Rapport Projet WEB – Find A Mate

Andrew Mc Donald, Téo Guilhou et Arthut Tréhet

May 2020

Introduction

Dans ce rapport, nous essaierons de vous expliquer avec le plus de précision possible en restant concis les choix faits et les difficultés rencontrées lors de l'élaboration de notre site Internet.

Ayant choisi le projet libre, on pense nécessaire de ré-expliquer le principe de notre site Internet "Find A Mate". Tout est venu du constat qu'une grosse communauté de joueurs est présente à l'école, l'idée est de mettre en relation ces joueurs afin qu'ils puissent trouver d'autres iiens pour jouer ensemble, d'où l'expression, Find A Mate.
Le principe est le suivant, un utilisateur va se connecter, il pourra alors choisir des jeux à ajouter à sa liste de jeux, et une fois ceci fait, il pourra lancer (ou rejoindre) des recherches avec les jeux qu'il possède. Ensuite, une fois que des joueurs sont connectés, on veut leur donner l'opportunité de discuter, de communiquer sur notre site.

1 La conception de la base de données

Notre base de données comporte 4 tables: *utilisateur* (l'appellation user faisait "buguer" PGSQL apparemment), *game*, *search* et *message*.

1.1 La table utilisateur

La table *utilisateur* va être une de nos tables les plus importantes. Elle comporte :

- Un utilisateurid, qui est de type SERIAL, qui sera la clé primaire de notre table. L'avantage du type SERIAL est que l'utilisateurid va donc s'incrémenter seul au fur et à mesure des inscriptions.
- Un pseudo, qui est un type VARCHAR, dont la taille maximale est 16. Il va permettre à l'utilisateur de s'identifier sur le site, avec une forme plus favorable à l'utilisateur qu'un utilisateurid

- Une date de création s'appelant created_at
- Une promo, qui est un type VARCHAR, étant donné que la cible de notre site est les élèves de l'ENSIIE.
- Un VARCHAR s'appelant isAdmin qui va permettre, ou non, d'accéder à des fonctionnalités d'administrateurs
- Un VARCHAR s'appelant pseudoDiscord. Effectivement, on s'est dit que si les joueurs voulaient jouer ensemble, il vaudrait mieux de leur faciliter l'accès à un chat vocal (type Discord).
- Une email et un mot de passe, qui sont aussi des VARCHAR, qui serviront à l'identification sur le site, pour pouvoir se connecter à son compte.

1.2 La table game

La table game correspond à la liste de tous les jeux disponibles pour lancer des recherches, et elle comporte:

- un gameid, qui est encore de type SERIAL (pour les mêmes raisons que dans la table utilisateur)
- Le nom du jeu, appelé gameName dans notre table, qui va permettre de savoir de quel jeu on parle.
- Un INTEGER isFree qui est égal à 1 si le jeu est gratuit, 0 sinon.
- Une description du jeu, de taille maximale de 100 caractères, appelée GameDescription
- Et un INTEGER isAccepted pour savoir si le jeu a été validé par les administrateurs. (car, on laisse les utilisateurs proposé des jeux, mais les demandes doivent être modérées par les admins.

1.3 La table search

La table search est la table qui va nous permettre de lancer des recherches, et elle comporte les champs suivants:

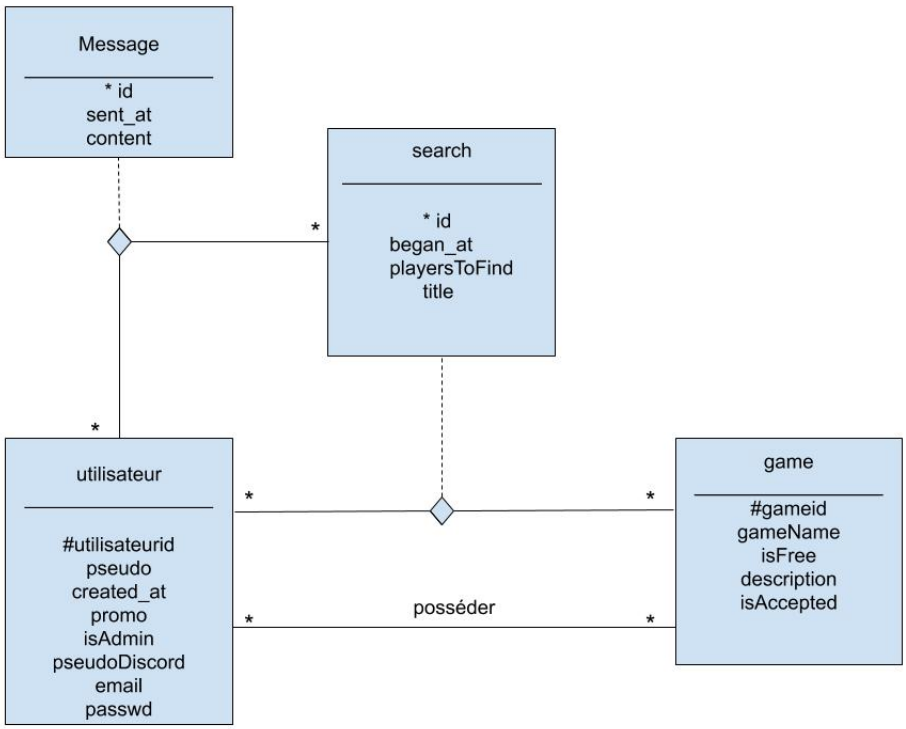
- un searchid, de type SERIAL.
- une date began_at, qui va nous permettre de connaître depuis combien de temps la recherche est en cours
- un pseudo, qui sera le pseudo de la personne qui a commencé la recherche.
- un gameName, qui sera le nom du jeu sur lequel on a lancé la recherche.
- un INTEGER playersToFind pour savoir combien de joueurs le créateur de la recherche veut trouver pour jouer
- un titre, appelé title, qui va permettre de donner un court résumé de sa recherche pour attirer les joueurs.

1.4 La table message

La table message est la table qui va permettre aux utilisateurs de communiquer lorsque l'un a rejoint la recherche de l'autre. Elle est composée de:

- un messageid de type SERIAL, qui sera la clé primaire.
- un searchid pour savoir dans quel "salon" ce message a été envoyé.
- un emittor, qui est en réalité un pseudo, pour savoir qui a envoyé le message.
- une date sent_at pour savoir l'heure à laquelle a été envoyé le message
- un VARCHAR content, qui n'est autre que le contenu du message

1.5 Schéma UML de notre base de données:



2 Infrastructure back-end

Dans cette partie, nous verrons comment nous avons fait pour réaliser tout le coté "serveur" de notre site, et les difficultées rencontrées pour y arriver.

2.1 La page d'accueil

Tout d'abord, dès qu'on arrive sur la première page de notre site (index.php), le site vous démarre une session, ce qui va nous permettre de déclarer des variables de type $_SESSION,et ainsi nous permettre de différencier les utilisateurs, et donc donner l'accès, ou non, aux fonctionnalités de notre site.

Cette page d'acceuil du site (home.php) a été réalisée par Andrew Mc Donald. Pour ajouter un peu de style et d'esthétique à notre site, on a décidé de faire une sorte de mini tutoriel d'utilisation de notre site, qui apparaît en cliquant sur un bouton. Cette animation a été faite grâce aux fonctionnalités de JQuery et de Bootstrap 4.

2.2 La page d'administration

Cette page, également réalisée par Andrew Mc Donald, est la seule page de notre site qui est accessible juste par URL (localhost:8080/admin.php). Elle affiche du contenu seulement si l'utilisateur qui est dessus est un administrateur, et sur cette page, il pourra modérer les utilisateurs du site si ils ont un comportement toxique avec les autres utilisateurs.

2.3 La partie authentification

Ici, on va détailler toute la partie authentification de notre site Internet, c'est-à-dire, l'inscription et la connexion à notre site internet.
Lorsqu'un utilisateur s'identifie sur notre site, on déclare ses variables de session, en l'occurence $_SESSION['name'], $_SESSION['id'] et $_SESSION['isAdmin']. Cela va nous permettre de personnaliser notre affichage en fonction de nos utilisateurs.

2.3.1 L'inscription:

Nous souhaitons d'abord dire que, par soucis de sécurité, nous avons crypter tous nos mots de passes en sha1, et de manière générale, tous les contenus des formulaires ont été "cryptés" en htmlspecialchars, ce qui nous permet de nous protéger contre des attaques par injections SQL.

Commençons par le début, l'inscription, implémenté par Téo Guilhou. Il est possible de s'inscrire sur notre site en cliquant sur le bouton "Sign Up" qui est dans la barre de navigation, qui nous redirige vers une page signup.php. Le contenu de cette page est assez simple, il s'agit d'un formulaire destiné à être rempli par l'utilisateur.
Il y a bien sûr un système de validation de formulaires (même si il n'est pas fait en JavaScript, car on ne connaissait pas assez bien ce langage au début du projet): on demande à l'utilisateur de rentrer une adresse mail valide, avec un format d'adresse mail (exemple@exemple.com), on lui demande de rentrer

un pseudo, n'importe lequel, il ne peut juste pas être vide, et on lui demande également de rentrer deux fois son mot de passe pour s'assurer qu'il ne fasse pas de fautes de frappe. Si tout ça n'est pas respecté, le formulaire n'est pas soumis, et l'utilisateur n'est pas ajouté à la base de données.
En parlant d'insertion à la base de données, cette dernière se fait via une fonction de UserRepository.php qui s'appelle insert.
Donc, une fois que le formulaire est envoyé, l'utilisateur est redirigé vers la page adduser.php, qui va se charger de faire toutes les vérifications, et si tout est bien en règle, on fait appel à la fonction insert de UserRepository.
De plus, les variables de sessions nous permettent de faire en sorte que l'utilisateur qui s'inscrit soit déjà connecté après son inscription, ce qui nous semblait essentiel pour l'expérience utilisateur.

2.3.2 La connexion

La partie connexion (donc le cas où l'utilisateur a déjà un compte) a été réalisée par Arthur Tréhet.

Il s'agit encore d'un formulaire, qui suit le même mécanisme à peu de choses près que l'inscription. Le formulaire de connexion est traité par une page verification.php qui va faire un fetchAll sur tous les utilisateurs de la base de donnée, puis comparer en php l'email ainsi que le mot de passe crypté rentrés par l'utilisateurs à ceux présents dans la base de donnée. Si il n'y a aucun compte associé à ce que l'utilisateur vient de rentrer, alors il sera redirigé vers la page de connexion avec un message d'erreur, sinon il sera redirigé vers la page home.

Nous avons de plus rajouter un lien vers la page Sign Up au cas où l'utilisateur qui essaye de se connecter n'a pas de compte.

2.4 La liste de jeux

Cette partie, implémentée par Andrew Mc Donald, a pour but de permettre aux utilisateurs de voir une liste des jeux disponibles sur le site, pour pouvoir lancer des recherches sur ces-derniers.

Notre souhait pour cette partie était que seuls les jeux acceptés par les administrateurs soient visible pour les "non-admin", donc la page de gamelist.php est aussi une page de modération à part entière. C'est pour ça que je pense qu'il est plus cohérent de faire une partie dans le cas où l'utilisateur est un utilisateur lambda, et une partie dans le cas où c'est un administrateur.

2.4.1 Cas de l'utilisateur lambda:

Tout d'abord, si l'utilisateur ne s'est pas identifié sur le site, il peut juste consulter la liste des jeux disponibles, il voit donc le nom du jeu, si le jeu est gratuit ou non, et la description de ce-dernier. On récupère toutes ces informations via une requête SQL, appelée par une fonction PHP déclarée dans GameRepository.php.

Cependant, si l'utilisateur est connecté, il y a un bouton "Ajouter" qui va permettre à l'utilisateur, comme son nom l'indique, d'ajouter un jeu à sa liste de jeux. De plus, Andrew a rédigé une fonction *isInAcquired* qui permet de vérifier si un jeu appartient à la liste de l'utilisateur, si c'est le cas, le bouton disparaît pour afficher à la place un texte indiquant que le jeu est possédé.

2.4.2 Cas de l'administrateur:

Comme je l'ai dit, cette page fait également office de page de modération. Effectivement, en plus des informations décrites ci-dessus, l'administrateur voit si le jeu a été accepté ou non (via le INTEGER isAccepted, qui est égale à 0 si le jeu est en cours de traitement, 1 si il est accepté et -1 si il a été refusé par un administrateur). Donc en plus du bouton "Ajouter", il y a également un bouton "Validate", un bouton "Decline" et un bouton "Delete" pour supprimer ce jeu de la base de données.

2.5 Requêtes d'ajout de jeu:

La page addgame.php est là pour permettre aux utilisateurs d'ajouter des jeux au site (moyennant une acceptation de l'administration) si ils veulent jouer à un jeu qui n'est pas sur le site. Cela fonctionne via un formulaire qui sera ensuite traité par le fichier GameRepository.php.

2.6 Profil

2.6.1 Consulter un profil:

On peut consulter son profil via un bouton dans le header du site, pour cela il faut d'abord se connecter si ce n'est pas déjà fait. Pour consulter le profil d'un autre utilisateur, il faut que celui-ci ait créé une recherche pour pouvoir cliquer sur son pseudo et ainsi être redirigé vers son profil. Le profil contient le peudo, l'addresse mail, la promo, le pseudo Discord ainsi que la liste des jeux que l'utilisateur possède. Si on regarde son propre profil alors on verra apparaitre deux boutons qui sont "Ajouter des jeux à ma liste" et "Modifier le profil".

2.6.2 Modifier son profil:

Si on clique sur le bouton de modification de profil la page change pour afficher une page où l'on peut éditer son profil via un formulaire, on pourra y changer son pseudo, à condition qu'il ne soit pas déjà pris par un autre utilisateur, on pourra

aussi changer sa promo, son pseudo discord. Dans cette page de modification de profil on peut aussi choisir de supprimer son compte. Cette suppression est confirmée via un script javascript réalisé par Andrew Mc Donald. L'idée était de faire un prompt, et si l'utilisateur rentre le texte que l'on veut, on soumet le formulaire de suppression.

2.7 Les recherches

2.7.1 Créer une recherche:

Si on est connecté on peut créer une recherche via un bouton dans le header du site. Sinon, on est redirigé vers la page de connexion. Dans cette page de création de recherche, un utilisateur peut créer une recherche. Pour cela il doit sélectionner un jeu parmi ceux qu'il possède, ensuite il doit ajouter un titre à sa recherche puis renseigner le nombre de joueurs recherchés.

2.7.2 Toutes les recherches:

Sur la page "allSearch", on peut tout d'abord voir l'ensemble des recherches présentes dans la base de données triées par ordre décroissant de date de création (les plus récentes en premier), avec pour chacune des recherches, toutes les informations contenues sur la recherche (titre, nombre de joueurs recherchés, jeu et pseudo de la personne qui a initié la recherche), ainsi qu'un bouton rejoindre. Si l'on n'est pas encore connecté, le bouton rejoindre nous renvoie vers la page de connexion, sinon, il nous permet de rejoindre un salon de discussion dédié à cette recherche, pour que les utilisateurs puissent se contacter, et ainsi s'échanger les informations nécessaires pour jouer ensemble s'ils le veulent. Pour chaque recherche, à l'emplacement où est écrit le pseudo de la personne qui a créé la recherche, se trouve un lien vers son profil. De plus, pour que les utilisateurs puissent cibler les recherches qui les intéressent, Il y a formulaire de recherche (recherche de recherche). Il n'y a qu'un seul champ, qui va faire un tri sur toutes les recherches dont le nom du jeu ou le titre contient ce qu'à rentrer l'utilisateur. Par exemple si un utilisateur tape "leg", alors les recherches associées au jeu "league of legends" apparaitront ainsi que toutes les recherches dont le nom contient "leg". Ce tri des recherches à afficher est fait par une requête SQL.

2.7.3 Salon de discussion:

Lorsqu'un utilisateur appuie sur le bouton "rejoindre" de la page "allSearch", cela le redirige vers le salon de la recherche en question, où les utilisateurs ayant rejoint cette recherche peuvent se parler via un chat php. Le créateur et les personnes qui rejoignent cette recherche peuvent donc s'échanger des informations via ce salon.