Authentification

Nous voilà avec un site e-commerce relativement modulable. Néanmoins pour qu'il puisse être déployé sur Internet nous devons absolument limiter l'accès de certaines pages à certaines personnes.

Ce chapitre se découpera donc en deux parties. Nous allons voir ici comment réaliser un système d'authentification pour lier une identité à une personne. Dans le prochain chapitre, nous aborderons plutôt la sécurisation des accès aux pages d'administration du site.

Pour réaliser ce système, nous nous appuierons d'un concept d'authentification simple. C'est-à-dire que pour se connecter l'utilisateur aura besoin d'un identifiant et d'un mot de passe. L'identifiant sera dans notre cas une adresse mail.

1. Découverte des sessions

Pour pouvoir créer un système d'authentification, nous allons devoir utiliser une des variables globales mises à disposition par PHP : $_SESSION

La variable session est destinée à stocker des valeurs pendant toute la durée de navigation d'un utilisateur. Ainsi, il est possible d'enregistrer une valeur qui sera accessible depuis n'importe quelle page du site.

La variable de session aura donc toute son utilité pour notre système d'authentification puisqu'elle sera capable de se souvenir d'un utilisateur qui se serait connecté au site. Nous pourrons également personnaliser l'affichage en fonction des informations de chaque utilisateur.

Sachez que la variable de session peut être utilisée pour beaucoup de fonctionnalité telle qu'un système de panier, un système de message d'erreur et d'information avancé, etc.

Travail à faire :

- Ajouter une nouvelle page ayant pour route « login ».
- Ajouter une deuxième page ayant pour route « register »
- Se limiter simplement au contrôleur, il n'est pas nécessaire de créer une vue pour le moment.

Nous allons maintenant observer le comportement de la variable. Pour cela, ajoutez dans la fonction du contrôleur le code suivant :

```
var_dump($_SESSION) ;
```

Pour rappel, la fonction « var_dump » permet de « débuguer » une fonction ou une variable PHP. Dans le cas présent, nous souhaitons « débuguer » la variable globale « $_SESSION ». Vous devez normalement obtenir une erreur, pourtant la variable de session fait bien partie de la liste des variables globales. Cela est en fait lié à une particularité des sessions. En nous référant à la documentation, nous remarquons qu'il est nécessaire d'indiquer à PHP qu'il doit gérer les sessions.

Il est possible de demander l'activation des sessions grâce à la fonction « session_start() ». Attention, quelques règles sont à respecter pour utiliser cette fonction. La fonction doit notamment être appelée :

- Obligatoirement avant toute autre exécution d'instructions PHP
- Sur chaque page où l'on souhaite accéder aux variables de session
- Une seule fois dans l'exécution d'une requête HTTP

Par rapport à l'architecture de l'application, vous allez pouvoir insérer cette fonction dans le fichier « index.php » du dossier « public ».

```
<?php

  // Démarrage de session
  session_start();


  ...
```

En rechargeant la page, vous devriez maintenant avoir accès à la variable. À savoir que « $_SESSION » est un tableau associatif que nous allons pouvoir utiliser pour stocker n'importe quelle valeur.

Exemple :
Dans le contrôleur de la route « register », adapter le code en suivant l'exemple ci-dessous :

```
function registerController($twig, $db) {
      $_SESSION['test'] = "Une valeur stockée";
 }
```

Accéder à la page « register » puis retourner sur la page « login ».

Vous devriez voir apparaitre la valeur associative « test ». Vous pouvez même tester de débuguer la variable « $_SESSION » sur d'autres pages, le résultat serait le même.

2. Création des tables dans la base de données

Maintenant que vous maitrisez le fonctionnement de la variable de session, vous allez mettre en place le système d'authentification.

De la même manière que nous avons fait pour le système de produit, vous allez créer les tables nécessaires dans la base de données. Pour le réaliser, vous allez avoir besoin de deux tables :

- La table « shop_roles » qui va permettre de conserver les différents rôles de l'application. Cette table se constituera de deux colonnes, dont une consacrée à l'ID et la deuxième pour le libellé. Lorsque la valeur de l'ID sera égale à 1, l'utilisateur sera considéré comme un « Client ». Lorsqu'il aura la valeur 2, l'utilisateur sera identifié comme un administrateur.
- La table « shop_users » pour stocker les informations de l'utilisateur. Elle accueillera 6 colonnes : un ID, un email, un nom, un prénom et l'ID du rôle.

Voici en détail la table « shop_roles » :

Nom du champ	Type	Taille	Indexe
Id	Int	-	Clé primaire et auto-incrément
label	Varchar	255	-

Et la table « shop_users » :

Nom du champ	Type	Taille	Indexe
Id	Int	-	Clé primaire et auto-incrément
email	Varchar	255	Unique
password	Varchar	255	-
lastname	Varchar	255	-
firstname	Varchar	255	-
idRole	int	-	-

Remarque : « password » est nécessairement une longue chaine de caractères puisqu'il sera « haché ». C'est-à-dire qu'il sera transformé en une chaine non compréhensible afin que personne ne puisse connaitre le mot de passe des utilisateurs.

Travail à faire :

- Créer les différentes tables en respectant les indications données
- Lier les deux tables entre elles grâce à l'intermédiaire « idRole » de la table « shop_users »
- Ajouter les deux rôles dans la table « shop_roles » en suivant les indications fournies

Information :

Vous pouvez vérifier que le lien s'est bien réalisé en consultant l'outil « concepteur » de PhpMyAdmin.

3. Création des formulaires

Vous allez maintenant pouvoir attaquer le système d'authentification. À savoir qu'un système d'authentification se consacre à l'inscription, la connexion et la déconnexion d'un utilisateur.

Côté inscription, l'utilisateur va devoir renseigner les informations suivantes :

- L'adresse email sera bien évidemment unique.
- Le mot de passe qui devra être validé une deuxième fois par l'utilisateur qui s'inscrit.
- Le nom et le prénom de l'utilisateur

Au total, le formulaire sera constitué de 5 champs.

Travail à faire :

- À l'aide de Bootstrap, réaliser le formulaire d'inscription dans une vue Twig qui se nommera « register.html.twig ».
- Afficher le formulaire en adaptant le fichier « RegisterController.php »
- Faire en sorte de récupérer les informations du formulaire lorsqu'il est envoyé. Ces informations seront stockées dans les variables : $email, $password, $passwordConfirm, $lastname, $firstname

Côté connexion, le formulaire se constitue seulement de deux champs :

- L'adresse email de l'utilisateur
- Le mot de passe de l'utilisateur

Travail à faire :

- À l'aide de Bootstrap, réaliser le formulaire d'inscription dans une vue Twig qui se nommera « login.html.twig ».
- Afficher le formulaire en adaptant le fichier « LoginController.php »
- Faire en sorte de récupérer les informations du formulaire lorsqu'il est envoyé. Ces informations seront stockées dans les variables : $email, $password

4. Gestion de l'inscription

Nous allons maintenant pouvoir lier le formulaire d'inscription avec notre base de données et lui permettre d'ajouter un nouvel utilisateur à chaque fois qu'il sera envoyé.

Pour pouvoir inscrire un utilisateur, nous allons avoir besoin d'un nouveau modèle pour la table « shop_users ».

Travail à faire :

- Créer le nouveau modèle « UserModel.php » dans le dossier « model »
- Écrire le code de la fonction « saveUser() » permettant l'ajout d'un utilisateur dans la table « shop_users »
- Écrire le code de la fonction « getOneUserCredentials() » permettant de récupérer un utilisateur par son adresse mail. Les valeurs renvoyées par la fonction se limiteront à l'id, l'adresse mail et le mot de passe. Cette fonction est à utiliser très judicieusement puisqu'elle manipule le mot de passe.

Information :
La fonction « getOneCredentials() » est ici importante, car elle nous permet de vérifier l'existence d'une adresse mail dans la base de données.

Une fois que vous avez réalisé les différentes fonctions du modèle, adaptez le code de votre fichier « RegisterController.php » comme cela :

```
function registerController($twig, $db) {

    include_once '../src/model/UserModel.php';

    $form = [];

    if (isset($_POST['btnPostRegister'])) {

        $email = $_POST['userEmail'];
        $password = $_POST['userPassword'];
        $passwordConfirm = $_POST['userPasswordConfirm'];
        $lastname = $_POST['userLastname'];
        $firstname = $_POST['userFirstname'];

        if (count(getOneUserCredentials($db, $email)) === 0) {
            if ($password === $passwordConfirm) {
                saveUser($db, $email, password_hash($password, PASSWORD_DEFAULT),
$lastname, $firstname, 1);

                $form = [
                    'state' => 'success',
                    'message' => "Vous êtes maintenant inscrit au site"
                ];
            } else {
                $form = [
                    'state' => 'danger',
                    'message' => "Les deux mots de passe ne correspondent pas !"
                ];
            }
```

```
        } else {
            $form = [
                'state' => 'danger',
                'message' => "Un compte avec cette adresse mail existe déjà !"
            ];
        }

    }

    echo $twig->render('register.html.twig', [
        'form' => $form
    ]);
}
```

Explications :

- « Count() » est une fonction permettant de compter le nombre d'éléments contenus dans un tableau. Dans notre cas nous comptons les éléments du tableau retourné par la fonction « getOneUserCredentials() » du modèle « UserModel ». Si le tableau ne contient pas d'éléments et donc que « count(getOneUserCredentials($db, $email)) » est égale à zéro, l'utilisateur est autorisé à s'enregistrer avec cette adresse mail. Dans le cas où la valeur serait différente de zéro, cela signifierait qu'un compte avec cette adresse mail existe déjà.
- La condition contenant « $password === $passwordConfirm » nous permet de vérifier que le mot de passe renseigné par l'utilisateur est bien valide.
- Lorsque les deux conditions sont vérifiées, l'inscription de l'utilisateur est autorisée à se faire. Au travers de la fonction « saveUser() » que vous avez créée dans le modèle, « UserModel » permet d'insérer les informations dans la base de données.
- Vous remarquerez que nous avons utilisé la fonction « password_hash() ». Cette fonction permet d'hacher le mot de passe pour le rendre illisible. Le premier paramètre de la fonction correspond au mot de passe entré par l'utilisateur. Le second lui correspond à l'algorithme utilisé pour hacher le mot de passe. Nous utilisons ici l'algorithme de « hachage » par défaut grâce au paramètre « PASSWORD_DEFAULT ».
- Par défaut le rôle est défini sur « 1 », seul un administrateur pourra passer un utilisateur vers le rôle « administrateur »

Vous pouvez maintenant tester votre formulaire d'inscription et vérifier que tout fonctionne correctement.
En jetant un coup d'œil dans la base de données, vous pourrez apercevoir que votre mot de passe a bien été haché.

Le résultat devrait ressembler à cela :

password
$2y$10$/5B0oSO3di7yYBNRpeijAuiWoZo7iXqnVXdtV4fW0Vg...

Travail à faire :

- Ajouter les vérifications qui permettent de s'assurer que les champs du formulaire ne sont pas vides.
- Vérifier que le mot de passe possède au minimum 3 caractères grâce à la fonction « strlen() » (documentation : https://www.php.net/manual/fr/function.strlen.php).

5. Gestion de la connexion

Nous avons tous les éléments nécessaires pour permettre à des utilisateurs de se connecter à notre site e-commerce.

Notre nouvel objectif consiste à rendre fonctionnel notre système de connexion. Pour cela nous allons devoir modifier le contrôleur « LoginController.php » de la manière suivante :

```
function loginController($twig, $db) {

    include_once '../src/model/UserModel.php';

    $form = [];

    if (isset($_POST['btnPostLogin'])) {
        $email = $_POST['userEmail'];
        $password = $_POST['userPassword'];

        $user = getOneUserCredentials($db, $email)[0];

        if ($user) {
            if (password_verify($password, $user['password'])) {
                $_SESSION['auth']['login'] = $user['email'];
                $_SESSION['auth']['role'] = $user['idRole'];

                $form = [
                    'state' => 'success',
                    'message' => "Connexion réussie !"
                ];
            } else {
                $form = [
                    'state' => 'danger',
                    'message' => "Vos informations de connexion sont incorrects !"
                ];
            }
        } else {
            $form = [
                'state' => 'danger',
                'message' => "L'un de vos identifiants est incorrect !"
```

```
            ];
        }
    }

    echo $twig->render('login.html.twig', [
        'form' => $form
    ]);
}
```

Explications :

- Globalement, le code ressemble fortement à celui que nous avons écrit dans le formulaire d'inscription.
- $user = getOneUserCredentials($db, $email)[0]; permet de récupérer les informations de notre utilisateur. Puisque la fonction « getOneUserCredentials() » renvoie un tableau, nous récupérons l'index 0 du résultat obtenu, notamment grâce à «[0] ».
- La première condition « if ($user) » vérifie qu'il existe bien un résultat dans la base de données.
- Dans le cas où nous avons bien un utilisateur, nous vérifions la correspondance du mot de passe contenu dans la base de données avec celui renseigné par l'utilisateur. Pour effectuer cette vérification, nous utilisons la fonction « password_verify() ». En effet, il nous serait impossible de vérifier le mot de passe qui a été haché avec celui entré par l'utilisateur. Même si nous hachons le mot de passe entré par l'utilisateur, le résultat serait différent à celui que nous avons stocké. PHP met donc à disposition la fonction « password_verify() ». Elle prend en premier paramètre le mot de passe du formulaire non haché et en deuxième paramètre le mot de passe haché contenu dans la base de données.
- Lorsque toutes les conditions sont vérifiées, nous pouvons utiliser une variable de sessions pour y stocker les informations nécessaires à l'identification de notre utilisateur qui se connecte.
    - $_SESSION['auth']['login'] = $user['email']; // nous stockons ici l'adresse mail de l'utilisateur, celle-ci nous permettra de récupérer toutes les données de l'utilisateur lorsque nous en aurons besoin.
    - $_SESSION['auth']['role'] = $user['idRole']; // ici nous gardons en mémoire l'id du rôle de l'utilisateur, cette valeur nous servira dans le prochain chapitre notamment pour sécuriser l'accès à certaine page.

Remarques :

Nous avons ici choisi de stocker les informations de l'utilisateur dans un tableau à deux niveaux. Ainsi pour accéder aux différentes informations, nous devrons naviguer dans le premier index « auth ». En second niveau nous pourrons récupérer le mail grâce à l'index « login » et l'id du rôle grâce à l'index « role ».
Autre remarque, il est important de ne pas stocker d'information sensible dans une session. Il est préférable d'exécuter une nouvelle requête pour récupérer l'ensemble des informations qui concernent l'utilisateur.

Vous pouvez maintenant vérifier que votre formulaire de connexion fonctionne correctement. Essayer avec des informations valides, un mot de passe invalide et une adresse mail non valide.

Travail à faire :

- Faire en sorte que le formulaire ne puisse plus être soumis sans remplir les champs.

6. Conditionner l'affichage d'un utilisateur authentifié

Notre système d'authentification est maintenant fonctionnel, néanmoins il serait intéressant de nous permettre d'afficher dans notre vue que l'utilisateur est connecté. Nous allons donc ajuster notre vue pour afficher un menu déroulant qui donnera la possibilité à un utilisateur connecté, d'accéder à des pages spécifiques.

La première chose que nous devons faire, c'est de permettre à Twig d'accéder à la variable de sessions « $_SESSION ». Dans le fichier « twig.php » nous allons adapter le code de la manière suivante :

```
...
$loader = new \Twig\Loader\FilesystemLoader($path);
$twig = new \Twig\Environment($loader, []);
$twig->addGlobal('session', $_SESSION);

return $twig;
...
```

Explication :
« addGlobal() » est une méthode proposée par Twig. Elle permet de lier une variable Twig avec une variable PHP. Sur ce principe, nous indiquons à Twig que la variable « session » doit faire référence à la variable « $_SESSION ».

Puisque nous avons maintenant l'accès à la variable « session » nous allons pouvoir adapter notre vue pour inviter l'utilisateur à se connecter s'il n'est pas authentifié. Dans le cas où il est authentifié, un menu déroulant sera affiché.

Pour ce faire, ouvrez la vue « layout.html.twig » qui doit contenir notre barre de navigation. Puis modifiez le code pour ajouter les lignes suivantes :

```
...
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <div class="container">
    <a class="navbar-brand" href="/">Shop</a>

    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false"
aria-label="Toggle navigation">
```

```
        <span class="navbar-toggler-icon"></span>
      </button>

      <div class="collapse navbar-collapse" id="navbarResponsive">
        <ul class="navbar-nav mb-2 mb-lg-0 me-auto">
          <li class="nav-item">
            <a class="nav-link" aria-current="page" href="index.php">Accueil</a>
          </li>
          ...
        </ul>

        {% if session.auth is defined %}
        <ul class="navbar-nav">
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-
toggle="dropdown" aria-expanded="false">
              {{ session.auth.login }}
            </a>
            <ul class="dropdown-menu">
              <li><a class="dropdown-item" href="">Mon panier (0)</a></li>
              <li><a class="dropdown-item text-danger" href="">Deconnexion</a></li>
            </ul>
          </li>
        </ul>
        {% else %}
          <a href="?page=register" class="btn ms-2 text-light">S'inscrire</a>
            <a href="?page=login" class="btn btn-success ms-2">Se connecter</a>
        {% endif %}

      </div>
    </div>
</nav>
...
```

Explications :

- {% if session.auth is defined %} // Condition qui vérifie que la clé « auth » est bien définie. Si elle est définie, cela signifie que l'utilisateur est connecté.
- {{ session.auth.login }} // affiche la valeur de la clé « login »
- Les 4 dernières lignes surlignées s'affichent seulement si l'utilisateur n'est pas connecté. Plus précisément, un utilisateur non authentifié obtiendra deux boutons le redirigeant soit vers l'action de connexion, soit vers l'action d'inscription.

Travail à faire :

- Afficher un nouveau menu « Administration » lorsqu'un utilisateur se connecte avec un compte « Administrateur ».

- Ce menu doit contenir :
  - Le lien vers la page de gestion des produits
  - Le lien vers la page de gestion des catégories de produit

7. Ajout de la déconnexion

Actuellement, il est impossible de se déconnecter de notre site. Cela peut être gênant si nous souhaitons tester la connexion avec un autre utilisateur. Ou même par la suite, un utilisateur aura certainement besoin de se déconnecter de notre site.

Travail à faire :

- Faire le nécessaire pour que l'accès à l'URL « index.php?page=logout » soit fonctionnelle.
- Nommer le contrôleur « LogoutController »

Une fois que vous avez configuré la nouvelle route, vous allez pouvoir compléter le code du contrôleur notamment grâce aux quelques lignes suivantes :

```
function logoutController($twig, $db) {
    $_SESSION = [];
    session_destroy();
}
```

Explication :

- $_SESSION = []; // la variable de session est réinitialisée avec un tableau vide. Cela assure la suppression des données que nous stockons à l'intérieur.
- session_destroy(); // la fonction détruit la session en cours

Vous pouvez tester d'accéder à la page de déconnexion.

Remarque :
Lorsque vous vous déconnectez, une page blanche apparait. Cela est tout à fait normal. Vous pouvez essayer de changer de page en modifiant l'URL.

8. Améliorations diverses

Redirection de page :

Comme vous avez dû le remarquer, lorsque nous nous déconnectons, nous restons bloqués sur une page blanche. C'est d'ailleurs le même phénomène qui se produit lorsque nous nous connectons au site.
Cela n'est vraiment pas pratique ni intuitif.

Nous allons donc y remédier notamment grâce à l'utilisation d'une fonction permettant de rediriger un utilisateur vers une autre page.

Attention !
La fonction que nous allons voir est à utiliser avec modération. Car celle-ci pourrait vous empêcher de détecter des erreurs dans votre code. Il est donc conseillé de s'assurer du bon fonctionnement du code avant d'utiliser cette fonction.

Nous allons ici utiliser la fonction « header() » qui permet de réécrire l'entête d'une requête HTTP et plus particulièrement nous utiliserons le type d'appel « Location » pour rediriger l'utilisateur sur une page spécifique.

Dans le contrôleur « LogoutController » vous pouvez adapter le code comme ceci :

```
function logoutController($twig, $db) {
    $_SESSION = [];
    session_destroy();
    header("Location: index.php");
}
```

Explication :

- Ici nous redirigerons l'utilisateur vers la page d'accueil du site.

Nous pouvons faire la même chose lorsqu'un utilisateur se connecte à notre site. Ajoutons la ligne suivante dans le fichier « LoginController.php » :

```
...
if ($user) {
    if (password_verify($password, $user[0]['password'])) {
        $_SESSION['auth']['login'] = $user[0]['email'];
        $_SESSION['auth']['role'] = $user[0]['idRole'];

        $form = [
            'state' => 'success',
            'message' => "Connexion réussie !"
        ];

        header("Location: index.php");
    } else {
        $form = [
            'state' => 'danger',
            'message' => "Vos informations de connexion sont incorrectes !"
        ];
    }
} else {
    $form = [
        'state' => 'danger',
```

```
        'message' => "L'un de vos identifiants est incorrect !"
    ];
}
...
```

Travail à faire :

- Réaliser les mêmes modifications lorsqu'un produit ou une catégorie de produit est supprimé.
- Rediriger l'utilisateur sur la page de gestion des produits ou des catégories de produits au lieu d'afficher une page blanche.

Envoi de mail :

Enfin, la dernière amélioration que nous pouvons apporter à notre système d'authentification c'est l'envoi d'un mail. Ce mail permettra de confirmer l'inscription de l'utilisateur sur le site e-commerce.

Nous allons ici utiliser la librairie PHPMailer qui simplifie la configuration et l'envoi de mail. Pour que nous puissions l'utiliser, il faut impérativement installer le paquet « Postfix » pour permettre à notre serveur d'envoyer ces mails.

À partir de la console de VSCode, passez en utilisateur « root ».
Si vous êtes sous l'utilisateur loginXXXX vous pouvez faire cette simple commande :

```
exit
```

Sinon vous pouvez utiliser la commande :

```
su root
```

Maintenant, installons le paquet « postfix » :

```
apt install postfix
```

Lors de l'installation, une question vous sera posée. Répondez simplement par « site internet ».

Une fois que l'installation est terminée, pensez à réutiliser l'utilisateur loginXXXX.

La configuration par défaut de « Postfix » conviendra parfaitement à nos besoins. Néanmoins, nous devons installer cette librairie avec Composer.

Travail à faire :

- Installer la librairie « phpmailer/phpmailer »

Une fois cela fait, vous pouvez créer le fichier et le dossier nécessaire suivant :
« template/mail/register_message.html.twig »

Dans ce fichier, insérez le code HTML suivant :

```
<p>Bonjour {{ email }},</p>
<p>Merci d'avoir rejoint notre site.</p>

<p>
   Cordialement,<br />
   L'équipe Shop
</p>
```

Ajoutons également le code suivant dans le contrôleur « RegisterController », juste au moment où l'utilisateur est enregistré en base de données :

```
...
// Initialisation de la librairie
$mail = new PHPMailer(true);
$mail->CharSet = "UTF-8";

// Ajout des adresses (origine et destinataire)
$mail->setFrom('noreply@shop.fr', 'Shop');
$mail->addAddress($email, $firstname . ' ' . $lastname);

// contenu
$mail->isHTML(true);
$mail->Subject = 'Inscription à Shop';
$mail->Body    = $twig->render("mail/register_message.html.twig", [
   'email' => $email
]);

$mail->send();
...
```

Et enfin dans le même fichier, nous devons adapter le contrôleur pour obtenir le code suivant :

```
<?php

use PHPMailer\PHPMailer\PHPMailer;

   function registerController($twig, $db) {
...
```

Explication :

- La ligne surlignée permet d'indiquer à PHP d'importer une librairie au travers d'un alias dans le code. Cela est couramment pratiqué en programmation-objet. Puisque la librairie PHPMailer est un objet, nous devons ici recourir à cette pratique.

Attention :
Il se peut que votre mail ne soit pas délivré correctement. Cela est en fait dû à la configuration du serveur et du nom de domaine. En conséquence les différentes boites mail (Gmail, Outlook, …) considèrent les mails comme indésirables et ne sont donc pas délivrées.

Pour y remédier, vous pouvez utiliser le serveur mail de Google en configurant la librairie PHPMailer par l'ajout des éléments suivants :

```
//Server settings
$mail->SMTPDebug = SMTP::DEBUG_SERVER;
$mail->isSMTP();
$mail->Host       = 'smtp.example.com'; // Lien du serveur SMTP
$mail->SMTPAuth   = true;
$mail->Username   = 'user@example.com'; // Adresse mail
$mail->Password   = 'secret'; // Mot de passe
$mail->SMTPSecure = PHPMailer::ENCRYPTION_SMTPS;
$mail->Port       = 465;
```

Pour plus d'informations concernant la librairie PHPMailer, vous pouvez consulter le lien suivant : https://github.com/PHPMailer/PHPMailer

Travail à faire :

- Envoyer votre code sur Github