

# Projet Internet et Outils

# Sommaire

INSTALLATION	2
Concept	3
Architecture	8
Sécurité	11
Expérience	13

# Installation

Pour un bon fonctionnement du projet HRP il est nécessaire d'effectuer les points suivants :

1. Installation d'un serveur Web : Vous avez le choix entre plusieurs serveurs web mais le serveur de type Apache est à privilégier.  
( <httpd.apache.org> )
2. Vous devez installer php de version 8.0.2 ou bien la plus récente.  
( <php.net/downloads> )
3. Installation de la base de données de type MariaDB.  
( <mariadb.org/download> )
4. Vous pouvez installer l'interface phpMyAdmin pour faciliter l'interaction avec la base de données. ( <phpmyadmin.net> )
5. Il est nécessaire de configurer le serveur.  
( </assets/config/config.ini> , [/root\\_public/.htaccess](/root_public/.htaccess) )
6. Il est nécessaire d'initialiser la base de données (comptes admin / bot) :
  - Modifiez le fichier de configuration ([/assets/config/db\\_init.json](/assets/config/db_init.json))
  - Vous pouvez ensuite lancer l'initialisation en ouvrant la page [/root\\_public/init\\_database.php](/root_public/init_database.php)

# Concept

Le projet HRP répond aux problématiques d'un site de rencontre respectueux de la vie privée de ses utilisateurs.

La plupart des réseaux sociaux ne donnent aucun contexte pour forcer l'interaction entre les utilisateurs. C'est pourquoi le projet HRP est organisé sous la forme d'un jeu de rôle. Chaque utilisateur se voit attribuer un rôle auquel il devra se conformer pour interagir publiquement. Cette contrainte permet de forcer l'inspiration dans les interactions. De plus, en fonction des rôles, des phrases auto-générées seront proposées pour les utilisateurs ne sachant pas comment lancer la conversation.

Le choix de l'univers d'Harry Potter est arbitraire. Un peu tout le monde a déjà vu un film Harry Potter, donc on s'est dit que ça serait une bonne idée. Mais évidemment on pourrait imaginer des équivalents du site pour d'autres univers.

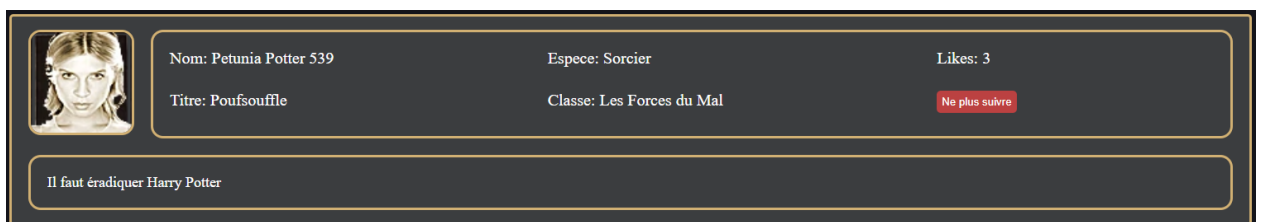
Chaque compte est scindé en 2 parties, une privée et une publique.

La partie publique sert à l'interaction avec les utilisateurs inconnus.

Elle doit être créée depuis la page de paramètre.

Les données du compte publique sont auto-générées et non modifiables (sauf pour la description).

Exemple de page publique auto-généré:



Le site possède un système de rencontre.

Pour pouvoir suivre de nouvelles personnes.

Les personnes qui suivent un compte ont plus de chance d'apparaître dans les rencontres proposées à ce compte.

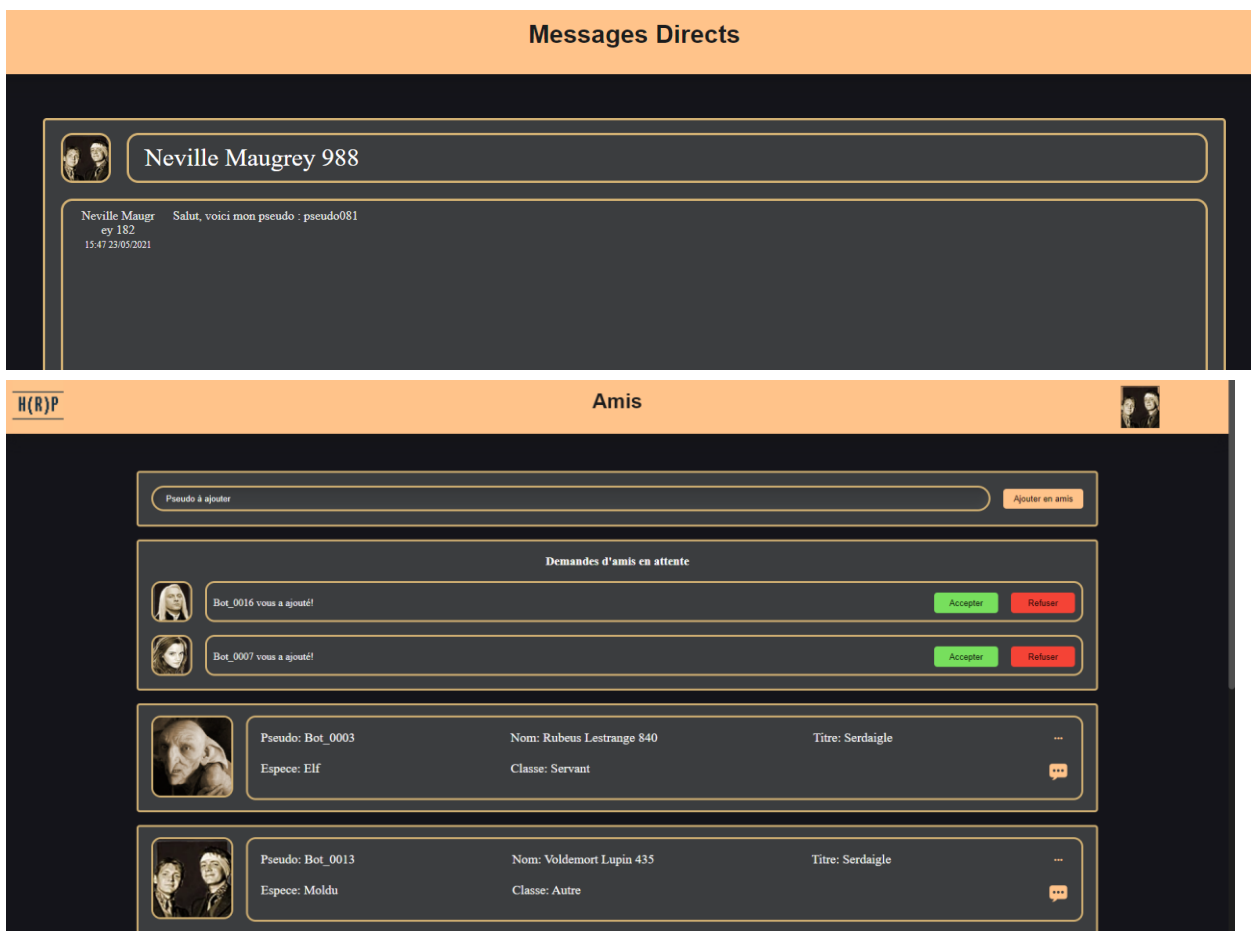


Si 2 comptes s'entre suivent (*like*), alors il y a une rencontre (*match*).



Un utilisateur peut révéler son pseudo privé à un autre par message direct pour que celui-ci l'ajoute en ami.

Les messages directs entre amis et comptes publics sont distincts

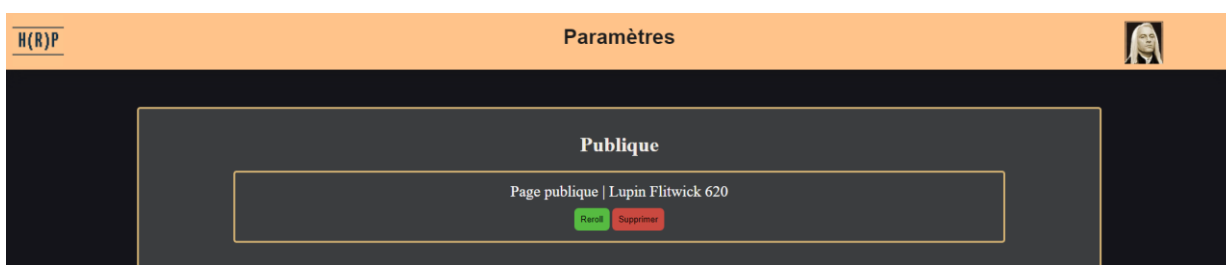


Le fil d'actualité ainsi que toutes autres fonctionnalités publiques sont réservés aux comptes publics.



Il est possible de générer un nouveau profile public toute les 24h (*Reroll*)  
On peut aussi récupérer le profil supprimé sans attendre 24h.

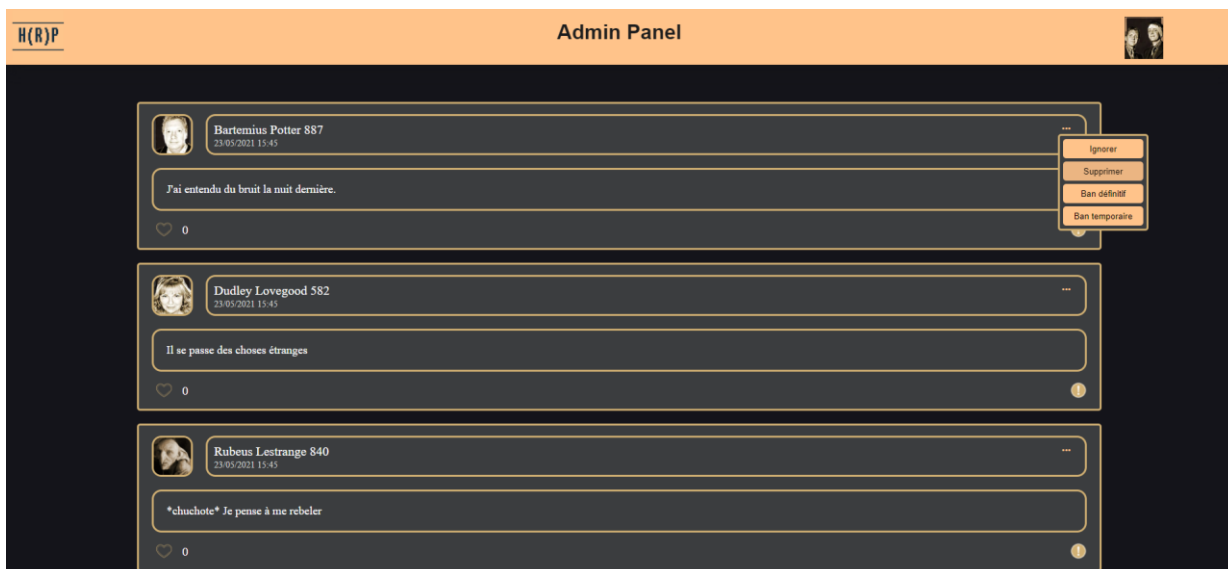
*Avant suppression:*



*Après suppression:*



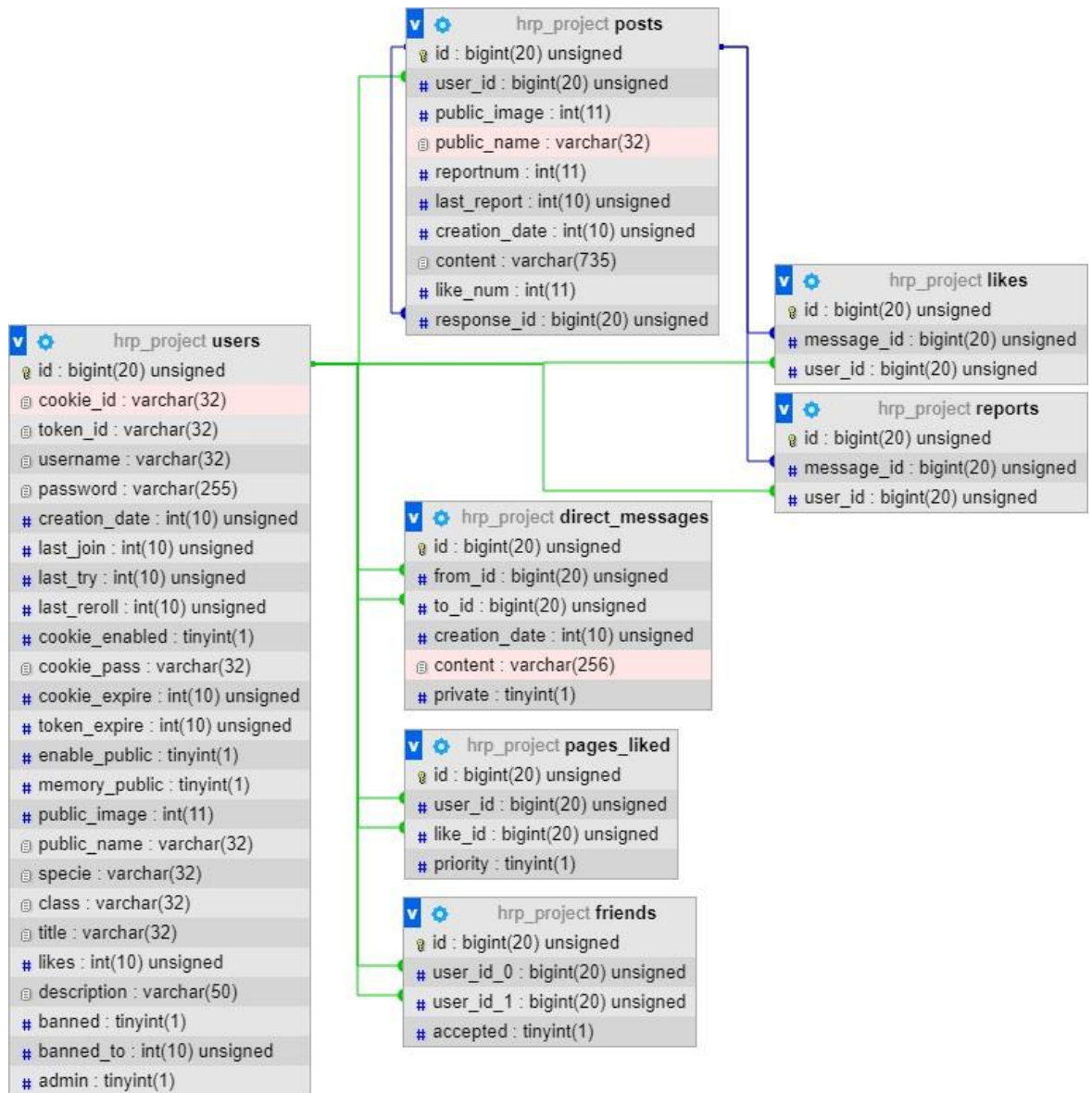
Il est possible de signaler les posts inappropriés.  
Si l'admin décide de bannir temporairement l'utilisateur,  
alors celui ci perdra son compte public et devra la réactiver ou en créer un  
nouveau passé la période de bannissement.



Un compte banni temporairement possède toujours sa partie privée et peut donc  
toujours interagir avec ses amis par messages directs.

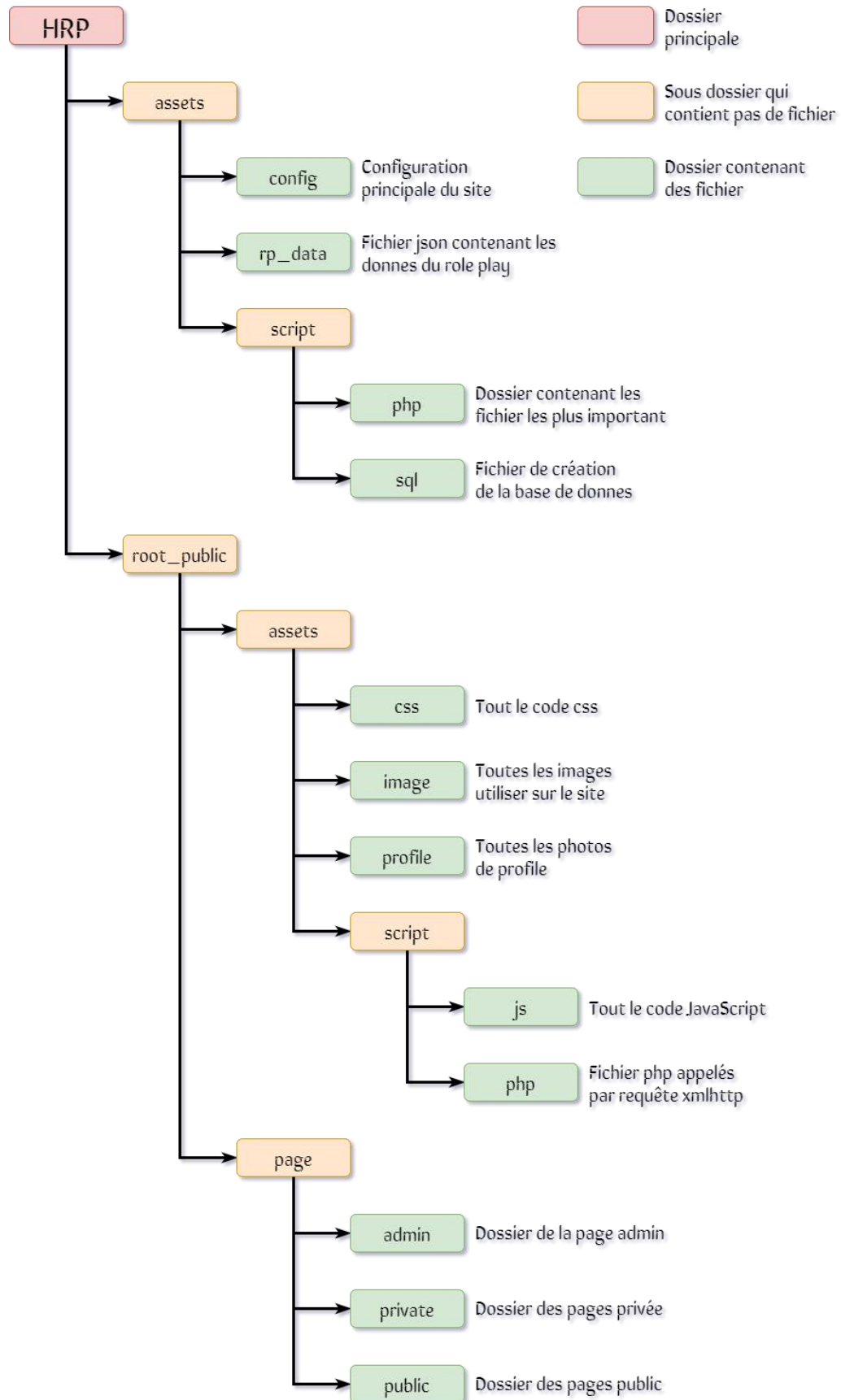
# Architecture

## base de donnée





## Arborescence



Dans l'arborescence,  
les pages sont scindées en 3 groupes (*admin*, *private*, *public*)

Les pages "admin" accessibles uniquement par les comptes administrateurs.

Les pages "private" accessibles uniquement par les utilisateurs.

Les pages "public" accessibles à tous.

A noter que les pages de profil publiques sont considérées comme des pages publiques et de même pour la barre de recherche de profil.

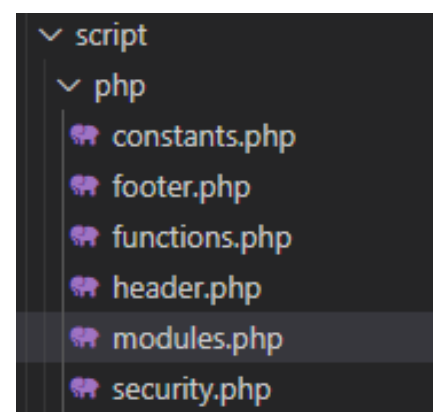
Le code minimal d'une page est le suivant :

```
<?php $global_params = [  
    "root"      => "../..../", // emplacement de la racine par rapport au fichier  
    "root_public" => "../..../", // emplacement du dossier root_public  
    "title"     => "Titre de la page",  
    "css_add"    => ["fichier1.css", "fichier2.css"], // css en plus du css de base  
    "redirect"   => TRUE, // TRUE pour les pages nécessitant un compte  
    "admin_req"  => TRUE // TRUE pour les pages admin  
];?>  
<!-- ----- -->  
<?php require($global_params["root"] . "assets/script/php/functions.php"); ?>  
<?php require($global_params["root"] . "assets/script/php/header.php"); ?>  
<!-- ----- -->  
  
    <!-- ICI LE CODE HTML -->  
  
<!-- ----- -->  
<?php require($global_params["root"] . "assets/script/php/footer.php"); ?>
```

Chaque fichier php utilise les fichiers constants.php et functions.php pour accéder aux fonctions générales du site et les variables globales.

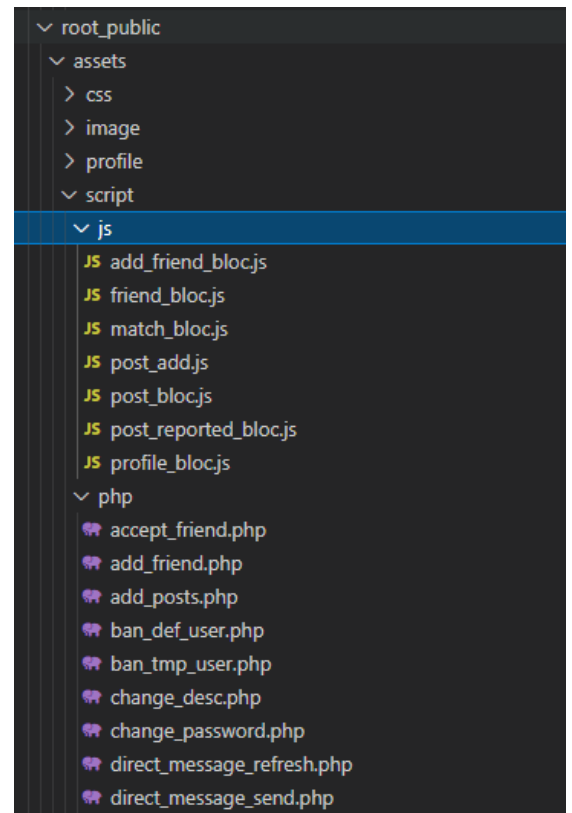
Le fichier module.php lui est utile pour les fonctions à effets de bords tels que les posts ou les blocs utilisateurs.

Le fichier security.php comporte des fonctions relatives à la connexion et la suppression de données.



Le site n'utilise presque pas de form.

Les fonctions du site sont assurées par requêtes xmlhttp en javascript vers des fichiers du dossier [root\\_public/assets/script/php](#)



La mise en place d'un serveur websocket pouvait poser problème, par conséquent les messages directs sont gérés par des requêtes xmlhttp en javascript.

Une fonction *refreshMessages* est appelée à intervalle régulier pour demander au serveur les messages de la base de données moins vieux que la dernière requête.

Pour éviter les problèmes de synchronisation,

Lors de l'envoi de message, la fonction *refreshMessages* est automatiquement appelée. De plus, au bout de k appels de la fonction *refreshMessages*, on efface tous les messages et on fait une demande propre des 20 derniers messages au serveur.

# Sécurité

## Protection contre les injections SQL

Pour toutes chaînes de caractères en provenance d'un utilisateur.  
La chaîne est échappée avant d'être mise dans une requête sql .

## Protection contre les injections HTML

Les messages/descriptions/posts échappent les balises html.  
Utile surtout pour éviter l'envoi de balise *<script>* par message.

## Protection des mots de passe par hash

Tout mot de passe est haché avant d'être envoyé dans la base de données.  
L'algorithme par défaut utilisé par la fonction *password\_hash* de php permet de rajouter du sel aux hashes pour éviter les attaques par dictionnaire sur la base de données.

On stocke dans la base de données les hash de mots de passe avec des *varchar(255)* et non pas des *varchar(66)* en cas d'évolution de la l'algorithme par défaut de *password\_hash*.

## Protection contre les attaques CSRF (Cross-site request forgery)

Les fonctions du site sont assurées par des fichiers php.  
Il est nécessaire de s'assurer que les appels de fichiers sont bien volontaires.  
Un site malveillant pourrait faire appeler à un utilisateur ces fichiers sans qu'il le sache en javascript. Ainsi il pourrait lui faire exécuter des actions à son insu.

Pour contre ça, chaque compte connecté possède un token d'utilisation.  
A l'ouverture d'une page, le serveur met dans le javascript de la page ce token.

```
<!-- HEADER -->
<!-- ----- -->

<script>
    // variable globals
    const token_id = <?=json_encode($_SESSION["connected"] ? $_SESSION["token_id"] : "none")?>;
    const next = <?=json_encode($label_params["next"])?>;
```

Ce token est envoyé avec chaque requête xmlhttp pour attester de la légitimité de la demande:

```
function verifyToken() {
    if (
        !isset($_SESSION["connected"]) || !$_SESSION["connected"] ||
        !isset($_POST["token_id"]) || !isset($_SESSION["token_id"]) || !isset($_SESSION["token_expire"]) ||
        $_POST["token_id"] != $_SESSION["token_id"] || $_SESSION["token_expire"] < time()
    ) {
        echo json_encode([
            "success" => false,
            "error" => "token_error"
        ]);
        exit();
    }
}
```

### Protection contre les attaques par dictionnaire et bruteforce

Pour empêcher les attaques par bruteforce,

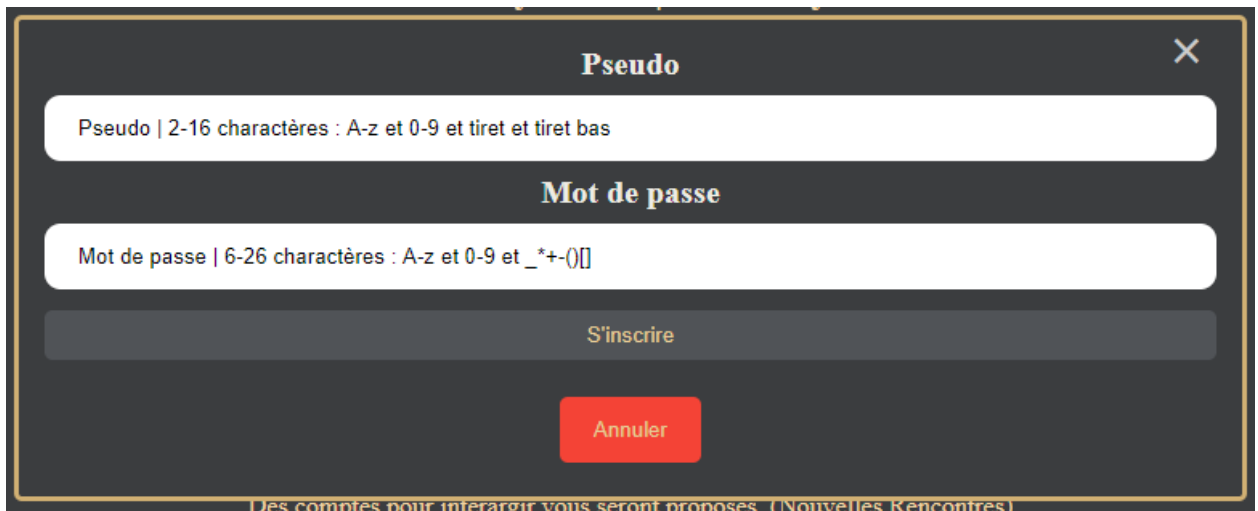
Les utilisateurs sont invités à utiliser des mots de passe un minimum long.

Pour empêcher les attaques par dictionnaire,

Les utilisateurs sont invités à utiliser des caractères spéciaux pour éviter que les mots qui composent leurs mot de passe soient devinable.

Exemple de permutation (e vers ?) :

GantDeBoxeSecheLinge => GantD?Box?S?ch?Ling?



The image shows a registration form with a dark background and a gold border. At the top, the title 'Pseudo' is centered in a gold font, with a close button (X) on the right. Below the title, there are two input fields. The first field is labeled 'Pseudo | 2-16 caractères : A-z et 0-9 et tiret et tiret bas'. The second field is labeled 'Mot de passe | 6-26 caractères : A-z et 0-9 et \_\*+~()[]'. Below these fields, there is a grey button labeled 'S'inscrire' and a red button labeled 'Annuler'. At the bottom of the form, there is a line of text: 'Des comptes pour interagir vous seront proposés. (Nouvelles Rencontres)'.

De plus, le nombre de requêtes de connexion à un compte est limité dans le temps.

# Expérience

Le travail à plusieurs nécessite de l'organisation. Avant de commencer à coder on s'est donc mis d'accord sur la direction générale du projet.

On s'est concertés pour savoir l'ensemble des pages dont on aurait besoin pour le projet, le fonctionnement de ces pages a été schématisé à l'avance.

Des conventions ont été fixés;  
pas de caractères spéciaux;  
les dossiers et table sql en minuscule avec tiret du bas;  
les fonctions javascript/php en CamelCase;

Pour éviter les conflits lors des merges, on évite de travailler en même temps sur les mêmes fichiers. Le fait d'avoir un fichier php par fonctionnalité du site à beaucoup aidé à éviter les conflits.

Le fichier all.css ne garde que les styles très généraux,  
chaque page a son propre fichier css pour éviter d'écrire du code qui se contredit au moment des merges sur github.

Nos 2 installations étant totalement différentes, l'ensemble des fichiers utilisent des chemins relatifs. Nous n'avions pas tous deux le serveur installé au même emplacement par rapport au projet, l'utilisation de chemin absolu causait bien trop de soucis.

Merci d'avoir pris le temps de lire notre rapport.