juste par les utilisateurs, plus elle aura un poids élevé. À l'inverse, plus la relation sera considérée comme fausse par les utilisateurs, plus elle aura un poids bas. Ainsi, on analysera ensuite les données pour voir si le jeu est à la fois ludique et si les données récupérées sont intéressantes. Cependant actuellement avec la situation du groupe, on n'a pas eu le temps de faire tourner le jeu dans notre entourage pour qu'ils puissent jouer et donner des données.

Jeux de mots

Nous avons une seule contrainte pour réaliser l'interface de zéro, qui était de choisir des mots présents dans la base *JeuX de Mot*. Pour cela on devait s'aider de *RESEAU DUMP* une interface dans *Jeux de Mots* qui permettent de récolter toutes les relations associées au mot demandé par type de relations et par relation d'entrée et relation de sortie. En effet, si on prend le mot "chat", on va retrouver "chat est un animal" dans les relations sortantes de "chat" tandis que "un chat roux est un chat" sera dans les relations entrantes de "chat".

Nous devons donc créer de zéro une interface ludique pour pouvoir récolter des données en respectant cette contrainte.

Interêt dans le monde de l'informatique

Ce projet rentre donc dans le cadre des jeux de réflexion. Il a été créé dans un but ludique et éducatif, afin de pouvoir mieux aborder nos notions de programmation web. C'est ce qui nous a semblé être le plus attractif et le plus intéressant parce qu'on y aborde à la fois l'aspect ludique d'un jeu mais aussi la réflexion que l'on doit avoir en tant que joueur pour trouver les différences. De plus, ce projet réunit deux domaines qui nous apprécions : les interactions avec les bases de données et la programmation web.

Choix du langage de programmation

Au départ, nous avons choisi d'utiliser les vues et le JS ¹ en particulier la bibliothèque *nuxt* pour réaliser notre projet. Mais suite au silence d'un des membres du groupe, et comme je ne connaissais vraiment pas très bien ces langages, j'ai choisi d'utiliser PHP ², HTML ³, CSS ⁴ et un peu de JS pour réaliser le projet de zéro.

HTML & CSS

Tout d'abord, nous avons dû utiliser la base avec de l'HTML et le CSS, nous avons opté pour une mise en page simple (header, body, et un peu de footer). Notre site web reprend le principe de beaucoup de sites avec un header permettant d'afficher à quel endroit nous nous situons sur le site et avec le nom de l'utilisateur connecté pour la page d'accueil. Il y a aussi le body où nous avons accès à toutes les informations de notre site (les pages PHP et le carrousel pour l'accueil), et ensuite le footer où nous pouvons avoir accès aux contacts ou aides relatives au site web, toujours sur la page d'accueil.

Quant au CSS, nous avons opté pour de la mise en forme classique avec une barre de menu où l'on peut accéder aux différentes pages du site.

PHP

Ensuite, nous avons de multiples scripts PHP permettant le bon fonctionnement de notre site web ; ces scripts nous permettent d'interagir avec notre base de données en insérant, sélectionnant, en mettant à jour les données selon ce dont on a besoin. Notamment pour la création d'une partie, on va plutôt insérer des parties. Alors que pour modifier son compte, on va utiliser la mise à jour de la page.

Puis nous utilisons en permanence la sélection des données dans la base pour afficher la liste des différences ou faire des actions en fonction de la base comme calculer le nombre de points gagnés en fonction des réponses attendues par le créateur d'une partie.

JavaScript

Finalement le dernier langage que nous avons utilisé est le JavaScript, qui est un langage de script, qui fonctionne avec HTML et CSS et qui nous a permis d'obtenir un côté plus dyna-

-
1. JavaScript
 2. PHP Hypertext Preprocessor
 3. Hypertext Markup Language
 4. Cascading Style Sheets

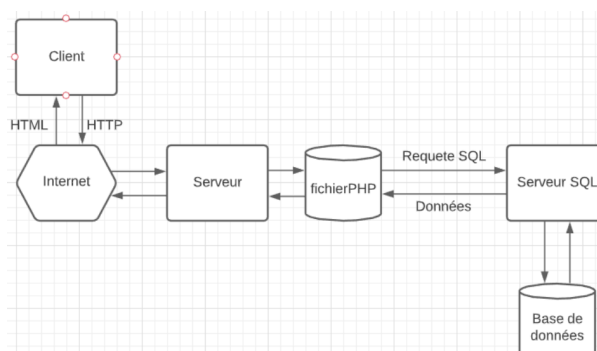
mique dans nos scripts PHP. On l'utilise pour l'affichage du mot de passe dans l'inscription et la connexion, ou encore pour l'appel des scripts PHP lors du remplissage des champs de saisie que nous verrons un peu plus loin.

Architecture du site

La figure 1 à la page 7 est un schéma qui représente l'architecture d'un site internet d'un point de vue client/serveur. En effet, le client est celui qui va vouloir venir sur notre site web et y interagir. Si l'on passe notre site en public (donc en online) le client interagira via des requêtes http avec le navigateur permettant donc d'accéder à notre site web. En retour Internet (WWW⁵) nous fournira, suite à ces requêtes une page HTML provenant de notre site.

Ensuite, entre Internet et le serveur (on part du principe où nous, les administrateurs du site sommes directement le serveur) les interactions sont les mêmes qu'entre le client et l'internet, le serveur va envoyer une réponse http à Internet sous forme de page HTML tandis que Internet va envoyer la requête http envoyée par le client. Voici donc comment fonctionne ; le côté client de notre site (s'il était en ligne).

FIGURE 1 – Schéma de l'architecture

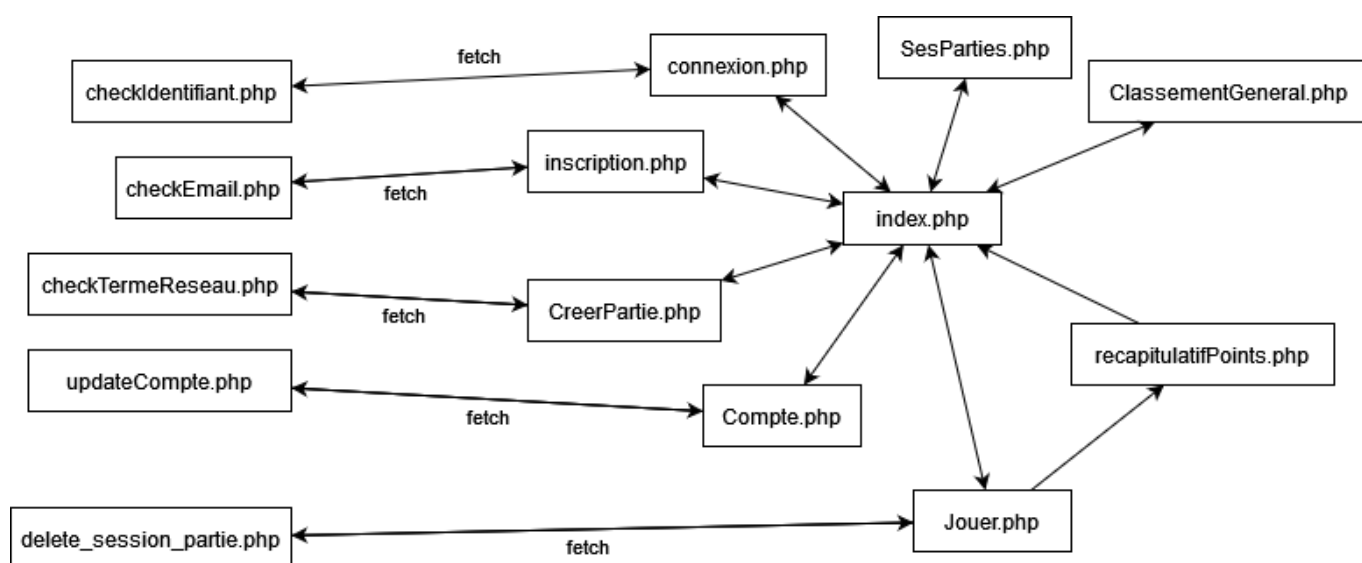


Maintenant, voyons de plus près comment marche le côté serveur. Le serveur a donc besoin de fichier PHP, car c'est indispensable pour faire marcher et interagir notre site web avec des clients. Reliés à ça, les fichiers PHP (reliés au serveur) serviront à envoyer des requêtes SQL au serveur SQL qui va les renvoyer à nos fichiers PHP sous forme de données. Par exemple, quand nous nous inscrivons sur le site, dans le fichier PHP prévu à cet effet, on enverra une requête SQL permettant d'insérer l'identifiant et le mot de passe dans des tables SQL. Ainsi, le serveur SQL renverra ces données aux fichiers pour que les clients puissent l'utiliser. Pour finir, le serveur Sql, obligatoire pour faire fonctionner un serveur PHP, est relié lui-même au SGBD⁶ permettant d'envoyer directement les requêtes SQL du type INSERT INTO, par exemple, dans nos bases de données.

5. World Wide Web

6. système de gestion de Base de Données

FIGURE 2 – Schéma du site



On peut alors voir sur la figure 2 à la page 8 l'architecture du site, c'est à dire les différentes pages utilisées dans le site sans les fichiers CSS et *Toasts.js*.

Base de données

Notre base de données est composée actuellement de 6 tables que nous stockons sur *php-MyAdmin* :

- **utilisateur** : Elle regroupe toutes les informations d'un utilisateur pour le bon déroulement du jeu ; elle a donc un score utilisateur ainsi que l'indication si c'est un administrateur du site ou non.
- **parties** : Cette table contient les informations nécessaires à une partie composée de deux entités définies par leur identifiant, l'identifiant de l'utilisateur qui crée la partie ainsi que le nombre de joueurs qui a joué à cette partie. L'argument *is_ajoute* sert pour récupérer les parties créées par les utilisateurs et non les administrateurs pour avoir une petite base au départ pour les joueurs.
- **entites** : Elle est composée d'un nom, d'une catégorie (non utilisée actuellement) et de *is_ajoute* comme pour la partie.
- **différences** : les différences sont définies par un type, une valeur et un argument qui permet de savoir dans quel champ la différence courante a été saisie. L'utilisateur rentre par exemple « chocolat » dans le champ du terme 1 alors *is_true_entite1* sera vrai (1). Sinon il sera faux (0)
- **différence_partie** : Cette table permet de faire le lien entre les différences et les parties : Ainsi on a l'id d'une différence associée à l'id d'une partie qui est associée à l'id d'un utilisateur pour savoir qui est-ce qui a rempli cette différence.
- **partie_jouer** : Pour finir, nous avons la table qui fait le lien entre une partie et un joueur. Un utilisateur va jouer à une partie et aura un certain score, le nombre de fois qu'il a déjà joué à la partie ainsi que le pourcentage de réponse « justes » qui correspond aux réponses attendues par le créateur de la partie courante. On a donc aussi l'id d'une partie et l'id d'un utilisateur.

Chapitre 1

Connexion / Inscription

1.1 Inscription

Au départ les utilisateurs arrivent sur la page d'accueil (*index.php*) qui leur montre la possibilité de se connecter, de s'inscrire ou de jouer en tant qu'invité (en cours, pas encore disponible). Pour une première utilisation, le joueur pourra donc soit jouer en tant qu'invité, soit s'inscrire.

Les fonctionnalités que nous évoquons dans cette première partie sont basiques à un site web, cependant toutes les vérifications ont pris du temps à être mis en place car la gestion d'appel de fichier PHP dans une fonction JS était une nouveauté dans ce projet.

Une fois sur la page d'inscription (*inscription.php*), il y a les champs de saisie habituel lors d'une inscription, comme rentrer un nom, un prénom, un âge etc...

Il y a également plusieurs vérifications ou fonctionnalités lors de la saisie des informations :

- **Email** : on vérifie, lorsqu'un utilisateur entre l'email, qu'il n'est pas déjà existant dans la base. Pour cela, on fait appel au fichier *./gestion_PHP/CHECK/checkEmail.php* pour faire une requête PHP dans une fonction JS appelée lorsqu'on entre quelque chose dans le champ de saisie. Ce script PHP va sélectionner les lignes qui correspondent à l'email rentré et regardé s'il existe une ligne qui correspond. Si c'est le cas et que l'email existe déjà, un message d'erreur s'affiche sur la page. Si ça n'est pas le cas, alors on peut continuer.

Adresse Mail :

gaby1@gmail.com

E-mail déjà utilisé.

FIGURE 1.1 – Email déjà utilisé

- **Confirmation de mot de passe** : on vérifie que les mots de passe entrés à la fois sur le mot de passe en lui-même et sa vérification sont les mêmes. Pour cela on utilise également une fonction JS pour afficher ou non un message d'erreur lorsque l'on entre des mots de passe.

Mot de Passe :

123456



Confirmation du Mot de Passe :

1234



Les mots de passe ne sont pas égaux.

FIGURE 1.2 – Mot de passe différents

- **Visualisation de mot de passe :** En utilisant une fonction JS, il existe un bouton pour visualiser les mots de passe entrés et les cacher également. On joue sur le type d'input pour cela, en changeant le visuel de l'icône à l'aide d'un script SVG ¹.

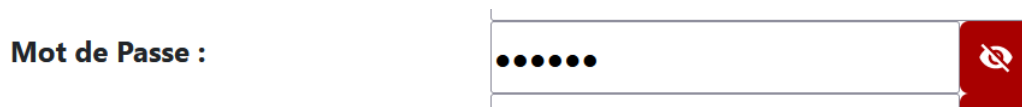


FIGURE 1.3 – Mot de passe non visible

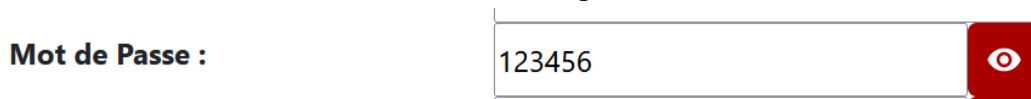


FIGURE 1.4 – Mot de passe visible

Il y a également sur la même page un bouton pour aller directement à la connexion si jamais l'utilisateur a déjà un compte.

1.2 Connexion

Si un utilisateur est déjà inscrit sur le site, il n'a plus qu'à se connecter (*connexion.php*). L'utilisateur met donc l'email qu'il a choisi lors de son inscription ainsi que son mot de passe, qu'il peut également afficher ou non.

Pour la vérification des informations, on utilise également un script JS pour pouvoir l'appliquer lorsqu'on entre un élément dans le champ de saisie pour appeler le script PHP *./gestion_PHP/CHECK/checkIdentifiant.php* (figure 7.7 page 29) pour pouvoir vérifier dans la base si la ligne existe et si ce sont les bons email et mot de passe.



FIGURE 1.5 – Incorrect



FIGURE 1.6 – Correct

1.3 Session

Une fois connecté ou inscrit avant d'aller sur la page d'accueil (*index.php*), on fait une recherche de l'utilisateur dans la base pour récupérer toutes ses informations. Celles-ci seront mises dans une variable `$_SESSION['utilisateur']` pour pouvoir les faire passer entre les pages sans devoir faire une recherche à chaque changement de page sur le site. La variable contient

1. Scalable Vector Graphics

toutes les informations de la base sur l'utilisateur de la même façon qu'elles sont stockées dans cette base.

Lors de la réalisation du site, nous avons un problème de session : celle-ci marchait très bien en local avec WampServer et le service Apache, mais dès qu'on les hébergeait sur *000webhost*, les sessions ne fonctionnaient pas. Par conséquent, nous avons ajouté un fichier pour résoudre ce problème : *.htaccess*.

Chapitre 2

Accueil

2.1 Barre de menu

Une barre de menus est présente sur chaque page principale du site pour pouvoir naviguer facilement entre les onglets. Nous allons rapidement expliquer chaque onglet que nous détaillerons ensuite plus bas :



FIGURE 2.1 – Barre de menus

Accueil : C'est le premier onglet que l'on voit dès que l'on rentre dans le site et l'endroit où l'on peut se connecter, s'inscrire ou jouer en tant qu'invité (en cours). Une fois connecté on a accès aux autres onglets. Par contre effectivement, si jamais on veut accéder aux autres onglets sans être connecté, cela nous amène à la page de connexion avant de pouvoir aller sur l'onglet sélectionné.

Creer une partie : Sur cet onglet, c'est l'endroit où l'on crée une partie pour qu'elle soit accessible à tous par la suite. On remplit donc les deux champs de terme, puis ensuite on choisit un type de différences et ses valeurs correspondantes pour chaque terme.

Jouer : C'est l'onglet où on joue au jeu en tant que tel ; on nous propose deux termes et on doit trouver des différences correspondantes.

Classement : Cet onglet permet de visualiser où l'on se trouve par rapport aux autres joueurs.

Parties : Cet onglet permet de visualiser l'ensemble de ses parties (en cours).

Compte : Cet onglet permet de visualiser les informations de l'utilisateur courant.

Contact : Ce n'est pas un onglet en tant que tel, mais cela ouvre l'application de messagerie sur votre appareil d'utilisation pour pouvoir envoyer un mail à l'adresse suivante : *gabrielle-pointeau.jeuxdedifferences@hotmail.com*

Icon de déconnexion : Il y a également à la fin de la barre de menu, un bouton noir pour pouvoir se déconnecter : Cela va supprimer la session puis la redémarrer et nous remettre sur la page d'accueil sans être connecté.

2.2 Carrousel

On utilise pour l’affichage sur l’accueil, un carrousel de mots. En effet, On va chercher tous les mots que les utilisateurs ont mis pour créer des parties dans le jeu et on va les faire défiler pour avoir une animation sur la page d’accueil.

Ainsi, pour chaque mot d’une liste, on ajoute un *carrousel-item* pour l’afficher ensuite en défilement. Comme on peut le voir avec le code 7.9 à la page 30.

2.3 Affichage en fonction de la connexion

Lorsqu’un utilisateur n’est pas encore connecté, on a un affichage qui peut nous montrer que personne n’est connecté. Il y a aussi un bandeau d’affichage de 3 boutons de connexion, inscription et invité. Cependant, lorsque l’utilisateur s’inscrit ou se connecte, les boutons ne sont plus affichés et on marque le nom et le prénom de l’utilisateur qui est connecté.

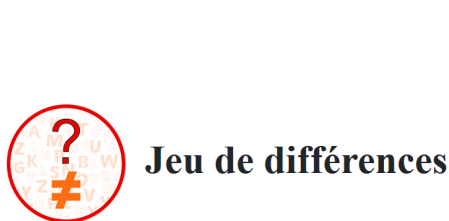


FIGURE 2.2 – Connecté



FIGURE 2.3 – Non connecté

2.4 Autres

On a également en bas de page, deux boutons qui permettent de voir son compte, ou de contacter l’administrateur du site. Il y a également une petite description du projet qu’on devait réaliser pour les utilisateurs ainsi que le logo créé pour ce jeu.

Chapitre 3

Création des parties

3.1 Entités

L'idée première lorsqu'on arrive sur l'onglet de création de partie, est de rentrer deux termes à comparer. On peut mettre deux mots que l'on veut cependant, ils doivent être présents dans la base de données, jeux de mots. Pour cela, on doit vérifier à chaque entrée d'un caractère, et ce jusqu'à ce que l'utilisateur ait fini de remplir le champ, que le mot existe dans la base.

3.1.1 Lien avec jeux de mots

On utilise donc une fonction JS pour pouvoir appeler un fichier PHP qui va réaliser le lien avec la base */gestion_PHP/CHECK/checkTermeReseau.php*. On donne à ce fichier le mot du champ saisi par l'utilisateur pour réaliser une requête CURL ¹ vers le site « *jeuxdemots.org* » pour vérifier si le mot existe dans leur base de données.

Pour savoir si le mot existe dans la base on utilise *resseauDUMP* qui nous indique si la balise *class="jdm-warning"* est présente dans la recherche, si cette balise est présente alors le mot n'existe pas. Puis, on nous renvoie un texte pour nous dire si le terme existe ou non dans la base en fonction. En revenant au script JS, on affiche un message d'erreur ou non selon la réponse. On peut le voir sur la figure 3.1 et 3.2 à la page 15

CURL est une bibliothèque logicielle utilisée pour effectuer des requêtes et des transferts de données via différents protocoles tels que HTTP, HTTPS... cURL permet aussi d'effectuer des opérations de transfert de données entre un client et un serveur, notamment l'envoi et la réception de requêtes HTTP. Cela permet au serveur d'interagir avec des API, et notamment d'obtenir des données à partir de sources externes, notre base jeux de mots, et donc de récupérer les pages web correspondantes avec *resseauDUMP*.

Par contre si jamais on va trop vite, l'affichage peut bugger et ne plus s'afficher alors que le mot est évidemment pas présent dans la base. Ainsi, n'hésitez pas à rajouter une ou deux lettres pour bien faire apparaître le mot.

1. Client URL

Premier terme :

Le mot n'existe pas dans la base !

FIGURE 3.1 – Mot n'existe pas

Premier terme :

FIGURE 3.2 – Mot existe

3.1.2 Base de données

Une fois que l'on n'a pas d'erreur et que les mots existent bel et bien dans la base, on a juste à appuyer le bouton de validation. Cela va remplir la base de données en ajoutant les deux entités dans la table *entites*.

De plus, il sera créé une partie qui va contenir l'identifiant des deux termes, l'identifiant du créateur de la partie donc l'utilisateur courant, et deux autres choses présentes dans la base que nous avons vue plus haut.

3.1.3 Session

Par ailleurs, comme on va recharger la page en appuyant sur chaque bouton, on a décidé de stocker la partie courante créée dans le tableau de session : `$_SESSION['partieCouranteCreer']` qui contiendra la structure suivante :

- premierTerme
- secondTerme
- idPremierTerme
- idSecondTerme
- idPartie

3.1.4 Affichage suivant

Puis, on affichera une liste déroulante ainsi que deux nouveaux champs de saisie pour remplir les différences de la partie qui seront les réponses attendues par le créateur du jeu.

3.2 Différences

Pour remplir les différences de la partie, il y a une liste déroulante qui regroupe plusieurs types de différences comme la couleur, la caractéristique, le lieu. C'est sur lesquels seront comparés les deux termes choisis auparavant.

Puis, il y a deux champs de saisies, qui permettent d'affiner la différence en mettant une valeur sur laquelle l'opposition est faite.

Par exemple, si l'on prend les deux termes entre une orange et un pamplemousse, on peut prendre comme type de différence leur couleur. Et on mettra dans pamplemousse la valeur rose et dans orange la valeur orange.

3.2.1 Base de données

Pour afficher les types de différences, on utilise celles présentes dans notre base de données. C'est-à-dire qu'on utilise la requête sur la figure 7.5 à la page 28 pour pouvoir sélectionner l'énumération présente dans l'attribut *type_diff* de la table *différences*. Ainsi, c'est à partir de cette énumération que l'on va créer les éléments présents dans la liste déroulante.

Il n'est pas possible d'en rajouter sur le moment, mais si l'utilisateur veut absolument avoir un certain type de différence, il peut envoyer un mail pour que l'administrateur puisse l'ajouter à la liste. Ainsi, l'utilisateur pourra le modifier dans ses parties.

Ensuite lorsqu'on appuie sur le bouton de validation de la différence rentrée, on insère dans la table *différences* les deux valeurs de différences ainsi que le type de différence. Il y a également l'argument de validité de la différence pour l'entité 1 qui sera vraie si jamais c'est bien la différence remplie sur le champ 1.

On remplit également la table *différence_partie* qui fait le lien entre la différence et la partie, ainsi que l'utilisateur qui l'a rempli. Ici comme c'est la création de la partie, l'utilisateur est le créateur de la partie.

3.2.2 Affichage suivant

Ensuite, on va sélectionner les différences que l'utilisateur a remplies dans la base pour afficher la liste des différences en rouge si la valeur de différence est fausse pour le terme, ou en vert si la valeur de différence est vraie selon la colonne dans laquelle l'affichage se situe.

Il sera possible également de supprimer la différence remplie dans la liste grâce au bouton déjà affiché à côté de chaque différence mais c'est encore en cours. Si jamais on supprime d'un côté en rouge la différence, cela supprimera la différence en vert de l'autre côté.

On affiche à nouveau les champs pour remplir une autre différence et on répète l'opération autant de fois qu'on a besoin. Puis on peut appuyer sur le bouton de validation de la partie tout en bas pour terminer la création de la partie.

Après avoir appuyé sur le bouton, on supprime la variable de session de partie courante créée et on affiche une pop-up en haut à droite pour indiquer que l'on a bien créé la partie.

Toast :

Cette pop-up est réalisé par un Toast à l'aide des fichiers *Toast.js* et *style_toast.css* ce qui, en fonction du type de pop-up que l'on veut, affiche une validation, une erreur, une information ou autre. Ici pour la création de partie ce sera un pop-up vert pour avoir réussi à valider la partie.

Pour pouvoir l'afficher, on ajoute un argument `$_GET['partie_enregistree']` qui nous indique qu'on a appuyé sur le bouton « valider la partie ». On peut afficher le Toast à l'aide de la fonction `supprimerParametreGET(nomParametre)` qui va enlever le paramètre `nomParametre` de l'URL, ainsi que `afficherToast()` qui utilise la classe `Toast` pour pouvoir construire le toast de notre choix en mettant les paramètres voulus.

On voit alors sur le code de la figure 7.4 à la page 28 l'utilisation du constructeur de `Toast` pour pouvoir afficher le pop-up.

La classe `Toast` dans le fichier *Toast.js* permet de spécifier où l'on veut positionner le pop-up, de quel type, avec quel couleur, sa durée, sa taille etc...

Une fois le tout réalisé, on retourne à la page de création de partie pour recréer une autre partie si on le souhaite, ou changer d'onglet grâce à la barre de menu.

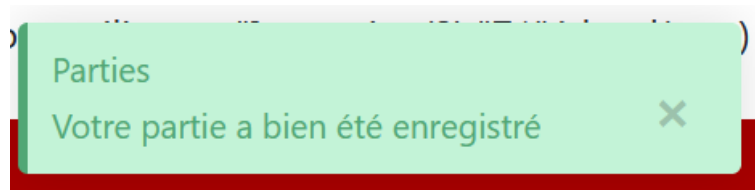


FIGURE 3.3 – Toast lors de la création

Chapitre 4

Jouer

Comme nous l'avons vu au départ, le but du jeu est de trouver un maximum de différences entre deux termes et notamment celles attendues par le créateur, qui nous font gagner le plus de points.

4.1 Entités

Ainsi, on nous propose deux termes qui font partie d'une partie tirée de façon aléatoire par une requête SQL faite exprès pour cela. Ainsi, on va tirer au hasard une partie si la variable session correspondante à la partie jouée n'existe pas. La requête est donc sur la figure 7.6 à la page 29. On voit également le début de l'enregistrement de la variable `$_SESSION['partieCouranteJoue']` être enregistré.

4.1.1 Session

Pour garder la même partie à chaque validation d'une différence, et puisque l'on recharge la page, on a décidé, comme lors de la création d'une partie, d'utiliser une variable de session pour stocker la partie courante jouée : `$_SESSION['partieCouranteJoue']`. Cette variable va stocker :

- `idPartie`
- `idUserCourant` : l'utilisateur qui joue
- `idUserCreate` : l'utilisateur qui a créé la partie
- `idEntite1`
- `idEntite2`
- `nomEntite1`
- `nomEntite2`

Au départ, on ne stocke que les informations de base pour pouvoir ensuite en récupérer les différences attendues par le créateur du jeu. Pour pouvoir les stocker dans la variable, on a :

- `differencePartie`

Cette information va nous permettre de calculer les points gagnés par l'utilisateur en comparant ses réponses à celles du créateur de la partie.

4.2 Différences

4.2.1 Affichage départ

De la même manière que pour la création de la partie, on a des champs de saisie afin de pouvoir remplir les différences que l'utilisateur peut trouver.

Cependant, il y a la liste des différences au sujet desquelles l'utilisateur a déjà répondu si jamais il a déjà joué à cette même partie, afin qu'il puisse tirer parti de ce qu'il a fait et continuer d'augmenter son pourcentage de réponses attendues. A contrario, si l'utilisateur n'a pas encore joué à cette partie, rien ne s'affiche dans les deux cases de la liste.

4.2.2 Base de données

Lorsque l'utilisateur remplit les champs et valide une différence trouvée, on stocke sa réponse de la même façon que lorsque l'on crée une partie, mais dans la table *difference_partie* on met l'utilisateur courant dans l'attribut *idUtilisateur* pour montrer que c'est une différence remplie par l'utilisateur qui joue et la distinguer de la réponse attendue.

4.2.3 Affichage suivant

De la même manière, pour l'affichage, on va sélectionner les différences remplies dans la base par l'utilisateur courant, pour afficher la liste des différences; en rouge si la valeur de différence est fausse pour le terme ou en vert si la valeur de différence est vraie, selon la colonne dans laquelle l'affichage se situe.

Il sera également possible de supprimer la différence remplie dans la liste grâce au bouton déjà affiché à côté de chaque différence, mais c'est encore en cours. Si jamais on supprime d'un côté en rouge la différence, cela supprimera aussi la différence en vert de l'autre côté.

On affiche aussi à nouveau les champs pour remplir une autre différence, et on répète ensuite l'opération autant de fois que nécessaire. Enfin, on peut appuyer sur le bouton de validation de la partie tout en bas, pour terminer la création de la partie.

Après avoir appuyé sur le bouton en question, on supprime la variable de session de la partie courante créée et on affiche un popup en haut à droite pour indiquer qu'on a bien créé la partie.

4.2.4 Session

A chaque recherche des différences remplies par l'utilisateur qui joue la partie on met à jour la variable session `$_SESSION['partieCouranteJoue']` :

— `differenceJoueur`

Cela permettra d'afficher plus facilement les différences et de les comparer dans l'onglet suivant.

4.3 Calcul des points

Lorsque l'utilisateur a fini cette partie, il la valide avec le bouton en bas de la page. Cela nous amène à une nouvelle page web *recapitulatifPoints.php* où l'on va indiquer à l'utilisateur qu'est ce qu'il a gagné comme points et à quel pourcentage de réponses attendues il en est.

Le pourcentage est calculé en comparant les valeurs de différences du joueur avec celles du créateur pour voir si il y trouve des différences identiques (jusqu' aux accents près). Si c'est le cas, il gagne alors des points en fonction du pourcentage de réponses attendues.

De plus, il peut aussi gagner quelques points sur des différences qu'il a données en plus. Cependant ces différences ne seront pas réellement vérifiées. On mettra un certain poids (à définir ensuite) sur la différence pour la valider ou non en faisant +1 à chaque fois qu'il est mentionné en tant que positif pour un terme et -1 à chaque fois qu'il est mentionné en tant que négatif (également pour un terme). Cependant, cette fonctionnalité concernant le poids n'a pas encore été mise en place, au vu du temps dont nous disposons sur la fin.

4.3.1 Base de données

Au niveau de la base de données, on fera des mises à jour des points acquis par le du joueur sur son score total du site. De plus, on mettra également à jour le score de la partie jouée par l'utilisateur. Par ailleurs, on ajoutera également 10% des points du joueur au créateur de la partie.

Les mises à jours concerneront également le nombre de joueurs qui auront participé à la partie ou le nombre de fois qu'un joueur aura joué à une partie.

Une fois la fonctionnalité mise en place, on pourra alors également mettre à jour le poids des différences dans la base.

Pour cela, on doit d'abord sélectionner le nombre courant déjà présent dans la base de données, comme le score d'un utilisateur, pour ensuite ajouter le résultat obtenu et mettre ainsi à jour la base.

Chapitre 5

Parties / Classement

5.1 Classement

L'idée de cet onglet est d'afficher le classement des joueurs en totalité. Pour cela, on sélectionne tous les utilisateurs dans la base de données, par ordre décroissant en fonction du score de chacun (figure 7.3 page 27).

Il seront donc triés et on n'aura plus qu'à les afficher à l'écran. On affiche alors tous les utilisateurs par leur nom et prénom et leur score. Pour différencier le joueur courant, on a mis un fond orange pour qu'il voit directement où il se situe dans le classement.

5.1.1 Ajout pour plus tard

L'idée est d'ajouter un autre classement par partie, afin de voir qui est le meilleur dans quelle partie. L'utilisateur ainsi peut voir le classement des joueurs d'une de ses parties, et un utilisateur peut voir son classement dans les parties qu'il a déjà jouées.

5.2 Parties (en cours)

Cet onglet permet de visualiser les différentes parties que l'utilisateur a créées en affichant les entités de chacune. L'idée est d'ajouter la possibilité d'appuyer sur chaque partie et d'afficher les différences associées ensuite dans un autre onglet, ou alors en faisant un défilement de la card.

Chapitre 6

Compte

En premier lieu, on peut voir toutes les informations de l'utilisateur (celles qu'il a remplies lors de son inscription) pour pouvoir visualiser son compte.

6.1 Modification

Chaque élément est modifiable, sauf tout ce qui est relatif au score de l'utilisateur. Pour cela on utilise le bouton présent à côté de chaque champ pour remplacer le mot courant par un champ de saisie pour pouvoir changer les informations.

Pour valider la modification on appuie à nouveau sur le bouton de modification. Cela changera les informations de l'utilisateur dans la base, ainsi qu' à l'écran et sur les variables de la session de l'utilisateur courant.

Au niveau du mot de passe : Lorsque l'on appuie sur le bouton de modification, le mot de passe est affiché dans le champ de saisie. Et quand on valide la différence on rend le mod de passe invisible.

Pour la modification des éléments et le lien avec la base, on utilise une fonction JS pour appeler un script PHP `/gestion_PHP/UPDATE/updateCompte.php` qui va changer tel ou tel élément de l'utilisateur dans la base de données, en fonction de l'argument qui est mis.

On peut voir sur le code 7.8 à la page 29, un exemple d'utilisation de fetch dans la fonction JS pour mettre à jour l'âge de l'utilisateur.


On aimerait afficher une pop-up à l'aide de Toast pour indiquer à l'utilisateur que la modification a bien été faite dans la base de données ou non si ce n'est pas le cas.

On voit donc sur la figure 6.1 à la page 23 les modification sur le compte sur le site, et sur la figure 6.2 à la page 22 l'évolution sur la base de données.




<input type="checkbox"/>	 Éditer  Copier  Supprimer	1	Gabrielle	POINTEAU	2002-08-05	20	gaby1@gmail.com	123456	0
<input type="checkbox"/>	 Éditer  Copier  Supprimer	1	Gaby	POINTEAU	2002-08-05	20	gaby1@gmail.com	123456	0

FIGURE 6.2 – Modification dans la base de données

On a également ajouté une pop-up de mise à jour des données qui nous indique que les données ont bien été modifier dans notre base ou pas. Pour cela on utilise un Toast comme pour la création d'une partie.

Informations :NOM DE FAMILLE : POINTEAU PRENOM : AGE : 20 DATE DE NAISSANCE : 2002-08-05 

SCORE : 71

Identifications :ADRESSE MAIL : gaby1@gmail.com MOT DE PASSE : ***** **Informations :**NOM DE FAMILLE : POINTEAU PRENOM : Gaby AGE : 20 DATE DE NAISSANCE : 2002-08-05 

SCORE : 71



Identifications :ADRESSE MAIL : gaby1@gmail.com MOT DE PASSE : ***** 

FIGURE 6.1 – Modification du prénom

Conclusion

Difficultés

— Gestion de groupe :

Au niveau de la gestion du groupe cela a été très compliqué de travailler ensemble : Nous n'avions pas beaucoup d'idées communes, et lorsqu'on gérait chacun de notre côté nos parties, nous n'avions aucune communication pour savoir où en était l'autre.

De plus, nous n'avions pas du tout la même façon de travailler, ce qui a rendu le projet assez chaotique et l'a surtout retardé. Par exemple, nous n'avions pas du tout fini une base du jeu, à un peu plus d'un mois de la date du rendu, alors que l'idée première était d'avoir un code pour pouvoir le mettre en ligne et faire jouer un maximum de personnes, afin d'avoir un avis sur le jeu, au niveau de la perception, de l'amusement et de l'intérêt du jeu auprès des utilisateurs.

Par ailleurs, comme à l'origine on était dans un langage de programmation qui n'était connu que d'un seul des deux membres du groupe, cela a été compliqué à suivre lorsque la communication a été rompue. Cependant, et même si les langages de programmations sont assez compréhensibles, cela devenait compliqué de passer à un projet effectué seule en connaissant assez le langage de programmation pour en faire un projet abouti.

C'est la raison pour laquelle il m'est apparu préférable de recommencer le projet dans un langage de programmation plus connu suite à l'abandon de l'un des membres. Ainsi, en moins d'un mois on (j') a(i) réussi à avoir une base convenable, ce que nous n'avions pas réussi à faire en quatre mois principalement en raison du langage choisi à l'origine.

— Session :

Comme dit plus haut, au départ une fois que l'on a mis le projet sur un site hébergeur, il y a eu un problème de session que l'on a résolu en utilisant un fichier `.htaccess` qui contient une ligne de code permettant d'activer la mise en mémoire tampon de sortie (output buffering) dans PHP. Cela a permis de stocker les informations avant de les envoyer au serveur, et donne ainsi la possibilité de faire marcher les variables de session.

Bénéfices

— Acquis

Tout cela nous a donc permis de nous améliorer en langage du web, et de mieux se familiariser avec le CSS pour un affichage optimal.

De plus, l'utilisation des fichiers PHP dans une fonction JS était nouvelle pour moi, ce qui m'a permis de découvrir cet élément et de pouvoir répondre à plusieurs problèmes.

Par ailleurs, l'utilisation d'une requête CURL pour pouvoir accéder à des pages web était aussi nouvelle, ce qui m'a permis d'en apprendre encore davantage sur ces langages de programmation.

— **Gestions/Organisation**

Au niveau de l'organisation, le fait de faire un groupe à deux où l'on ne se connaissaient pas du tout, nous a mis face aux confrontations qui pouvaient exister entre deux personnes qui n'ont pas du tout le même point de vue sur quelque chose. Ainsi, cela nous a appris à nous adapter l'un à l'autre pour répondre aux besoins du projet.

— **Site internet**

De plus, comme nous avons mis le projet sur internet, cela nous a permis de nous familiariser avec l'hébergeur 000webhost pour gérer le site. Ainsi, c'est satisfaisant qu'un projet soit sur internet à la vue de tous pour pouvoir le montrer à plusieurs personnes de notre entourage sans devoir envoyer le code entier.

Améliorations possibles

— **Parties**

En effet, nous avons vu que l'idée était d'afficher les parties en details en soulignant les différences de la partie sélectionnée à l'aide de la liste qui déroule.

Parmi les améliorations possibles, on pourrait également afficher les parties auxquelles on a déjà joué ainsi que les différences remplies et les résultats obtenus pour chaque partie.

— **Compte**

Quant à laisser la possibilité à un utilisateur de pouvoir supprimer son compte : L'idée serait simplement de voir que l'utilisateur n'existe plus mais que son enregistrement existe toujours tout en étant devenu inactif. Il faudrait alors ajouter un état du compte pour savoir s'il est actif ou non, ou supprimé. Cela permettrait de garder toutes les parties pour les autres joueurs, sans que l'utilisateur concerné ne gagne dorénavant de points.

— **Classement**

Comme nous l'avons vu plus haut, une autre idée serait d'ajouter des classements par parties jouées et par parties créées par les utilisateurs pour qu'ils puissent voir leur place et aussi qui est sur le « podium ».

— **Système de point**

Deplus, on pourrait ajouter au système de points, des récompenses par des badges lorsque des paliers sont atteints ; par exemple, pour chaque joueur arrivé à 100 points ou 50 parties jouées, ou alors 10 parties créés. Cela inciterait certainement les joueurs à collectionner les badges et donc créer une attraction supplémentaire par rapport au jeu.

On pourrait alors ajouter des animations indiquant ces badges.

— **Minuteur**

Ajouter un minuteur lorsqu'on cherche des différences pourrait être également intéressant, car si le joueur a tout son temps il pourrait rentrer une centaine de différences.

— Chat

L'idée aussi sera de réaliser un chat pour que les utilisateurs puissent discuter entre eux pendant un jeu.

Pour finir, et malgré le challenge que les circonstances ont créé, je regrette que le temps ne m'ait pas permis d'approfondir davantage le sujet, ne serait-ce que par le biais de retour d'expérience d'utilisateurs par exemple, comme je l'ai précisé edans l'introduction. C'est donc pour cela que ce TER mentionne essentiellement les aspects techniques et conceptuels, sans le recul correct quant aux améliorations à apporter suite à l'expérimentation du jeu.

En vous remerciant d'avoir pris le temps de lire ce rapport, voici donc le lien du site de jeu : <https://jeuxdedifferences.000webhostapp.com/>

Chapitre 7

Annexes

7.1 checkTermeReseau.php

```
// Construction de l'URL de la requête
$params = [
    'gotermsubmit' => 'Chercher',
    'gotermrel' => $word,
    'rel' => '',
    'relin' => 'norelin'
];
$url = "https://www.jeuxdemots.org/rezo-dump.php?gotermsubmit=Chercher&gotermrel=$word&rel=";
```

FIGURE 7.1 – Url pour recherche sur réseau

```
if (strpos($response, "Le terme '$word' n'existe pas !") !== false) {
    echo 'La balise class="jdm-warning" existe.';
} else {
    echo 'La balise n\'existe pas.';
}
```

FIGURE 7.2 – Reponse de la recherche reseau

7.2 Classement

```
<?php
// Récupération des données de classement
$query = "SELECT id_utilisateur, nom, prenom, score_utilisateur FROM utilisateur ORDER BY score_utilisateur DESC";
$stmt = $dbh->prepare($query);
$stmt->execute();
$classement = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

FIGURE 7.3 – Classement des joueurs par score en décroissant

7.3 CreerParties.php

```
// Fonction pour afficher le toast
function afficherToast() {
    const toasts = new Toasts({
        offsetX: 20, // 20px
        offsetY: 20, // 20px
        gap: 20, // The gap size in pixels between toasts
        width: 300, // 300px
        timing: 'ease', // See list of available CSS transition timings
        duration: '.5s', // Transition duration
        dimOld: true, // Dim old notifications while the newest notification stays highlighted
        position: 'top-right' // top-left | top-center | top-right | bottom-left | bottom-center | bottom-right
    });
    toasts.push({
        title: 'Parties',
        content: 'Votre partie a bien été enregistré',
        style: 'success'
    });
}
```

FIGURE 7.4 – Utilisation d'un toast pour afficher un pop-up

```
<select style="margin-left: 40px; font-size: 20px; width: 40%;" name="typeDiff">

    <option id="val0" value="typeVal0">Choisissez un type</option>
    <?php
        $sql = "SHOW COLUMNS FROM differences WHERE Field = 'type_diff'";
        $stmt = $dbh->query($sql);
        $result = $stmt->fetch(PDO::FETCH_ASSOC);

        // Récupérez les valeurs de l'énumération
        $typeDiffValues = array();
        if(preg_match("/^enum\('(.*?)'\)$/ ", $result['Type'], $matches)) {
            $enumValues = explode("'", "'", $matches[1]);
            $typeDiffValues = $enumValues;
        }

        // Affichez les valeurs de l'énumération
        foreach($typeDiffValues as $value) {
            echo '<option id="typeDiff" value="'. $value. '>' . $value. '</option>';
        }
    ?>
</select>
```

FIGURE 7.5 – Récupération de l'énumération du type de différence

7.4 Jouer.php

```
// Récupérer une partie au hasard
$sqlRandom = "SELECT * FROM parties WHERE idUtilisateur <> :idUser ORDER BY RAND() LIMIT 1;";
$sthRandom = $dbh->prepare($sqlRandom);
$sthRandom -> bindParam(':idUser', $_SESSION['utilisateur']['id_utilisateur']);
$sthRandom -> execute();
$partieCourante = $sthRandom->fetch(PDO::FETCH_ASSOC);
//var_dump($partieCourante);

//Ajout dans la session
$_SESSION['partieCouranteJoue']['idPartie'] = (int)$partieCourante['id_parties'];
$_SESSION['partieCouranteJoue']['idUserCourant'] = (int)$_SESSION['utilisateur']['id_utilisateur'];
$_SESSION['partieCouranteJoue']['idUserCreate'] = (int)$partieCourante['idUtilisateur'];
```

FIGURE 7.6 – Récupération d’une partie au hasard

7.5 connexion.php

```
$stmt = $dbh->prepare("SELECT * FROM utilisateur WHERE email = '$email' AND mot_de_passe = '$mdp'");
$stmt->execute();
$result = $stmt->fetch(PDO::FETCH_ASSOC);

$_SESSION['utilisateur'] = $result;

if ($SESSION['utilisateur']['email'] == $email && $SESSION['utilisateur']['mot_de_passe'] == $mdp) {
    echo $SESSION['utilisateur']['id_utilisateur'];
} else {
    echo "";
}
```

FIGURE 7.7 – Vérification de l’utilisateur et récupération dans la session des informations

7.6 Compte.php

```
fetch("./gestion_PHP/UPDATE/updateCompte.php?age=" + ageModifieur.value)
    .then(response => response.text())
    .then(data => {
        if (data == "update") {
            console.log(data);
            //TODO ajouter le toast pour dire update fait
        } else {
            console.log("noooo");
            //TODO ajouter le toast pour dire update non fait car erreur
        }
    })
    .catch(error => console.error(error));
```

FIGURE 7.8 – Exemple de fetch pour mettre à jour le compte pour l’âge

7.7 index.php

```
<div id="carousel-container"
  style="border: 1px solid black; max-width: 400px; max-height: 50px; text-align: center; margin: 10px auto 0;">

  <div id="carousel" class="carousel slide col-sm-9 align-self-center" data-ride="carousel"
    style="display: inline-block;">
    <?php foreach($motCourants as $index=>$mot){ ?>
    <div class="carousel-item <?php echo $index==0 ? 'active' : ''; ?>"
      <h3><?php echo $mot["nom_entite"]; ?></h3>
    </div>
    <?php } ?>
  </div>
  <a class="carousel-control-prev" href="#carousel" role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carousel" role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

FIGURE 7.9 – Code pour le carrousel à partir de la liste des entités

7.8 Organisation

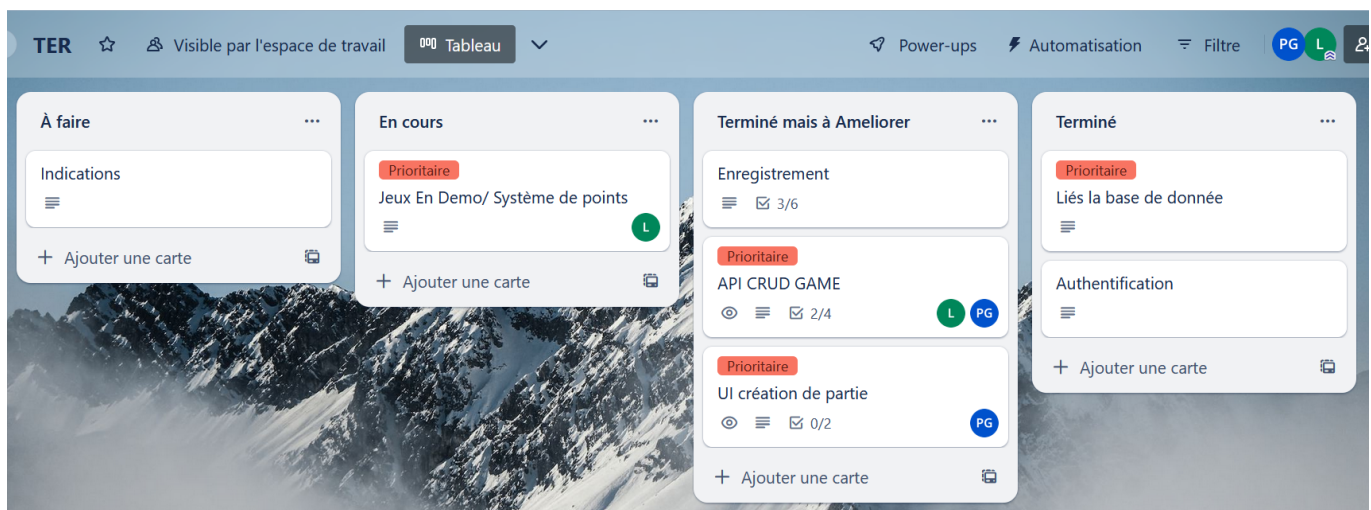


FIGURE 7.10 – Répartition des tâches sur Trello avant l'abandon

Bibliographie

Photos

Logo université montpellier

Recherche sur les jeux

Educational games (EG) design framework: Combination of game design, pedagogy and content modeling

Ressources de bases

Base de connaissances Jeux de Mots
Reseau DUMP

Organisation

Trello
Google Document

Aides en LaTeX

Accents en LaTeX
Modèle de rapport
En-tête et bas de page