Année académique 2017 - 2018

Programmation Web

Rapport

Cumbo Fabio

Bachelier en Informatique & systèmes orientation réseaux & télécommunications avec *option développement*

2e année

Enseignants : MALAISE Antoine

SCOPEL Fabrice

HAUTE ECOLE EN HAINAUT

Campus technique

8a, avenue V. Maistriau B-7000 Mons
Tél : +32 (0)65 33 81 54
Fax : +32 (0)65 31 30 51
E-mail : tech-mons@heh.be
www.heh.be

Table des matières

1. Introduction

Ce projet porte sur la création d'un site Web dynamique en PHP dans le cadre du cours de « Programmation Web ».
Le thème du site Web porte sur le jeu mobile Clash Royal.

Le principe du jeu est assez simple, chaque joueur choisit un deck de départ comportant 8 cartes uniques parmi une liste. Une fois qu'une partie est lancée, 2 adversaires doivent combattre avec leur deck composé et l'objectif est de faire tomber le plus de tours possibles de l'adversaire avant un temps imparti. Chaque carte représente un personnage jouable sur la carte du jeu et possède un certain cout.

Le but principal du site permettra aux utilisateurs de les aider à trouver leur deck favori au moyen du partage de decks par la communauté. Chaque utilisateur inscrit peut donc contribuer à une liste de decks en y ajoutant le sien et tout le monde pourra comparer le sien avec celui de quelqu'un d'autres.

2. Conception de la base de données

La base de données du site (*clashroyal*) comprend 3 tables :

- Une table « *profil* » pour stocker les comptes inscrits et les informations relatives aux comptes.

- Une table « *listeDeck* » pour enregistrer les données des decks des utilisateurs inscrits.

- Une table « *categorie* » qui contient la liste des types de decks possible dans le jeu.

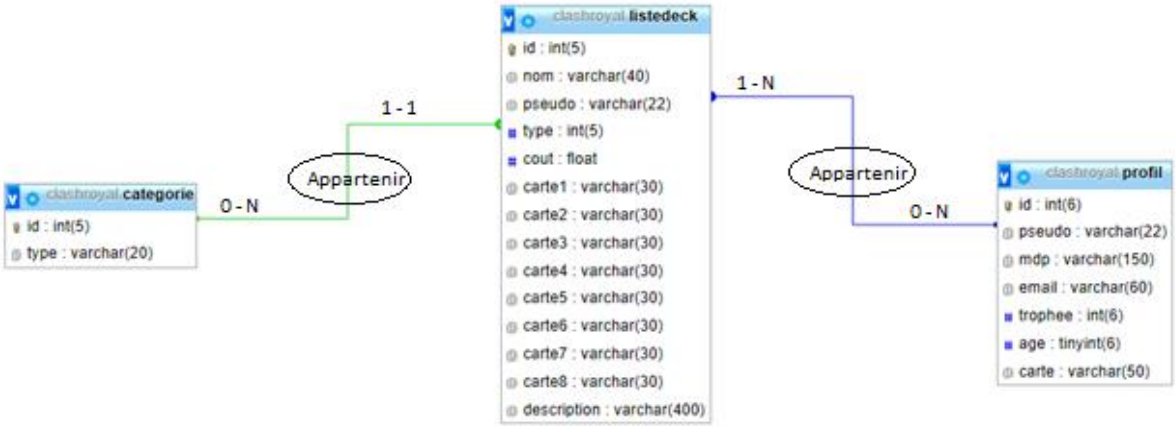


Schéma conceptuel de données

Pour éviter la redondance au niveau des types de deck, une table « *categorie* » a été construite. Une contrainte de clé étrangère a donc été établie entre l'ID de la table « *categorie* » et le type de deck de la table « *listeDeck* » . Les catégories de decks sont donc affichées grâce à une jointure entre ces 2 tables pour récupérer le nom du type de deck.

La table « *profil* » est également liée avec la table « *listeDeck* » étant donné qu'un utilisateur possédant un deck est forcément inscrit et donc enregistrée dans la table « *profil* ». On a donc une autre contrainte de clé étrangère entre les pseudos de la table des utilisateurs inscris et ceux dans la table de la liste des decks.

Le système de gestion de base de données utilisé est MySQL.

3. Architecture de conception du site

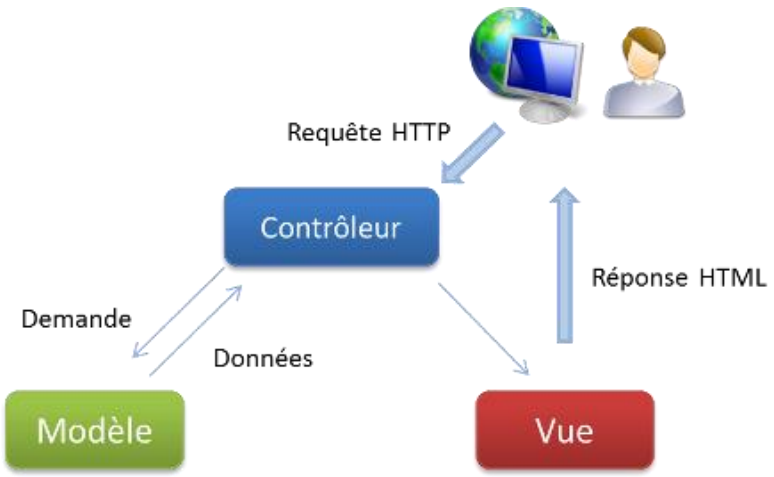
La conception du site est divisée en 3 parties distinctes :

- La partie vue visant a contenir le code HTML/CSS (le CSS n'étant pas directement intégré dans l'HTML, un fichier style.css est donc appelé depuis l'HTML) et un peu de PHP quand cela est nécessaire. Lorsque l'utilisateur navigue sur le site, c'est cette partie qu'il pourra visionner.

- La partie modèle visant à ne contenir uniquement les fonctions effectuant des requêtes sur la base de données permettant l'insertion, la sélection, la suppression et la mise à jour sur les tables (méthode CRUD) .

- La partie contrôleur qui s'occupe de faire le lien entre les 2 autres parties, elle représente la conception logique du site, lorsque l'on navigue, on accède au contrôleur qui lui appellera les 2 autres parties.

Le site se base donc sur l'architecture MVC (Model-View-Controler) ce qui permet une meilleure évolutivité ainsi qu'un code plus propre et plus logique.



4. Arborescence du site

```
| index.php
|
├──controleur
|    ajoutDeck.php
|    connexion.php
|    contact.php
|    deconnexion.php
|    deleteDeck.php
|    deleteProfil.php
```

```
|   editDeck.php
|   editProfil.php
|   erreur.php
|   index.php
|   inscription.php
|   jeu.php
|   listeDeck.php
|   listeMesDecks.php
|   profil.php
|   rechercheDecks.php
|   sessionCheck.php
|
├---css
|   bootstrap3.min.css
|   bootstrap4.min.css
|   style.css
|
├---fonts
|   glyphicons-halflings-regular.eot
|   glyphicons-halflings-regular.svg
|   glyphicons-halflings-regular.ttf
|   glyphicons-halflings-regular.woff
|   glyphicons-halflings-regular.woff2
|
├---images
|   banniere.jpg
|   jeu.jpg
|
├---js
|   bootstrap3.min.js
|   bootstrap4.min.js
|   jquery.min.js
|   script.js
|
├---modele
|   ajoutDeckModele.php
|   connexionModele.php
```

```
|       deleteDeckModele.php
|       deleteProfilModele.php
|       editDeckModele.php
|       inscriptionModele.php
|       listeDeckModele.php
|       listeMesDecksModele.php
|       modele.php
|       profilModele.php
|
└──vue
        ajoutDeckView.php
        connexionView.php
        contactView.php
        deleteProfilView.php
        editDeckView.php
        editProfilView.php
        erreurView.php
        footerView.php
        indexView.php
        inscriptionView.php
        jeuView.php
        listeDeckView.php
        listeMesDecksView.php
       menuView.php
       profilView.php
        rechercheView.php
```

5. Accès à la base de données

Les accès à la base de données ont été réalisés à l'aide de l'interface de connexion PDO qui possède quelques avantages.

En effet, l'interface PDO permet de se connecter à n'importe quelle base de données, les fonctions pour la récupération des données sont identiques quel que soit le SGBD SQL utilisé contrairement à l'interface mysqli qui est limité à MySQL.
De plus il est en plein développement et plus intuitif à utiliser.

Modele.php

```php
<?php
date_default_timezone_set('Europe/Brussels');
function getBdd()
{
    try {
        $bdd = new PDO('mysql:host=localhost;dbname=clashroyal;charset=utf8', 'root', 'Test123*');
        return $bdd;
    }
    catch (Exception $e) {
        echo '<p> Erreur de connexion à la BD <p> ' . $e;
    }
}

function executeReq($sql, $param)
{
    $bdd = getBdd();
    $req = $bdd->prepare($sql);
    $req->execute($param);
    return $req;
}

function executeQueryReq($sql)
{
    $bdd = getBdd();
    $req = $bdd->query($sql);

    return $req;
}

?>
```

Pour effectuer des manipulations dans la base de données, j'ai utilisé les requêtes préparées lorsqu'il y avait un ou plusieurs paramètres. Celles-ci effectuent toutes les vérifications portant sur les variables transmises avant d'exécuter la requête, ce qui empêche les injections SQL et améliore donc la sécurité. Dans les autres cas, la fonction query() a été utilisé, celle-ci permet de faire des requêtes directes sans vérification et donc plus rapidement.

6. Conception du site

Les principale conception logique et choix entrepris pour la conception du site web seront présentés dans cette section.

6.1. Inscription

Cette page permet l'inscription d'un utilisateur, un formulaire est utilisé dans la vue pour qu'il puisse insérer ses informations et les envoyer.

inscription.php

```php
<?php
```

```php
include_once('sessionCheck.php');
include_once('../modele/inscriptionModele.php');
include_once('../vue/menuView.php'); //Obligé de faire ça sinon quand je suis connecté je peux encore m'inscire
if (!isset($_SESSION['droit']) || $_SESSION['droit'] == 0) //0 je peux m'inscrire car je ne suis pas connecté
    {
    include_once('../vue/inscriptionView.php');

    if (isset($_POST['pseudo']) || isset($_POST['mdp']) || isset($_POST['mdp2']) || isset($_POST['mail']) || isset($_POST['tr'])) //si je suis passé par le formulaire
        {
        $pseudo = htmlspecialchars($_POST['pseudo']);
        $mdp    = htmlspecialchars($_POST['mdp']);
        $mdp2   = htmlspecialchars($_POST['mdp2']);
        $mail   = htmlspecialchars($_POST['mail']);
        $tr     = htmlspecialchars($_POST['tr']);
        //Si on supprime le required dans le form
        if (!empty($_POST['pseudo']) && !empty($_POST['mdp']) && !empty($_POST['mdp2']) && !empty($_POST['mail']) && !empty($_POST['tr'])) {
            if (preg_match("#^[a-zA-Z0-9\'\.\,\;\-\_\+\-\!\?\&\é\@\"\'\#\è\à\ç\€\$\ù\%\:\)\(\* ]{4,20}$#", $pseudo)) {
                if (strlen($mdp) >= 4 && strlen($mdp) <= 20) {
                    if ($mdp == $mdp2) {
                        if (preg_match("#^[a-z0-9._-]+@[a-z0-9._-]{2,}\.[a-z]{2,4}$#", $mail)) {
                            if (preg_match("#^[0-9]{0,5}$#", $tr) && $tr >= 0 && $tr <= 8000) {
                                //Check pseudo déjà utilisé
                                $donnees = checkPseudo($pseudo);

                                if (empty($donnees)) {
                                    //Check e-mail déjà utilisé
                                    $donnees = checkEmail($mail);

                                    if (empty($donnees)) {
                                        $_SESSION['droit'] = 0;
                                        $mdpChiffre        = sha1($mdp);
                                        inscription($pseudo, $mdpChiffre, $mail, $tr);

                                        //On remet les paramètres initial
                                        $pseudo = "";
                                        $mail   = "";
                                        $tr     = "";
                                        //On rafraichis la page pour prendre en compte les paramètre initial
                                        echo '<meta http-equiv="refresh" content="0;URL=inscription.php">';
                                        //On prend en compte qu'on vient de s'inscrire
                                        $_SESSION['inscriptionStatut'] = 1;
                                    }

                                    else {
                                        echo "<p> Errreur, cette adresse e-mail a déjà été utilisé, veuillez en choisir une autre. </p>";
                                    }

                                }

                                else {
                                    echo "<p> Errreur, ce pseudo existe déjà, veuillez en choisir un autre. </p>";
                                }
```

```
54.
55.                             }
56.
57.                             else {
58.                                 echo "<p> Errreur, vous devez rentrer un nombre
    de trophé cohérent. </p>";
59.                             }
60.                         }
61.
62.                         else {
63.                             echo "<p> Errreur, vous avez entré(e) une adresse e-
    mail invalide. </p>";
64.                         }
65.
66.                     } else {
67.                         echo "<p> Errreur, vos 2 mot de passe doivent
    correspondent. </p>";
68.                     }
69.                 }
70.
71.                 else {
72.                     echo "<p> Errreur, votre mot de passe doit contenir entre 4
    et 20 caractères. </p>";
73.                 }
74.             } else {
75.                 echo "<p> Errreur, votre pseudo doit contenir 4 à 20 lettres ou
    chiffres. </p>";
76.             }
77.         } else {
78.             echo "<p> Errreur, tout les champs doivent être remplis. </p>";
79.         }
80.     }
81.
82.     else {
83.         //Si on vient de s'inscrire
84.         if (isset($_SESSION['inscriptionStatut']) && $_SESSION['inscriptionStatu
    t'] == 1) {
85.             //On préviens l'utilisateur
86.             echo '<p>Vous êtes bien inscris. Vous pouvez à présent vous
    connecter</p>';
87.             //Plus besoin de prévenir
88.             $_SESSION['inscriptionStatut'] = 0;
89.         }
90.     }
91.
92. }
93.
94. else {
95.     $erreur = "Erreur, vous êtes déjà connecté(e).";
96.     include_once('erreur.php');
97. }
98.
99. include_once("../vue/footerView.php");
100.                 ?>
```

La page d'inscription commence par inclure les fichiers nécessaires à l'inscription. Le fichier sessionCheck permettra d'initialiser une session avec la fonction session_start() seulement s'il n'en existe pas déjà une (pour éviter les erreurs E_NOTICE), cela permettra d'accéder aux variables superglobales $_SESSION.

On inclut également le modèle qui contient les fonctions pour effectuer les requêtes PHP permettant l'inscription, et la vue du menu qui sera importé pour tous les contrôleurs du site.

Ensuite, on vérifie que l'utilisateur n'est pas déjà connecté pour pouvoir s'inscrire, dans le cas contraire, on le prévient. On peut dès lors vérifier que l'utilisateur a bien cliqué sur le bouton permettant d'envoyer ses informations. En effet, une fois les informations envoyées, un tableau associatif $_POST contenant les données va être créé (ces données sont accessibles par un indice qui correspond au nom qu'on a attribué au champ « name » dans la balise input correspondant). Grâce à cela, on peut vérifier que les différentes variables $_POST existent bien (on passe donc bien par le formulaire dans ce cas) et qu'ils ne sont pas vides.

Ces variables sont ensuite traduites en texte grâce à la fonction htmlspecialchars() pour éviter que l'utilisateur puisse entrer du code sous forme de balise, par exemple Fabio afficherait **Fabio** si rien n'était vérifié.

Après ça s'en suit toute une série de vérifications (pseudo/e-mail conforme, longueur du mot de passe, même mot de passe inséré, nombre de trophées conforme, pseudo et e-mail déjà existant) afin de déterminer si les données envoyées peuvent être insérées dans la base de données. On utilise pour cela la fonction preg_match() suivie d'une regex permettant de vérifier la chaine de caractère inséré. Les mots de passe sont bien évidemment toujours hashés avec la fonction sha1() avant d'être stockés dans la base de données.

Étant donné que dans la vue, nous vérifions si les variables $pseudo, $mdp, $mdp2, $mail et $tr existent bien afin d'auto compléter les champs en cas d'erreur de l'utilisateur, lorsque celui-ci s'inscrit, il faut recharger la page afin de ne plus auto compléter les entrées. Cependant pour que l'utilisateur soit prévenu qu'il est bel et bien inscrit il faut vérifier qu'on vient de s'inscrire, on enregistre donc cet état dans une variable de session lorsqu'on s'inscrit et on la réutilise à la ligne 84. Une fois le message affiché, on met l'état 0, cela signifie qu'on ne vient plus d'être inscrit.

inscriptionModele.php

```
1.  <?php
2.  include_once('../modele/modele.php');
3.
4.  function inscription($pseudo,$mdpChiffre,$email,$trophee)
5.  {
6.      $sql = 'INSERT INTO profil (pseudo, mdp, email, trophee) VALUES
    (:pseudo,:mdp,:mail,:tr)';
7.      $param = array('pseudo' => $pseudo, 'mdp' => $mdpChiffre , 'mail' =>$email,
     'tr'=>$trophee);
8.
9.      executeReq($sql,$param);
10. }
11.
12. function checkPseudo($pseudo)
13. {
14.     $sql = 'SELECT pseudo FROM profil WHERE pseudo = :pseudo';
15.     $param = array('pseudo' => $pseudo);
```

```php

    $donnees = executeReq($sql,$param) -> fetch();
    print_r($donnees);
    return $donnees;
}

function checkEmail($email)
{
    $sql = 'SELECT email FROM profil WHERE email = :mail';
    $param = array('mail' => $email);

    $donnees = executeReq($sql,$param)->fetch();
    return $donnees;
}
?>
```

Le principe est toujours le même pour les requêtes préparées, on enregistre la requête dans une variable, ensuite on crée un tableau associatif contenant la valeur du/des paramètres de la requête et on l'exécute. La fonction fetch() permet de récupérer la ligne correspondant à la requête dans un tableau associatif.

6.2. Connexion

Cette page permet la connexion d'un membre, un formulaire est également présenté à l'utilisateur pour qu'il puisse entrer son login et mot de passe.

Connexion.php

```php
<?php
include_once('sessionCheck.php');
include('../modele/connexionModele.php');
include_once('../vue/menuView.php');

if (!isset($_SESSION['droit']) || $_SESSION['droit'] == 0) {
    include('../vue/connexionView.php');

    if (isset($_POST['pseudo']) || isset($_POST['mdp'])) //si je viens du formulaire
        {
        if (!empty($_POST['pseudo']) && !empty($_POST['mdp'])) //Si mdp + login pas vide
            {
            $pseudo    = htmlspecialchars($_POST['pseudo']);
            $mdpEnvoye = sha1($_POST['mdp']);
            //On vérifie les identifiants entré
            $donnees   = connexion($pseudo, $mdpEnvoye);

            if (!empty($donnees)) {
                $_SESSION['droit']  = 1; // On est connecté
                $_SESSION['id']     = $donnees['id'];
                $_SESSION['pseudo'] = $donnees['pseudo'];
                $_SESSION['mail']   = $donnees['email'];
                $_SESSION['tr']     = $donnees['trophee'];
                $_SESSION['carte']  = $donnees['carte'];
                $_SESSION['age']    = $donnees['age'];

                //Redirection vers l'accueil
                header('Location: ../index.php');
```

```
            }

            else {
                session_destroy();
                echo '<p>Combinaison pseudo - mot de passe invalide ! </p>';
            }
        }
    }
}

else {
    $erreur = "Erreur, vous êtes déjà connecté(e).";
    include_once('erreur.php');
}

include_once("../vue/footerView.php");
?>
```

On inclut d'abord les fichiers nécessaires pour la connexion. Si l'utilisateur est déjà connecté, on lui signale. Les données entrées sont également vérifiées avec htmlspecialchars(). Pour vérifier le login et le mot de passe, on va comparer le mot de passe entrée (hashé) avec le pseudo entrée , s'il y a une correspondance dans la base de données, alors on peut se connecter, et on peut mettre à jour les variables de session. Ces variables permettront d'être réutilisé dans d'autres pages pour vérifier qui est l'utilisateur connecté. S'il n'y a pas de correspondance, on le signale à l'utilisateur. Une fois connecté, l'utilisateur est redirigé vers l'index du site web.

connexionModele.php

```
<?php
require_once('../modele/modele.php');

function connexion($pseudo, $mdpEnvoye)
{
    $sql   = 'SELECT * FROM profil WHERE pseudo = :pseudo && mdp = :mdp';
    $param = array(
        'pseudo' => $_POST['pseudo'],
        'mdp' => $mdpEnvoye
    );

    $donnees = executeReq($sql, $param)->fetch();
    return $donnees;
}
?>
```

On utilise également les requêtes préparées pour récupérer les informations relative à la connexion dans la base de donnée.

6.3. Profil

Chaque utilisateur va pouvoir posséder un profil qu'il pourra modifier ou supprimer à tout moment.

Profil.php

```php
<?php
include_once('sessionCheck.php');
include_once('../modele/profilModele.php');
include_once('../vue/menuView.php');
if (isset($_SESSION['droit']) && $_SESSION['droit'] == 1) {
    //Par défaut si on a pas mis d'id, on considère qu'on visionne son profil
    if (!isset($_GET['id'])) {
        $_GET['id'] = $_SESSION['pseudo'];
    }
    $getId   = strval($_GET['id']);
    $donnees = profil($getId);
    if (!empty($donnees)) {
        $userInfo['id']     = $donnees['id'];
        $userInfo['pseudo'] = $donnees['pseudo'];
        $userInfo['mail']   = $donnees['email'];
        $userInfo['tr']     = $donnees['trophee'];
        $userInfo['carte']  = $donnees['carte'];
        $userInfo['age']    = $donnees['age'];

        include_once('../vue/profilView.php');
    }

    else {
        $erreur = "Erreur, ce profil n'existe pas.";
        include_once('erreur.php');
    }
}

else {
    $erreur = "Erreur, vous devez être connecté(e) pour accéder à cette page.";
    include_once('erreur.php');
}

//Je l'inclus ici car autrement si l'utilisateur n'est pas connecté, je n'ai plus de footer
include_once("../vue/footerView.php");

?>
```

Chaque profil contient un ID et on considère ici que l'ID est le nom du pseudo de l'utilisateur, on va donc pouvoir accéder à n'importe quel profil (avec ?id=pseudoDuMembre) d'un membre dans le cas où on est connecté.
Si aucun ID est spécifié, alors on considère qu'on visite son profil grâce à une variable de session enregistrée lors de la connexion. L'ID de l'URL est récupéré à l'aide de la variable $_GET

La fonction strval() va permettre de ne prendre que les string entrés comme ID. Ensuite on va récupérer le profil du membre en fonction de l'ID passé en paramètre dans l'URL . Si la requête n'a rien retourné alors c'est que le profil n'existe pas, on prévient donc l'utilisateur. Dans le cas contraire, on enregistre les informations de l'utilisateur et elles seront réutilisées dans la vue pour afficher les informations du membre.

profilView.php

```php
<?php
include_once("../vue/menuView.php");
?>
<br>
<section>
    <h3>Profil</h3>
    <br>
    <br>
    <table class = "profil">
    <tr>
        <td>Pseudo :</td>
        <td> <?php
echo $userInfo['pseudo'];
?> </td>
    </tr>
    <tr>
        <td>Adresse e-mail :</td>
        <td> <?php
echo $userInfo['mail'];
?> </td>
    </tr>
    <tr>
        <td>Nombre de trophées :</td>
        <td> <?php
echo $userInfo['tr'];
?> </td>
    </tr>
    <tr>
        <td>Age :</td>
        <td> <?php
echo $userInfo['age'];
?> </td>
    </tr>
    <tr>
        <td>Carte favorite :</td>
        <td> <?php
echo $userInfo['carte'];
?> </td>
    </tr>
    <tr>
    </table>


    <?php
if ($_SESSION['pseudo'] == $userInfo['pseudo'] || $_SESSION['pseudo'] == "fabio"
)//TODO si je suis admin je peux modifier
    {
?>
   <a href=<?php
    echo "editProfil.php?id=" . $_GET['id'];
?>> Editer ce profil </a>
    <a href=<?php
    echo "deleteProfil.php?id=" . $_GET['id'];
?>> Supprimer ce compte </a><br>
    <?php
}
?>
```

La vue affichera toujours la possibilité d'éditer ou supprimer un profil si l'utilisateur est le sien ou l'utilisateur est l'administrateur du site, autrement ne l'affiche pas.

6.4. Modification d'un profil

Les membres peuvent mettre à jour leur profil , le principe est assez similaire que l'inscription des comptes utilisateurs.
La principale différence se joue sur le faite que seul l'administrateur ou l'utilisateur à qui appartient le compte est autorisé à modifier son profil et également le type requêtes.

editProfil.php (non complet)

```
1.  //On récupère les informations en fonction de l'ID qu'on va récupérer dans le
    GET
2.      if(!isset($_GET['id'])){
3.          $_GET['id'] = $_SESSION['pseudo'];
4.      }
5.      $getId = strval($_GET['id']);
6.      $donneesProfil = profil($getId);
7.      //On vérifie qu'on a bien le droit d'éditer le profil càd qu'on est
    l'utilisateur du profil ou l'admin
8.      if(!empty($donneesProfil))
9.      {
10.         if($_SESSION['pseudo'] == $donneesProfil['pseudo'] || $_SESSION['pseudo
    '] =="fabio"){
11.             ...
12.          }
13.
14.         else
15.         {
16.             echo "<p>Erreur, vous n'avez pas l'autorisation de modifier ce
    profil. </p>";
17.         }
18.     }
19.
20.     else{
21.         echo "<p>Erreur, ce profil n'existe pas. </p>";
22.     }
23. }
```

Comme pour profil.php, on vérifie l'ID inséré. La variable $donneesProfil va contenir les informations du profil qu'on veut éditer, cette variable sera réutiliser dans la suite du code pour pouvoir éditer le compte, elle contient les informations avant la modification. Une fois la modification effectué, on met à jour les variables de session dans le cas où l'utilisateur qui a été modifié est celui qui est connecté et non l'administrateur.

6.5. Suppression d'un profil

Les membres peuvent également supprimer leur compte.

deleteProfil.php

```
1.  <?php
2.  include_once('sessionCheck.php');
3.  require_once('../modele/profilModele.php');
4.  include_once('../vue/menuView.php');
```

```php

if (isset($_SESSION['droit']) && $_SESSION['droit'] == 1) {
    //On récupère les informations en fonction de l'ID qu'on va récupérer dans le GET
    if (!isset($_GET['id'])) {
        $_GET['id'] = $_SESSION['pseudo'];
    }
    $getId         = strval($_GET['id']);
    $donneesProfil = profil($getId);
    if (!empty($donneesProfil)) {
        //On vérifie qu'on a bien le droit d'éditer le profil càd qu'on est l'utilisateur du profil ou l'admin
        if ($_SESSION['pseudo'] == $donneesProfil['pseudo'] || $_SESSION['pseudo'] =="fabio") {
            deleteProfil($donneesProfil['pseudo']); //A TESTER
            include_once('../vue/deleteProfilView.php');
            //Si je supprime mon pseudo, je me déconnecte
            if ($_SESSION['pseudo'] == $donneesProfil['pseudo']) {
                session_destroy();
            }
        }

        else {
            echo "<p>Erreur, vous n'avez pas l'autorisation de supprimer ce profil.</p>";
        }
    }

    else {
        echo "<p>Erreur, Ce profil n'existe pas.</p>";
    }
}

else {
    $erreur = "Erreur, vous devez être connecté(e) pour accéder à cette page.";
    include_once('erreur.php');
}

include_once('../vue/footerView.php');
?>
```

On effectue encore une fois les vérifications autour des ID introduits dans l'URL.
Si on a bien l'autorisation et que le profil existe, on peut le supprimer, une fois cela fait, l'utilisateur est déconnecté avec la fonction session_destroy()

profilModele.php

```php
<?php
require_once('../modele/modele.php');

function profil($getid)
{
    $getid = strval($getid);
    $sql = 'SELECT * FROM profil WHERE pseudo = ?';
    $param = array($getid);

    return executeReq($sql,$param)->fetch();
}

function editProfil($pseudo,$mdpChiffre,$email,$trophees,$age,$carte,$id)
{
```

```
    $sql = 'SET FOREIGN_KEY_CHECKS=0; UPDATE profil SET pseudo = :pseudo, mdp = :mdp, email = :mail,
            trophee = :tr, age = :age, carte = :carte WHERE id = :id; SET FOREIGN_KEY_CHECKS=1;';
    $param = array('pseudo' => $pseudo, 'mdp' => $mdpChiffre , 'mail' => $email,
                    'tr' => $trophees, 'age' => $age, 'carte' => $carte, 'id' => $id);
    return executeReq($sql,$param) -> fetch();
}

function checkProfil($pseudo,$id)
{
    $sql = 'SELECT pseudo FROM profil WHERE pseudo = :pseudo AND id != :id';
    $param = array('pseudo' => $pseudo, 'id' => $id);
    return executeReq($sql,$param)->fetch();
}

function checkEmail($email,$id)
{
    $sql = 'SELECT email FROM profil WHERE email = :mail AND id != :id';
    $param = array('mail' =>$email, 'id' =>$id);
    return executeReq($sql,$param)->fetch();
}

function deleteProfil ($pseudo)
{
    $sql = 'SET FOREIGN_KEY_CHECKS=0; DELETE FROM profil WHERE pseudo = :pseudo; SET FOREIGN_KEY_CHECKS=1;';
    $param = array('pseudo' => $pseudo);
    executeReq($sql,$param);
}
?>
```

Le modèle est similaire à celui de l'inscription à l'exception qu'il contient les fonctions avec les requêtes UPDATE et DELETE pour mettre à jour et supprimer des comptes.

Étant donné qu'on souhaite quand même garder les decks des utilisateurs une fois le compte mis à jour ou supprimé, on désactive la contrainte de clé étrangère lorsqu'on met à jour ou supprime un compte à l'aide de l'instruction SET FOREIGN_KEY_CHECKS=0 ,et on la réactive ensuite.

6.6. Ajout d'un deck

Comme dit précédemment, les membres peuvent ajouter leurs decks pour aider les autres membres.

ajoutDeck.php

```
<?php
include_once('sessionCheck.php');
require_once('../modele/ajoutDeckModele.php');
include_once('../vue/menuView.php');

if (isset($_SESSION['droit']) && $_SESSION['droit'] == 1) {
    if (isset($_POST['type']) || isset($_POST['cout']) || isset($_POST['carte1']) || isset($_POST['carte2']) || isset($_POST['carte3']) ||isset($_POST['carte4']
```

```php
    ) || isset($_POST['carte5']) || isset($_POST['carte6']) || isset($_POST['carte7'
    ]) || isset($_POST['carte8'])) {
        include_once('../vue/ajoutDeckView.php');

        if (!empty($_POST['type']) && !empty($_POST['cout']) && !empty($_POST['c
    arte1']) && !empty($_POST['carte2']) &&!empty($_POST['carte3']) && !empty($_POST
    ['carte4']) && !empty($_POST['carte5']) && !empty($_POST['carte6']) && !empty($_
    POST['carte7']) &&!empty($_POST['carte8'])) {
            $nom         = htmlspecialchars($_POST['nom']);
            $type        = htmlspecialchars($_POST['type']);
            $cout        = htmlspecialchars($_POST['cout']);
            $carte1      = htmlspecialchars($_POST['carte1']);
            $carte2      = htmlspecialchars($_POST['carte2']);
            $carte3      = htmlspecialchars($_POST['carte3']);
            $carte4      = htmlspecialchars($_POST['carte4']);
            $carte5      = htmlspecialchars($_POST['carte5']);
            $carte6      = htmlspecialchars($_POST['carte6']);
            $carte7      = htmlspecialchars($_POST['carte7']);
            $carte8      = htmlspecialchars($_POST['carte8']);
            $description = htmlspecialchars($_POST['description']);

            if (empty($nom) || preg_match("#^[a-zA-Z0-9\'\.\,\;\-\_\+\-
    \!\?\&\é\@\"\'\#\è\à\ç\€\$\ù\%\:\)\(\* ]{1,40}$#", $nom)) {
                if (preg_match("#^[2-8]*.?[1-9]+$#", $cout)) {
                    if (strlen($description) <= 400) {
                        ajoutDeck($_SESSION['pseudo'], $nom, $type, $cout, $cart
    e1, $carte2, $carte3, $carte4, $carte5, $carte6, $carte7,$carte8, $description);

                        echo "<p>Le deck " . $nom . " a bien été ajouté à la
    liste.</p>";
                    }

                    else {
                        echo "<p> Erreur, la taille de la description du deck
    doit être inférieur à 400 caractères </p>";
                    }
                }

                else {
                    echo "<p> Erreur, le coût en elixir introduit n'est pas
    cohérent. </p>";
                }
            }

            else {
                echo '<p> Erreur, vous devez entrez un nom comportant uniquement
    lettre et chiffre </p>';
            }
        }

        else {
            echo '<p> Erreur, vous devez remplir les champs nécessaire </p>';
        }
    }

    else {
        include_once('../vue/ajoutDeckView.php');
    }
}

else {
    $erreur = "Erreur, vous devez être connecté(e) pour accéder à cette page.";
    include_once('erreur.php');
}

include_once('../vue/footerView.php');
```

```
63. ?>
```

Le principe est proche de l'inscription, une fois les fichiers importés, on vérifie qu'on a bien envoyé les informations du deck depuis le formulaire, on procède ensuite à une série de vérification pour que les données enregistrées soient logiques par rapport au jeu.

6.7. Liste des decks

Une fois les decks ajoutés, les utilisateurs peuvent visionner les visionner. On considère qu'on ne va afficher maximum 5 decks par page pour permettre une meilleure lecture pour l'utilisateur.

listeDeck.php

```php
<?php
include_once('sessionCheck.php');
include_once('../vue/menuView.php');

if (isset($_SESSION['droit']) && $_SESSION['droit'] == 1) {
    include_once('../modele/listeDeckModele.php');
    $countAll = countAllDeck();
    $nbPage   = ceil($countAll[0] / 5);

    if (isset($_GET['page']) && !empty($_GET['page']) && $_GET['page'] > 0 &&$_GET['page'] <= $nbPage) {
        $_GET['page'] = intval($_GET['page']);
        $depart       = ($_GET['page'] - 1) * 5;
        $donnees      = listeDeck($depart);
        $nbLigne      = countDeck($depart);
        $pageCourante = $_GET['page'];

        include_once('../vue/listeDeckView.php');

    }

    else {
        echo "<p> Erreur, page inexistante </p>";
    }
} else {
    $erreur = "Erreur, vous devez être connecté(e) pour accéder à cette page.";
    include_once('erreur.php');
}

include_once('../vue/footerView.php');

?>
```

Une fois que nous sommes connectés, on peut visiter la liste des decks. La variable $countAll va contenir la nombre total de decks postés par les utilisateurs, la variable $nbPage va contenir un calcul permettant de déterminer le nombre de pages nécessaires pour afficher l'ensemble de decks.

Ensuite, on peut vérifier l'ID de la page entré, le calcul de $nbPage à l'avance prend son sens pour cette vérification. On va également stocker dans une variable

l'endroit où la requête va démarrer pour récupérer la liste des decks. On stocke également le nombre de decks affiché par page dans $nbLigne (car la dernière page n'affichera pas forcemment 5 decks) ainsi que la page courante où l'utilisateur se situe. Toutes ces variables serviront pour le modèle, mais aussi la vue.

listeDeckView.php

Cette page contiendra l'affichage des decks avec la possibilité de changer de page, elle contiendra plus de code PHP que les autres vues.
Le site contient également une page permettant de rechercher un deck en fonction d'une catégorie, cette page est assez simple et s'occupe d'appeler les fonctions du modèle (listeDeckRecherche($type) et countDeckRecherche($type)), son affichage se fait sur cette même vue.

```
1.  <section>
2.  <h2> Decks de la communauté </h2>
3.
4.  <?php
5.      $i = 0;
6.      if(isset($_POST['type'])){ //Si je viens du form de la recherche
7.          $type = htmlspecialchars($_POST['type']);
8.
9.          //Si on a des lignes alors c'est qu'il y a des decks de cette catégorie
    sinon on préviens l'utilisateur
10.         if($nbLigne > 0){
11.             while ($i < $nbLigne){
12.                 if($donnees[$i]['type'] == $type){ //Si on a une correspondance de
    catégorie pour la recherche
13.                     ?>
14.                     <div class="rescensement">Pseudo: <a
    href="../controleur/profil.php?id=<?php echo $donnees[$i]['pseudo']; ?>"><?php ech
    o$donnees[$i]['pseudo']; ?></a></div>
15.                     <div
    class="rescensement">Nom: <?php echo $donnees[$i]['nom']; ?></div>
16.                     <div class="rescensement">Type
    : <?php echo $donnees[$i]['type']; ?></div>
17.                     <div class="rescensement">Coût en
    élixir: <?php echo $donnees[$i]['cout']; ?></div>
18.                     <div class="rescensement">Carte
    1: <?php echo $donnees[$i]['carte1']; ?></div>
19.                     <div class="rescensement">Carte
    2: <?php echo $donnees[$i]['carte2']; ?></div>
20.                     <div class="rescensement">Carte
    3: <?php echo $donnees[$i]['carte3']; ?></div>
21.                     <div class="rescensement">Carte
    4: <?php echo $donnees[$i]['carte4']; ?></div>
22.                     <div class="rescensement">Carte
    5: <?php echo $donnees[$i]['carte5']; ?></div>
23.                     <div class="rescensement">Carte
    6: <?php echo $donnees[$i]['carte6']; ?></div>
24.                     <div class="rescensement">Carte
    7: <?php echo $donnees[$i]['carte7']; ?></div>
25.                     <div class="rescensement">Carte
    8: <?php echo $donnees[$i]['carte8']; ?></div>
26.                     <div
    class="rescensement">Description: <?php echo $donnees[$i]['description']; ?></div>
27.
```

```
                    <hr>

                    <?php
                }
                $i++;
            }
        }

        else{
            echo "<p> Aucun deck ne correspond à cette catégorie. </p>";
        }
    }

    else{
        if($nbLigne > 0){
            while ($i < $nbLigne){
                ?>
                <div class="rescensement">Pseudo: <a
href="../controleur/profil.php?id=<?php echo $donnees[$i]['pseudo']; ?>"><?php ech
o$donnees[$i]['pseudo']; ?></a></div>
                <div
class="rescensement">Nom: <?php echo $donnees[$i]['nom']; ?></div>
                <div class="rescensement">Type
: <?php echo $donnees[$i]['type']; ?></div>
                <div class="rescensement">Coût en
élixir: <?php echo $donnees[$i]['cout']; ?></div>
                <div class="rescensement">Carte
1: <?php echo $donnees[$i]['carte1']; ?></div>
                <div class="rescensement">Carte
2: <?php echo $donnees[$i]['carte2']; ?></div>
                <div class="rescensement">Carte
3: <?php echo $donnees[$i]['carte3']; ?></div>
                <div class="rescensement">Carte
4: <?php echo $donnees[$i]['carte4']; ?></div>
                <div class="rescensement">Carte
5: <?php echo $donnees[$i]['carte5']; ?></div>
                <div class="rescensement">Carte
6: <?php echo $donnees[$i]['carte6']; ?></div>
                <div class="rescensement">Carte
7: <?php echo $donnees[$i]['carte7']; ?></div>
                <div class="rescensement">Carte
8: <?php echo $donnees[$i]['carte8']; ?></div>
                <div
class="rescensement">Description: <?php echo $donnees[$i]['description']; ?></div>

                <?php
                //Si je suis admin je peux modifier
                if($_SESSION['pseudo'] == "fabio"){
                ?>
                    <form
action="../controleur/editDeck.php?id=<?php echo $donnees[$i][0];?>"
method="post">
                    <input type="hidden" name="id"
value="<?php echo $donnees[$i][0]?>" />
                    <input type="submit" value="Modifier ce deck"/></form>

                    <form action="../controleur/deleteDeck.php" method="post">
                    <input type="hidden" name="id"
value="<?php echo $donnees[$i][0]?>" />
                    <input type="submit" value="Supprimer ce deck"/></form>
                <?php
                }
                ?>
                <!---On va envoyer l'id du deck avec une methode post si on clique
sur modifier -->
```

```
74.                 <hr>
75.                 <?php
76.                 $i++;
77.             }
78.         }
79.
80.         else{
81.             echo "<p>Aucun deck n'a encore été ajouté</p>";
82.         }
83.
84.         //On affiche ici la page suivante si on retourne plus de 5 résultats
85.         echo 'Page: ';
86.         for($i = 1; $i <= $nbPage; $i++){
87.             //Si on est sur la même page que la page courante, on affiche le
    numéro de la page sans pouvor recliquer dessus
88.             if($i == $pageCourante){
89.                 echo $i.' ';
90.             }
91.
92.             else{
93.                 //Le numéro de page ne correspond pas à la page courante, elle est
    devient cliquable
94.                 echo '<a href="listeDeck.php?page='.$i.'">'.$i.'</a>';
95.             }
96.         }
97.     }
98. ?>
```

On vérifie au début si on vient du formulaire, alors c'est qu'on a effectué une recherche, dans ce cas $nbLigne contient le nombre de decks en fonction de la catégorie entrée, on fait donc une boucle en utilisant de ce nombre et tant qu'on ne l'a pas atteint, affiche les decks.

Si on ne vient pas du formulaire alors c'est qu'on cherche tous les decks, on réalise le même principe que pour la recherche. Cependant, on laisse la possibilité à l'administrateur de modifier ou supprimer chaque deck rencontré. On affiche également les différentes pages à partir de la ligne 84, la page courante est grisée tandis que les autres pages disponibles sont cliquables.

listeDeckModele.php

```
1.  <?php
2.  require_once('../modele/modele.php');
3.
4.  function joinTable($depart)
5.  {
6.      $sql = 'SELECT * FROM listeDeck INNER JOIN categorie ON listeDeck.type =
    categorie.id
7.     ORDER BY cout LIMIT ' . $depart . ',5';
8.      $req = executeQueryReq($sql);
9.      return $req;
10. }
11.
12. function joinTableRecherche($categorie)
13. {
```

```
14.     $sql    = 'SELECT * FROM listeDeck INNER JOIN categorie ON listeDeck.type =
    categorie.id WHERE
15.    categorie.type = ? ORDER BY cout';
16.     $param = array(
17.         $categorie
18.     );
19.     return executeReq($sql, $param);
20. }
21. function listeDeck($depart)
22. {
23.     $donnees = joinTable($depart)->fetchAll();
24.     return $donnees;
25. }
26.
27. function listeDeckRecherche($categorie)
28. {
29.     $donnees = joinTableRecherche($categorie)->fetchAll();
30.     return $donnees;
31. }
32.
33. function countDeck($depart)
34. {
35.     $nbLigne = joinTable($depart)->rowCount();
36.     return $nbLigne;
37. }
38.
39.
40. function countDeckRecherche($categorie)
41. {
42.     $nbLigne = joinTableRecherche($categorie)->rowCount();
43.     return $nbLigne;
44. }
45.
46. function countAllDeck()
47. {
48.     $sql = 'SELECT COUNT(*) FROM listeDeck';
49.     $req = executeQueryReq($sql);
50.     return $req->fetch();
51. }
52.
53. ?>
```

La fonction joinTable() du modèle va permettre de faire une jointure entre la table catégorie et listeDeck afin de pouvoir lier un deck à sa catégorie « en français » étant donné que l'ID dans la table catégorie est en correspondance avec l'ID qu'on entre dans la table listeDeck à la colonne type.

La clause LIMIT est utilisée pour spécifier le nombre maximum de résultats que l'ont souhaite récupérer afin d'obtenir le système de pagination pour la liste des decks. Le premier nombre $depart est l'OFFSET et indique à partir de quel endroit de la table on récupère les données. Le second chiffre (5) spécifie le nombre de lignes qu'on récupère.

La fonction listeDeck() va ensuite permettre récupérer toutes les lignes correspondantes aux decks dans un tableau associatif à 2 dimensions à l'aide de la fonction fetchAll().

La fonction countDeck() permet de compter le nombre de lignes que la fonction listeDeck() a retourné. Elle est utile pour l'affichage des decks dans la vue.

6.8. Voir ses decks

L'utilisateur a également la possibilité de visionner seulement ses decks, le principe est le même que pour la liste des decks sauf qu'on affiche les résultats sur une seule page et qu'on laisse donc la possibilité à l'utilisateur sur cette page de modifier ou supprimer son deck (cette possibilité est seulement disponible pour l'administrateur dans la liste de tous les decks). Pour ne pas surcharger la vue de la liste des decks, on en crée une nouvelle qui est très similaire à cette dernière.

6.9. Éditer un deck

La modification d'un deck est également proche de l'ajout d'un deck, sauf qu'on va pouvoir modifier le deck en fonction de son ID.

editDeck.php (non complet)

```
1.  if(isset($_GET['id']) && $_GET['id'] >= 0){
2.          $getId = intval($_GET['id']);
3.           //On check si l'utilisateur connecté  modifie son deck et pas celui
    d'un autre si jamais on modifie l'id
4.           $donnees = recupDeck($getId);
5.           if(!empty($donnees)){
6.               if($donnees['pseudo'] == $_SESSION['pseudo'] || $_SESSION['pseudo']
     == "fabio"){
7.                   ...
8.               }
9.                 else{
10.                  echo "<p>Erreur, vous n'avez pas le droit de modifier ce
    deck.</p>";
11.              }
12.
13.          }
14.          else{
15.              echo "<p> Erreur, ce deck n'existe pas.</p>";
16.          }
17.
18.      }
19.  else{
20.         echo "<p> Erreur, ce deck n'existe pas, vérifiez l'identifiant entré.
    </p>";
21.      }
```

Les différences notables sont proches de ceux entre l'inscription et la modification de profil. On vérifie d'abord l'ID passé dans l'URL, s'il n'est pas cohérent on le signale. Ensuite on va pouvoir récupérer le deck correspondant à l'ID qu'on envoie dans l'URL. S'il y a une correspondance alors on vérifie bien que le deck qu'on va modifier appartient à l'utilisateur qui le modifie ou bien qu'il soit l'administrateur.

6.10. Supprimer un deck

Le membre ou l'administrateur peut décider aussi de supprimer un deck.

deleteDeck.php

```php
<?php
include_once('sessionCheck.php');
require_once('../modele/deleteDeckModele.php');
include_once('../vue/menuView.php');

if (isset($_SESSION['droit']) && $_SESSION['droit'] == 1) {
    //Post pour éviter que l'on puisse se tromper dans barre de recherche
    if (isset($_POST['id'])) {
        $id      = htmlspecialchars($_POST['id']);
        $donnees = recupDeck($id);
        //Check si jamais on a modifié l'id dans le form et qu'il n'appartient pas au pseudo connecté
        if ($donnees['pseudo'] == $_SESSION['pseudo'] || $_SESSION['pseudo'] == "fabio") {
            deleteDeck($id);
            header("Location: listeDeck.php?page=1");
        }

        else {
            echo "<p>Erreur, vous n'avez pas le droit de supprimer ce deck</p>";
        }
    }

    else {
        echo '<p> Erreur, impossible de modifier ce deck </p>';
    }
}

else {
    $erreur = "Erreur, vous devez être connecté(e) pour accéder à cette page.";
    include_once('erreur.php');
}

include_once('../vue/footerView.php');
?>
```

Le principe est simple, on récupère l'ID envoyé depuis le formulaire caché de la vue (hidden) de la liste de ses decks (ou de tous les decks pour l'administrateur) ? Ensuite avec cet ID on appelle une fonction dans le modèle pour supprimer le deck correspondant à cet ID.

6.11. Contact

Les membres et les visiteurs peuvent contacter l'administrateur du site via un formulaire de contact.

contact.php

```php
<?php
include_once('sessionCheck.php');
include_once('../vue/menuView.php');

if (isset($_POST['nom']) || isset($_POST['sujet']) || isset($_POST['mail']) || isset($_POST['message'])) {
    //Si je suis connecté, on va afficher déjà le nom et le mail de l'utilsateur
    if (!empty($_POST['nom']) && !empty($_POST['sujet']) && !empty($_POST['mail']) && !empty($_POST['message']) && !empty($_POST['mail'])) {
        $nom     = htmlspecialchars($_POST['nom']);
        $mail    = htmlspecialchars($_POST['mail']);
        $sujet   = htmlspecialchars($_POST['sujet']);
        $message = htmlspecialchars($_POST['message']);
        include_once('../vue/contactView.php');

        if (preg_match("#^[a-zA-Z0-9\'\.\,\;\-\_\+\-\!\?\&\é\@\"\'\#\è\à\ç\€\$\ù\%\:\)\(\* ]{2,20}$#", $nom)) {
            if (preg_match("#^[a-zA-Z0-9\'\.\,\;\-\_\+\-\!\?\&\é\@\"\'\#\è\à\ç\€\$\ù\%\:\)\(\* ]{2,40}$#", $sujet)) {
                if (preg_match("#^[a-z0-9._-]+@[a-z0-9._-]{2,}\.[a-z]{2,4}$#", $mail)) {
                    $mail = 'fabio181097@gmail.com';
                    if (!preg_match("#^[a-z0-9._-]+@(hotmail|live|msn|outlook).[a-z]{2,4}$#", $mail)) // filtrage des serveurs qui posent problème
                        {
                        $passage_ligne = "\r\n";
                    } else {
                        $passage_ligne = "\n";
                    }

                    $message_txt = $_POST['message'];

                    // création de boundary
                    $boundary = "-----=" . md5(rand());

                    $sujet = "Mail de " . $nom . " - " . $_POST['sujet'];

                    // création du header du mail
                    $header = "From:" . $nom . "<" . $_POST['mail'] . ">" . $passage_ligne;
                    $header .= "Reply-to:<" . $_POST['mail'] . ">" . $passage_ligne;
                    $header .= "MIME-Version: 1.0" . $passage_ligne;
                    $header .= "Content-Type: multipart/alternative;" . $passage_ligne . " boundary=\"$boundary\"" . $passage_ligne;

                    //  création du message
                    $message = $passage_ligne . "--" . $boundary . $passage_ligne;

                    $message .= "Content-Type: text/plain; charset=\"UTF-8\"" . $passage_ligne;
                    $message .= "Content-Transfer-Encoding: 8bit" . $passage_ligne;
                    $message .= $passage_ligne . $message_txt . $passage_ligne . $passage_ligne . "Mail de " . $nom . " : " . $_POST['mail']. $passage_ligne;

                    $message .= $passage_ligne . "--" . $boundary . "--" . $passage_ligne;

                    // envoi du mail
                    if (mail($mail, $sujet, $message, $header)) {
                        //On rafraichit la page pour prendre en compte le nettoyage des entrées
```

```
50.                         echo '<meta http-equiv="refresh"
    content="0;URL=contact.php">';
51.                         //On a envoyé le mail, on enregistre ça dans une variable
    pour prévenir l'utilisateur
52.                         $_SESSION['contactStatut'] = 1;
53.
54.                     } else {
55.                         echo "<p>Echec, de l'envoi, réessayez plus tard.</p>";
56.                     }
57.                 } else {
58.                     echo "<p>Votre adresse mail est incorrect, veuillez
    vérifier.</p>";
59.                 }
60.             } else {
61.                 echo "<p>Le sujet de votre mail doit contenir 2 à 40 lettres ou
    chiffres.</p>";
62.             }
63.         } else {
64.             echo "<p>Votre nom doit contenir 2 à 20 lettres ou chiffres.</p>";
65.         }
66.
67.     } else {
68.         echo "<p>Remplissez tous les champs.</p>";
69.     }
70. }
71.
72. else {
73.     //Si on est déjà connecté, alors on préremplis le nom et le mail
74.     if (isset($_SESSION['droit']) && $_SESSION['droit'] == 1) {
75.         $nom  = $_SESSION['pseudo'];
76.         $mail = $_SESSION['mail'];
77.     }
78.     include_once('../vue/contactView.php');
79.
80.     //Si on a envoyé le mail, on préviens l'utilisateur
81.     if (isset($_SESSION['contactStatut']) && $_SESSION['contactStatut'] == 1) {
82.         echo "<p>Votre message a bien été envoyé, l'administrateur vous répondra
    d'ici maximum 48 heures.</p>";
            $_SESSION['contactStatut'] = 0 ;
83.     }
84. }
85.
86. include_once('../vue/footerView.php');
87. ?>
```

La vue de cette page est un formulaire de contact basique avec le champ nom, sujet, e-mail et message. Lorsque l'utilisateur est un membre du site, on auto complète les champs.

Le début du code ressemble à tous les autres, on importe bien les fichiers nécessaires, on vérifie les entrées, on utilise la fonction htmlspecialchars() et on vérifie les données encore avec la fonction pregmatch() et une regex.

Ensuite, on va définir en fonction de l'adresse e-mail d'envoi si le passage à la ligne sera '\r\n' ou '\n'. Suite à cela le message est mis en forme et les configurations nécessaires sont établies. Dans le header la ligne « Reply-to » est ajoutée permettant au destinataire de répondre directement sur l'adresse de l'expéditeur et non sur l'adresse utilisée pour l'envoi.

Une fois le message envoyé, on informe l'utilisateur et on efface les champs du formulaire, dans le cas contraire on affiche un message d'erreur et les champs sont conservé.

Afin d'envoyer des mails en local avec Wamp, un serveur mail a été nécessaire, j'ai utilisé Fake Sendmail qui émule la commande Unix sendmail nécessitant un serveur SMTP. Le fichier sendmail.ini a été configuré pour permettre l'envoie des mails et les fichiers de l'application ont être placé dans le répertoire C:/Wamp/sendmail.

7. Sécurité

Concernant la sécurité globale du site, les variables de session ont été utilisées. En effet, elles comportent des données relatives à la session en cours et sont stockées côté serveur, elles sont donc sécurisées et permettent de déterminer quel utilisateur est connecté. Lorsque le site utilise les ID contenus dans l'URL, ces ID sont toujours vérifiés par rapport à leur syntaxe, mais il y a également une vérification pour déterminer si l'utilisateur a bien le droit de consulter la page en question grâce aux variables de session.

Toutes les données entrées par l'utilisateur au travers des formulaires sont traitées par la fonction htmlspecialchars() pour éviter les instructions malveillantes (faille XSS). De plus les mots de passe stockés dans la base de données sont toujours hashés avec la fonction sha1().

L'accès à la base de données se fait à l'aide du compte « root » et un mot de passe a été ajouté sur ce compte pour protéger l'accès aux données.

Enfin, l'interface de connexion utilisée pour se connecter à la base de données est PDO qui possède certains avantages non négligeables en matière de sécurité. De plus, les requêtes préparées ont été utilisées pour minimiser les risques d'injections SQL.

8. Améliorations

Quelques améliorations au niveau de l'interface et du développement front-end du site aurait pu être envisagé, cependant le projet ayant pour but de se perfectionner au langage PHP, je ne m'y suis pas trop attardé.

Le respect total du design pattern MVC ainsi qu'un développement orienté objet du site aurait permis de structurer encore mieux le code. Cependant j'ai préféré consacrer plus de temps au développement des fonctionnalités du site afin de donner plus de sens à l'aspect communautaire du site.

9. Conclusion

Pour conclure, le site est fonctionnel ainsi que toutes ses fonctionnalités. Il est sécurisé et respecte bien les engagements énoncés dans le cahier des charges. Quelques ajouts ont néanmoins été effectués tels que le système de pagination pour la liste des decks ainsi que l'autocomplétion des champs dans les formulaires.

Ce projet m'aura permis d'en apprendre plus sur le langage PHP et le développement back-end, mais aussi sur la sécurité des sites Web.