

Compte-Rendu de TP4 :

POST/Session avec Smarty.

Ferdinand Justin-Groupe 4

Objectif :

Créer un ensemble de pages pour créer un compte, s'identifier et accéder à une ressource une fois connecté.

Sommaire



1. Analyse du sujet



2. Formulaires et traitements de données



3. Stockage des mots de passe



4. Connexion et session



5. Schéma fonctionnel



6. Conclusion

1. Analyse du sujet

Le sujet a pour objectif de créer un site Web où l'utilisateur peut créer et / ou se connecter à un compte. Il s'agit ici de développer la partie serveur du site. Dans un premier temps, afin de traiter les informations d'une personne, il faut créer une base de données. Celle-ci contiendra des champs tel que le nom, la ville, l'adresse e-mail ou encore le mot de passe. Ces données seront insérée dans la base de données pour être réutilisées.

Enfin, Il faut mettre en place des formulaires pour que l'utilisateur puisse saisir ces données, un formulaire d'inscription et un formulaire de connexion seront nécessaires.

Pour finir, il faut réaliser un système de session pour que la personne n'ait pas à ressaisir ces identifiants à chaque page.

Pourquoi utiliser Smarty ?

Voici la définition donnée sur la page d'accueil de Smarty :

Smarty est un moteur de template pour PHP. Plus précisément, il facilite la séparation entre la logique applicative et la présentation. Cela s'explique plus facilement dans une situation où le programmeur et le designer de templates jouent des rôles différents, ou, comme la plupart du temps, sont deux personnes distinctes.

Ainsi Smarty est un outil logique qui permet la mise en place de divers services sous formes de templates. Tels que le traitement de base de données, d'informations, formulaires, etc.

De plus, il permet de séparer le travail, cela s'est vu lors de la programmation de ce TP4. En effet, les templates (contenant les formulaires des pages de login, register, etc) ne sont pas pourvus d'algorithmes. Templates et routes peuvent se faire séparément.

Le code PHP servant à réaliser les objectifs de ce TP est placé dans des fichiers .php tel que routes.php, index.php ou encore pdo.php. Les templates utilisés sont placés dans le dossier templates.

2. Formulaires et traitement des données

L'objectif des exercices 1 et 2 est de créer des services de saisies pour la création d'un compte utilisateur. Pour cela nous allons faire un formulaire nommé register. Ce sera un template Smarty qui permet d'envoyer les données aux routes flight GET et POST register.

Contenu du template register :

Un formulaire est composé de plusieurs labels qui permettent à l'utilisateur de saisir une donnée. Ces labels sont créés de la manière ci-dessous.

```
div class="pure-control-group">
    <label for="aligned-name"><b>Votre nom :</b></label>
    <input name="name" type="name"
value="{ $valeurs.name|escape|default: '' }" placeholder="Jean" />
    { $msg['name']|default: '' }
</div>
```

Voilà l'exemple pour le label Nom qui permet à l'utilisateur de saisir son nom.

Une balise Input permet de contenir la valeur et la renvoyer. Pour enregistrer les différentes données, le tableau associatif nommé *valeurs* possède plusieurs attributs nommés name pour le nom, city pour la ville, country pour le pays, email pour l'adresse email et psw pour le mot de passe.

L'attribut escape permet de renvoyer la valeur dans un tableau utilisé dans les routes.

L'attribut default permet de mettre une valeur par défaut sur le label. Ici nous ne mettons rien.

Le placeholder permet de mettre un exemple dans l'input. L'utilisateur comprendra qu'il faut mettre son nom.

De plus, un tableau \$msg de la valeur name contient les erreurs de saisies s'il y en a. Sa valeur par défaut est nulle mais peut changer si une erreur sur le nom apparaît.

Enfin, un bouton situé en bas du formulaire permet de valider la saisie.

```
<button type="submit" class="btn btn-primary">Se connecter</button>
```

Les données seront envoyées vers les routes. Pour cela nous devons créer deux d'entre-elles, GET et POST register.

Route de type GET dans routes.php :

```
Flight::route('GET /register', function(){  
    Flight::render("register.tpl",array());  
});
```

Cette route de chemin /register permet de renvoyer l'utilisateur sur le formulaire d'inscription. Pour voyager à travers le site et ses pages, nous allons déclarer des routes GET /page pour chacune des pages. Même cas pour POST si on récupère des données dans une page.

Route de type POST dans routes.php :

```
Flight::route('POST /register', function(){  
});
```

Cette route récupère les données du formulaire et les traite. Pour cela, elle utilise la fonction request :

```
$data = Flight::request()->data;  
//$data->name contient le nom
```

Le tableau data récupère les données du tableau associatif du template, il est composé de ces attributs :

- name : Nom de l'utilisateur
- city : Ville de l'utilisateur
- country : Pays de l'utilisateur
- email : Adresse e-mail de l'utilisateur
- psw : Mot de passe de l'utilisateur qui sera crypté (Cela se verra par la suite)
- rpsw : Second mot de passe qui atteste que l'utilisateur a bien saisi ce-dernier.

Maintenant que nous sommes en possession de ces données, il faut, avant de les insérer dans la base de données, vérifier qu'elles soient correctes. Exemples de cas de vérification :

- Le champ n'a pas été saisi, l'attribut du tableau est vide
- L'adresse e-mail n'est pas dans un format valide
- L'adresse e-mail est déjà reliée à un compte
- Le mot de passe fait moins de 8 caractères (sécurité)
- Les mots de passe ne correspondent pas.

La route POST /register va tester tous ces paramètres. De plus, elle va pouvoir retourner les erreurs dans le template register afin d'informer l'utilisateur de ce qui n'est pas correct dans sa saisie.

Les erreurs vont être passées dans un tableau nommé message.

```
$msg = array();
```

Tests :

Afin de vérifier que les champs des labels ne soient pas vides, nous allons tester les valeurs qu'ils renvoient à l'aide de fonctions PHP telle que `empty()` qui vérifie si le paramètre est vide et renvoie vraie. Nous aurons aussi besoin de la fonction PHP `trim()` qui retire les espaces d'une chaîne de caractères passée en paramètre.

Ces fonctions sont donc utilisées pour tous les attributs du formulaire, exemple ci-dessous avec le nom.

```
if (empty(trim($data->name))){
    $msg['name'] = "nom obligatoire";
    $erreur = true;
}
else
    //Reste des tests de l'adresse e-mail
```

La condition vérifie si la fonction `empty()` renvoie vrai. Cette dernière passe en paramètre la fonction `trim()` qui elle-même passe en paramètre le nom : `$data-> name`. Dans le cas où la condition est vraie, le tableau `msg` prend en valeur une chaîne de caractère expliquant l'erreur dans la case du nom. Ensuite, un booléen nommé `erreur` passe en `true`. Il permettra plus tard de montrer qu'il existe une erreur.

Test de l'adresse e-mail :

La condition suivante se situe dans le `else` du test précédent. En effet, si le champ est vide, il est inutile de tester si l'adresse est dans un format correct.

```
if (!filter_var($data->email, FILTER_VALIDATE_EMAIL))
{
    $msg['email'] = 'Email non correcte';
    $erreur = true;
}
```

La fonction PHP `filter_var` passant en paramètre l'email saisie et un filter `FILTER_VALIDATE_EMAIL` renvoie vrai si l'adresse est correcte.

Cependant, ici nous allons tester si l'adresse n'est pas correcte. Pour cela nous rajoutons l'opérateur « ! » qui permet de faire NON(condition) -> La condition `if` teste alors si l'adresse n'est pas correcte.

Si c'est le cas, alors le tableau `msg` prendra en valeur le texte de l'erreur dans la case email et le booléen `erreur` passera en `true`.

Si l'adresse est dans un format correct, alors nous allons tester si celle-ci n'existe pas déjà dans la base de données.

Pour récupérer cette dernière, nous allons utiliser le fichier pdo.php qui déclare la variable db récupérant les bases de données de phpmyadmin.

```
<?php $db= new
PDO('mysql:host=localhost;port=3306;dbname=tp2;charset=utf8','root','justin');
$db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
?>
```

Dans notre route POST /register, nous allons récupérer cette variable db :

```
$db = Flight::get('pdo');
```

Pour manipuler la base de données, nous allons créer une requête sql nommée testEmail. La fonction PHP prepare() va préparer la requête qui sera exécutée par la fonction PHP execute(). Enfin, une variable nommée verifUser va récupérer le résultat de la requête avec la fonction PHP fetch(). Si cette variable contient une chaîne de caractère (et donc une adresse e-mail), alors l'e-mail est déjà utilisée sur un autre compte. L'erreur est enregistrée dans le tableau msg.

```
//Test pour vérifier que le mail n'est pas dans la base
    $testEmail=$db->prepare('SELECT Email FROM utilisateurs WHERE Email= ?');
    $testEmail->execute([$data->email]);
    $verifUser = $testEmail->fetch();
    if ($verifUser){
        $msg['email'] = 'Email déjà utilisé';
        $erreur = true;
    }}
```

Si aucune de ces erreurs n'est trouvée, alors on considère que l'adresse e-mail est utilisable.

Test du mot de passe

```
if (strlen($data->psw) < 8) {  
    $erreur = true;  
    $msg['psw'] = "Mot de passe trop court";  
}  
if (empty(trim($data->psw))){  
    $msg['psw'] = "Mot de passe obligatoire";  
    $erreur = true;  
}
```

La fonction PHP strlen() permet de renvoyer le nombre de caractères présents dans une chaîne passée en paramètre. Ici, nous allons tester si le mot de passe contenu dans \$data->psw contient au minimum 8 caractères.

Ensuite nous allons tester si le champ n'est pas vide. Cet ordre de conditions peut paraître étrange mais d'un point de vue algorithmique ; Tester s'il est vide avant de regarder s'il est inférieur à 8 caractères renverra l'erreur suivante : « Mot de passe trop court » alors qu'aucun n'a été saisi. Pour cause, le tableau prendrait cette erreur en dernier.

Par la suite, nous allons tester si les deux mots de passe sont identiques, pour cela, nous utilisons l'opérateur de différence « != » entre les deux attributs « psw » et « rpsw » du tableau data.

```
if ($data->psw != $data->rpsw){  
    $erreur = true;  
    $msg['rpsw'] = "Saisissez le même mot de passe";  
}
```

En cas d'erreur, le tableau msg du mot de passe de confirmation indiquera l'erreur.

Insertion des données.

Maintenant que tous les tests ont été effectués, nous allons pouvoir insérer dans la base de données les informations, à condition qu'aucune erreur ne soit détectée.

C'est là qu'intervient le booléen erreur. Il agit comme un drapeau, en cas d'erreur il se lève en passant à true.

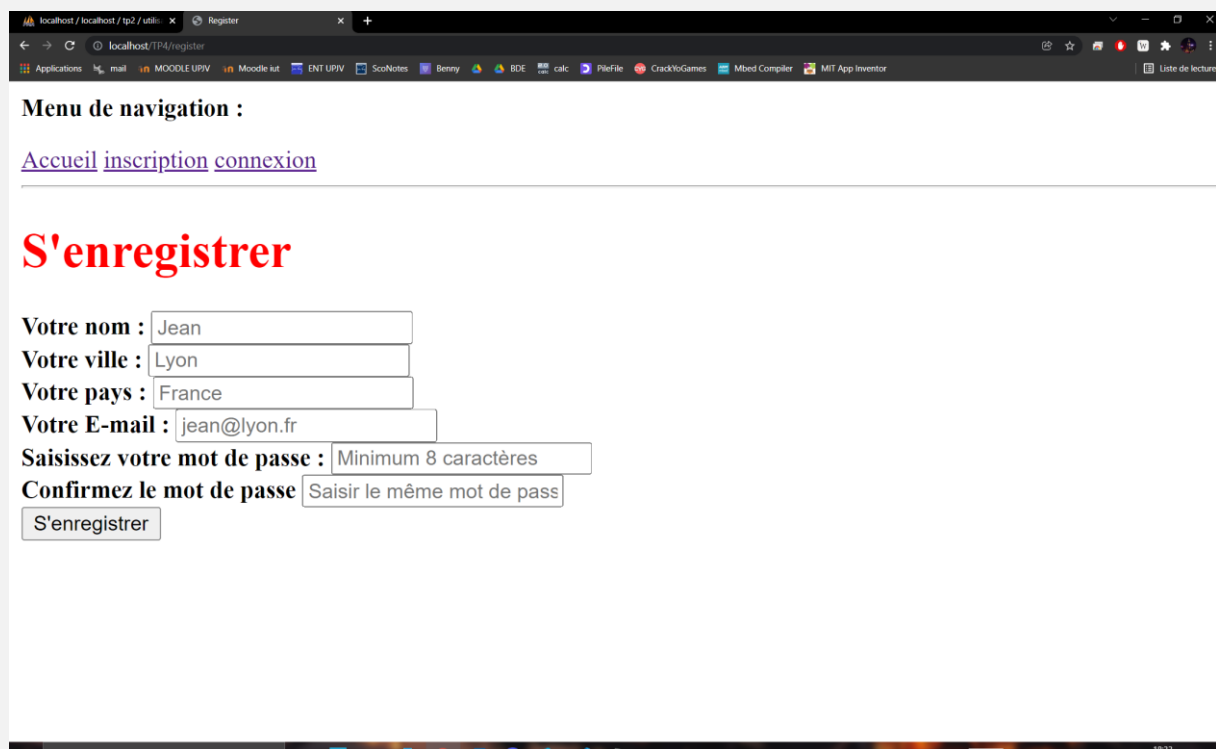
```
if ($erreur == true) {  
    Flight::render("register.tpl",array("msg" => $msg, 'valeurs' => $_POST));  
} else {  
    $requete = $db -> prepare("INSERT INTO `utilisateurs` (`Nom`, `Pays`,  
`Ville`, `Email`, `Motdepasse`) VALUES (:Nom,:Pays,:Ville,:Email,:Motdepasse)");  
    $requete->execute(array(  
        ':Nom' => $_POST['name'],  
        ':Pays' => $_POST['country'],  
        ':Ville' => $_POST['city'],  
        ':Email' => $_POST['email'],  
        ':Motdepasse' => password_hash($_POST['psw'], PASSWORD_DEFAULT)  
    ));  
    Flight::render("success.tpl",array());  
}
```

Dans un premier temps, nous allons tester si erreur est levé, si c'est le cas alors une erreur de saisie existe. La fonction render de flight permet de renvoyer l'utilisateur sur le template register, accompagné du tableau d'erreur msg et du tableau de valeur _POST. Ainsi l'utilisateur pourra conserver sa saisie et ne modifiera que le champ où une erreur est présente.

S'il n'y a pas d'erreur, une requête sql est créée comme vu précédemment. Celle-ci va insérer les données dans la base avec INSERT INTO table VALUES valeurs.

La requête est exécutée par execute() qui prend en paramètre les attributs de la base qui prennent les valeurs des champs. Notons que seul le premier mot de passe est enregistré, il est de plus crypté avec la fonction PHP password_hash().

Enfin, l'utilisateur est renvoyé vers le template success avec la fonction render de flight.

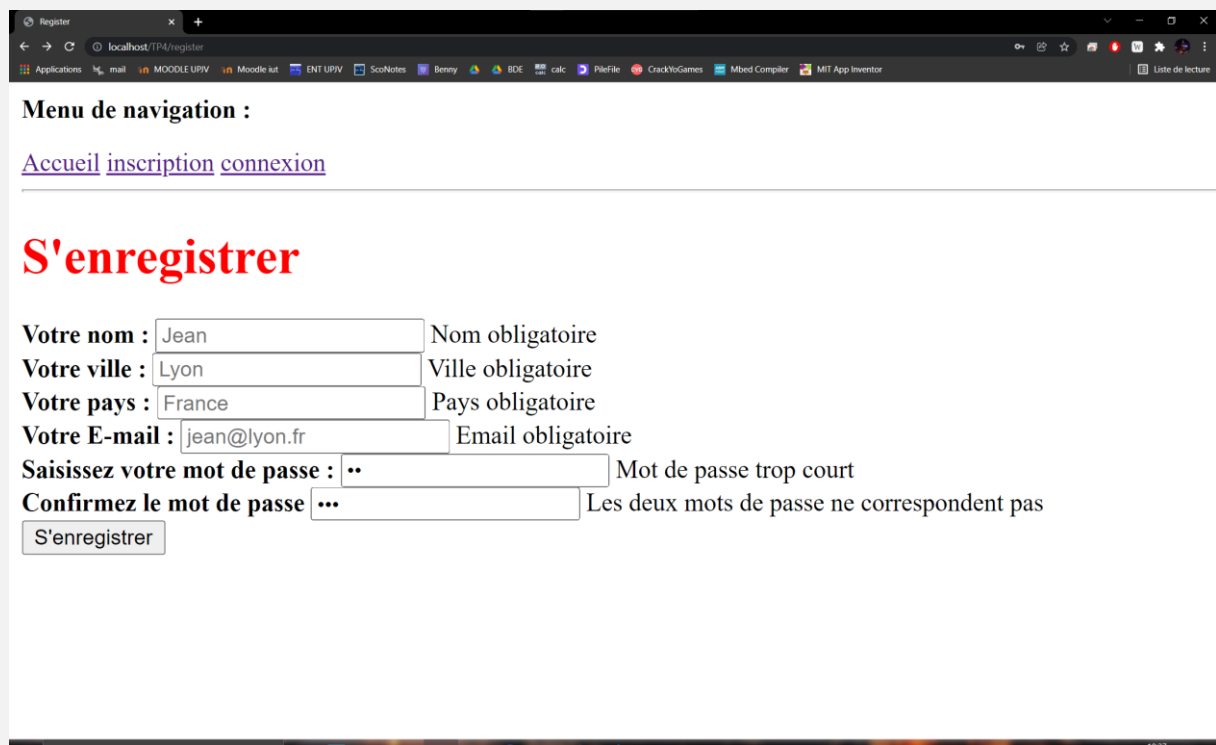
Rendu du formulaire sur le site :

The screenshot shows a web browser window with the URL `localhost/tp4/register`. The page has a navigation menu with links: [Accueil](#), [inscription](#), and [connexion](#). Below the menu is a large red heading **S'enregistrer**. The registration form contains the following fields and labels:

- Votre nom :
- Votre ville :
- Votre pays :
- Votre E-mail :
- Saisissez votre mot de passe :
- Confirmez le mot de passe :
-

Le formulaire présent sur la route register permet de saisir les données suivantes : nom, ville, pays, e-mail, mot de passe et second mot de passe.

En cas d'erreur de saisie, l'utilisateur peut voir les messages suivants :



The screenshot shows the same registration form as before, but with validation errors displayed next to the password fields. The errors are:

- Saisissez votre mot de passe : Mot de passe trop court
- Confirmez le mot de passe : Les deux mots de passe ne correspondent pas

The other fields (nom, ville, pays, e-mail) and the "S'enregistrer" button remain the same.

Votre E-mail :

! Veuillez inclure "@" dans l'adresse e-mail. Il manque un symbole "@" dans "j".

Si le mail n'est pas dans un format correct, une bulle va indiquer à l'utilisateur ce qu'il manque

Dans le cas où toutes les saisies sont correctes, l'utilisateur sera redirigé vers la page succès.

La requête sql insérera les données dans la base de données, comme le montre la capture d'écran suivante de phpMyAdmin :

phpMyAdmin interface showing the 'utilisateurs' table in the 'tp2' database. The table contains 7 rows of user data. The SQL query 'SELECT * FROM `utilisateurs`' is shown at the top. The interface includes navigation tabs, a sidebar with database structure, and a console at the bottom.

	Nom	Pays	Ville	Email	Motdepasse			
<input type="checkbox"/>	Éditer	Copier	Supprimer	a	a	a.f@g.f	\$2y\$10\$C9PF5guvpJpJLOiSqizte/bi/rD76eclDnxsFDSb/RL...	
<input type="checkbox"/>	Éditer	Copier	Supprimer	Nom	Pays	Ville	email2@email.com	\$2y\$10\$EuIlMudNdXVS8WkiBZfupS6pB1yTjJ.AN.YWmri/H...
<input type="checkbox"/>	Éditer	Copier	Supprimer	Nom	Pays	Ville	Email@email.com	12345678
<input type="checkbox"/>	Éditer	Copier	Supprimer	Jean	a	a	J@f.fr	\$2y\$10\$wvV8KzRbnUqsWxhLJq/OZCsAHCmf6fAT8N3nmjCun...
<input type="checkbox"/>	Éditer	Copier	Supprimer	J	F	L	jean@lyon.fr	\$2y\$10\$wjDRFLY4Sa52TaT9fXsOWBB22xGP6oi4oGNXgUE0o...
<input type="checkbox"/>	Éditer	Copier	Supprimer	Justin	France	Amiens	justin@amiens.fr	\$2y\$10\$0XKaNkiTwFJlspInQRA8Ye8xkvre.RjztYZ7jZzhum...
<input type="checkbox"/>	Éditer	Copier	Supprimer	Vincent	France	Paris	vincent@ipsa.fr	\$2y\$10\$Ny5.83E8/BBMGw6MPcOf4.q7RecV6j/FSePR7c1sP1...

Il s'agit de tous les comptes créés par l'utilisateur.

Le même mot de passe « 12345678 » a été saisi pour tous les comptes, nous pouvons remarquer qu'il est enregistré de manière cryptée par une longue chaîne de caractères différentes pour chaque.

3. Stockage des mots de passe

La protection des données est primordiale dans le développement web côté serveur. Si la base de données venait à être récupérée par une personne malhonnête, les mots de passe seraient accessibles à tous. Dans l'objectif d'empêcher, ou de ralentir les malfaiteurs, nous avons la possibilité de crypter les mots de passe de nos utilisateurs. La technique du hachage permet d'insérer le mot de passe déjà chiffré dans la base de données, c'est une sécurité de plus en cas de récupération pendant le transfert de données.

Un mot de passe chiffré à bas niveau serait facilement déchiffré par l'utilisation de dictionnaires de mots de passe. Dans notre cas, le hachage prévu par php est dit salé. Ainsi, chaque mot de passe sera crypté par ce grain de sel. La fonction php `password_hash()` le propose.

```
password_hash($_POST['psw'], PASSWORD_DEFAULT)
```

```
$2y$10$C9PF5guvpJpJLOiSqiuzte/b/rD76eclDNxsFDSbfRL663RPI6tRC
```

```
$2y$10$EuilMrudNdXVS8Wki/BZfupS6pB1yT/J.AN.YWmri/HO4.RFijfDS
```

Les deux mots de passe ci-dessus équivaux à 12345678.

Nous pouvons remarquer que le grain de sel joue son jeu puisque les deux sont différents en grande partie. Le décryptage sera plus compliqué à réaliser pour le voleur.

Mais comment vérifions-nous si le mot de passe saisi et le mot de passe de la base de données sont identiques si l'un est haché ?

Php est bien pensé puisqu'il propose une fonction de déchiffrement nommée `password_verify()`. Elle prend en paramètre les deux mots de passe à comparer.

```
password_verify($data->psw, $passwd[0]){
```

4.Connexion et session

Maintenant que la création de compte est réalisée, nous pouvons développer la connexion à ce dernier.

L'objectif du 2^{ème} exercice est de présenter un affichage d'une page de connexion avec formulaire pour que l'utilisateur puisse retrouver son compte. Une gestion d'erreurs similaire à l'exercice 1 sera réalisée avant de laisser l'ouverture du profil.

Voici les tests réalisés pour les 2 champs requis :

Email :

- Vérifier que l'email soit saisi
- Vérifier que l'email soit dans un format correct
- Vérifier que l'email soit présent dans la base de données

Mot de passe :

- Vérifier que le mot de passe soit saisi
- On considère que le mot de passe est déjà supérieur à 8 caractères puisque le register le demande. Dans le cas où il s'agit d'un ancien compte, il est possible que le formulaire de ce temps ne demandait pas de taille minimale de mot de passe.
- Vérifier que le mot de passe corresponde à l'adresse e-mail donné.

La réalisation des tests étant similaire à celle de l'exercice 1, je ne développerai pas sur leur fonctionnement dans cette partie.

Dans un premier temps, nous allons recueillir, une fois les tests de l'adresse passé, le mot de passe du compte que l'on va comparer à celui saisi dans le champ.

```
$mdp = $db->prepare("SELECT Motdepasse FROM utilisateur WHERE Email = ?");  
$mdp->execute(array($data->email));  
$passwd = $mdp->fetch();
```

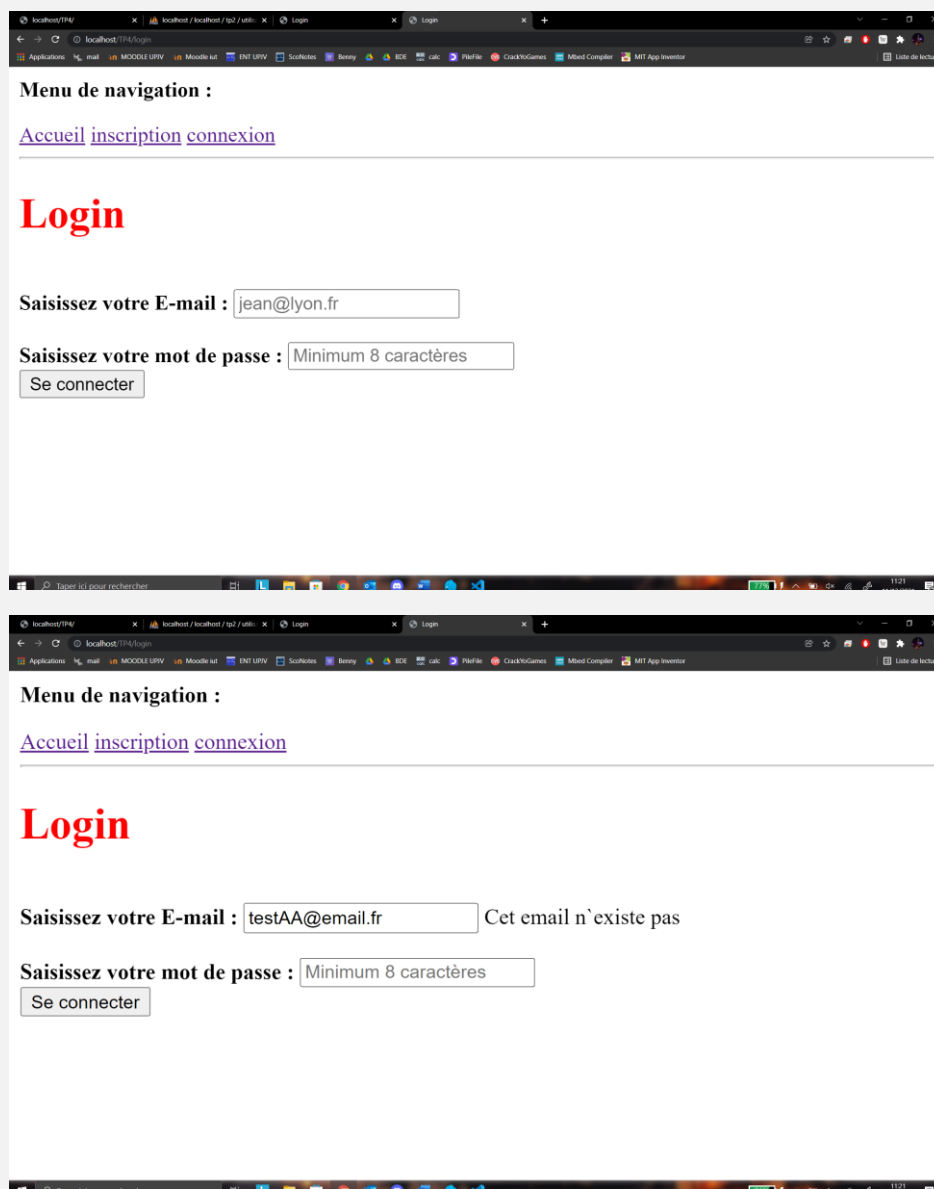
On utilise une requête sql pour récupérer le mot de passe chiffré de la base de donnée à l'aide de la fonction prepare(), execute() et fetch() pour récupérer le mot de passe.

```
if(!password_verify($data->psw, $passwd[0])){  
    //Si le mot de passe saisi par l'utilisateur ne correspond  
    pas à celui dans la bdd, alors il est incorrect  
    $msgLogin['psw'] = "Mot de passe incorrect";  
    $TestErreur = true;  
}
```

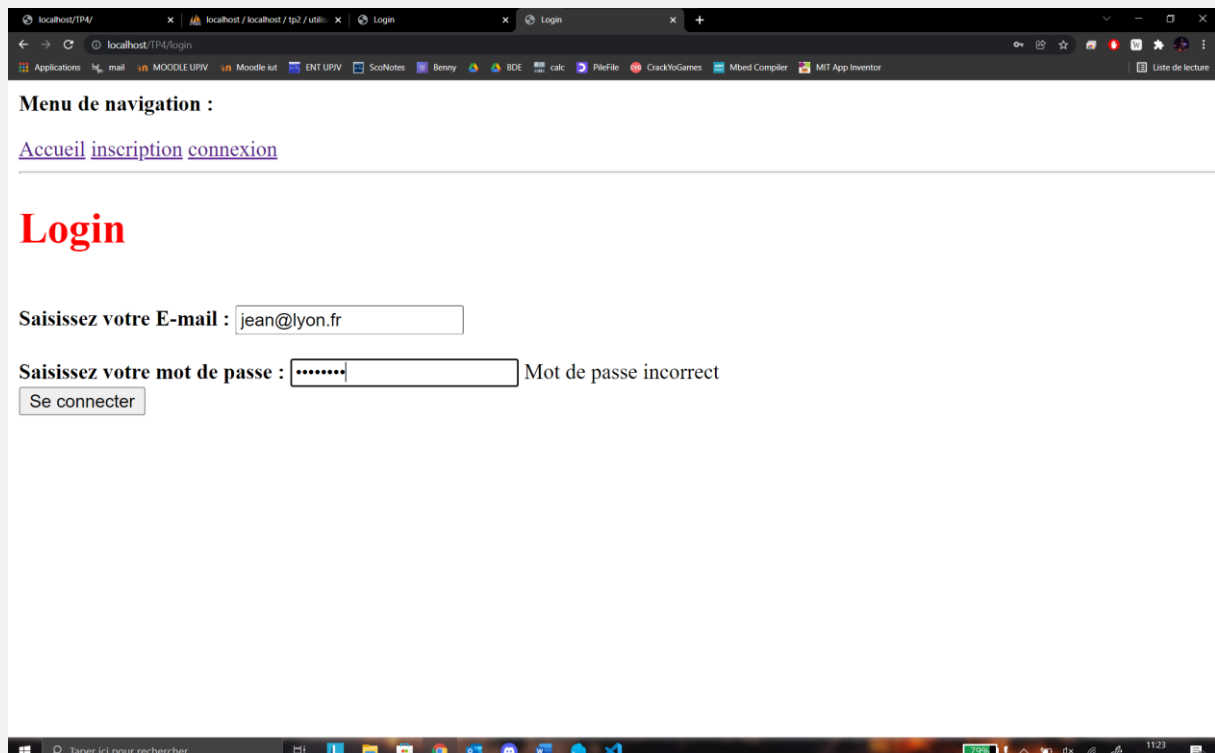
La fonction php password_verify prend en paramètre le mot de passe saisi ainsi que le mot de passe chiffré de la base de données. Elle renvoie vrai si les deux sont identiques.

Dans la capture d'écran ci-dessous, nous testons si l'adresse email existe et si le mot de passe est correct. Une erreur sera alors détectée.

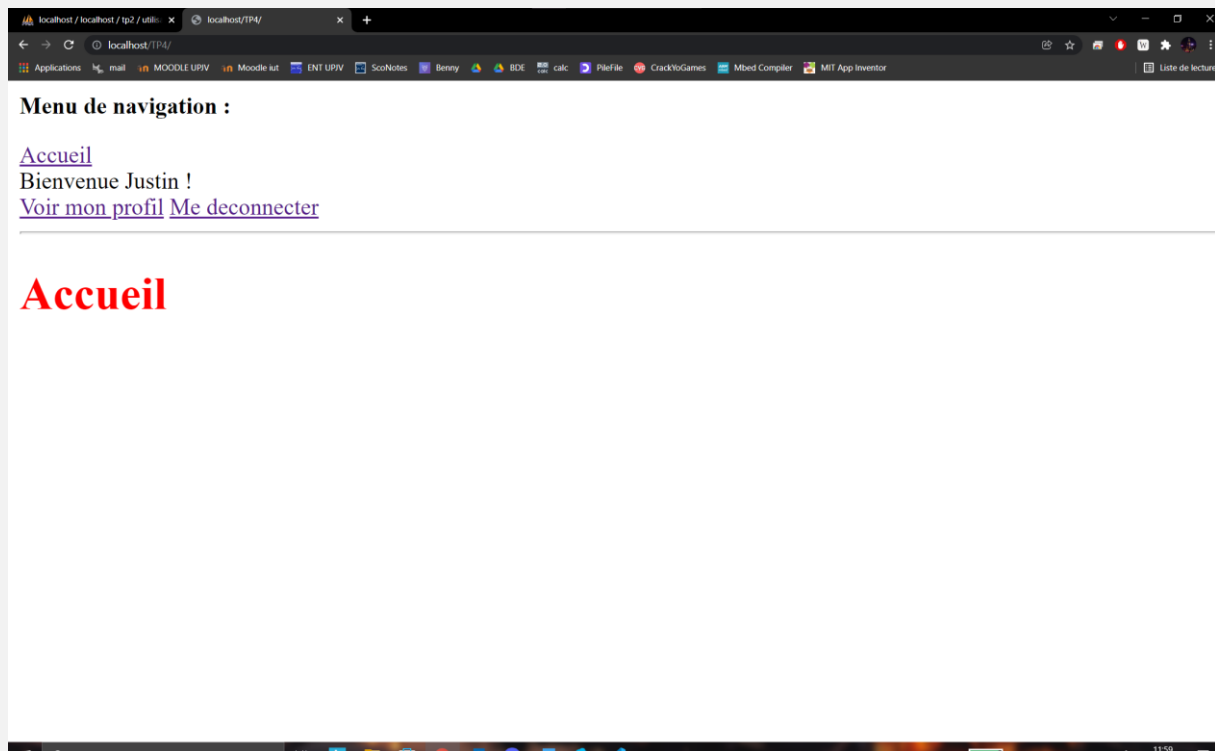
Rendu sur la page web :



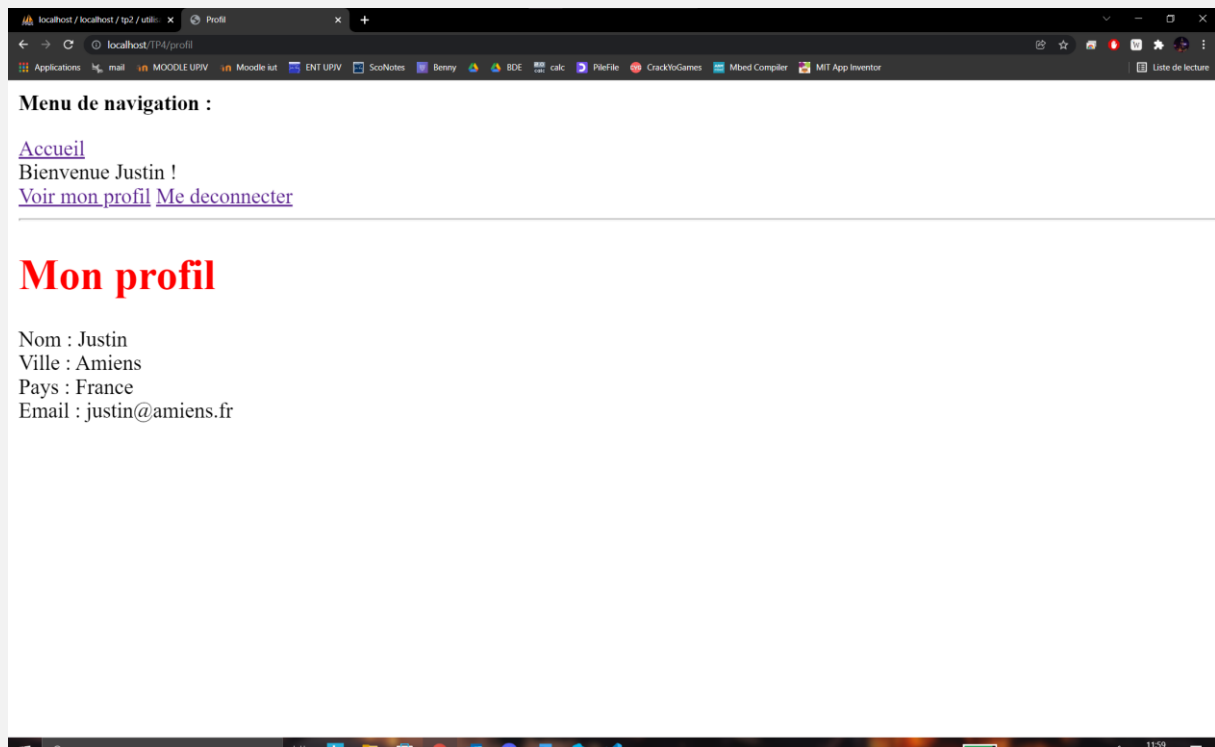
Essai avec le compte de Jean de Lyon :



Une fois l'adresse e-mail et le bon mot de passe saisis, l'utilisateur est renvoyé sur la page d'accueil qui change :



Le lien pour le profil apparaît, ainsi qu'un lien pour se déconnecter. Le nom de l'utilisateur apparaît dans le template layout.



Le template profil affiche les informations de l'utilisateur.

La session reste active entre les pages comme nous pouvons le voir. L'utilisateur a la possibilité de se déconnecter en cliquant sur le lien Me deconnecter.

Une session permet à un utilisateur de rester connecté à son profil tout en naviguant à travers les pages du site. Il peut voir ses informations, ses messages etc.

Pour commencer la session, il faut la démarrer depuis l'index.php :

```
session_start();
```

La session s'active à la première connexion et reste ouverte jusqu'à ce qu'il décide de se déconnecter ou quitter le site. Elle prend des valeurs telles que le nom, la ville, le pays et l'adresse e-mail qui sont stockées dans la route POST /login.

Le tableau \$_SESSION prend les valeurs depuis le résultat de la requête qui récupère toutes les informations de l'utilisateur.

```
$nameUser = $db->prepare("SELECT * FROM utilisateurs WHERE Email = ?");
$nameUser->execute(array($data->email));
$resultat = $nameUser->fetchAll();
//Le tableau _SESSION va récupérer les valeurs du nom et de l'email
foreach($resultat as $indiceTab){
    $_SESSION['name'] = $indiceTab[0];
```



```

        $_SESSION['country'] = $indiceTab[1];
        $_SESSION['city'] = $indiceTab[2];
        $_SESSION['email'] = $indiceTab[3];
    }
    Flight::render("index.tpl", array());

```

Ces données ne sont pas supprimées tant que l'utilisateur ne se déconnecte pas ou ne quitte pas le site.

Pour se déconnecter, on crée la route /logout qui va supprimer le contenu du tableau avec la fonction php session_unset(). La session sera détruite par la fonction php session_destroy(). Une fois déconnecté, l'utilisateur est retourné vers l'accueil du site.

Sans la session, on retrouve l'affichage habituel du site.

```

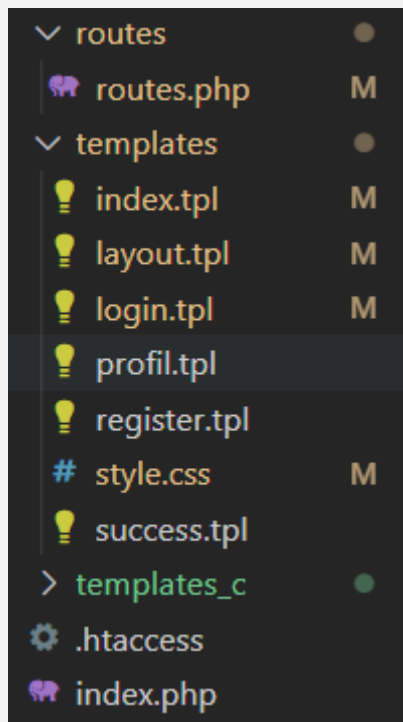
Flight::route('GET /logout', function(){
    session_unset();
    session_destroy();
    Flight::render("index.tpl", array());
});

```

Données sauvegardées dans phpMyAdmin :

Nom	Pays	Ville	Email	Motdepasse
a	a	a	a.f@g.f	\$2y\$10\$C9PF5guyvpJpJLOiSquiuzte/b/rD76eclDNxsFDSbfRL663RPI6tRC
Nom	Pays	Ville	email2@email.com	\$2y\$10\$EuilMrudNdXVS8Wki/BZfupS6pB1yT/J.AN.YWmri/HO4.RFijfDS
Nom	Pays	Ville	Email@email.com	12345678
Jean	a	a	J@f.fr	\$2y\$10\$iwvV8tKzRbnUqsWxhLJq/OZCsAHCmfbfAT8N3nmjCun4oxiFLFSKy
J	F	L	jean@lyon.fr	\$2y\$10\$wjDRFXLY4Sa52Tat9fXSxOWBB22xGP6oi4oGNXgUE0oVj.9BzNb8.
Justin	France	Amiens	justin@amiens.fr	\$2y\$10\$0XKaNkiTwFJlspInQRA8Ye8xkvre.RjztYZ7jbZ2humX6w36bsBKu
Vincent	France	Paris	vincent@ipsa.fr	\$2y\$10\$Ny5.83E8/BBMGw6MPcOf4.q7RecVi6j/FSePR7c1sP1RTubXojVVO

5.Schéma fonctionnel



Cette capture d'écran réalisée dans Visual Studio Code présente l'architecture du site.

Nous remarquons 3 catégories :

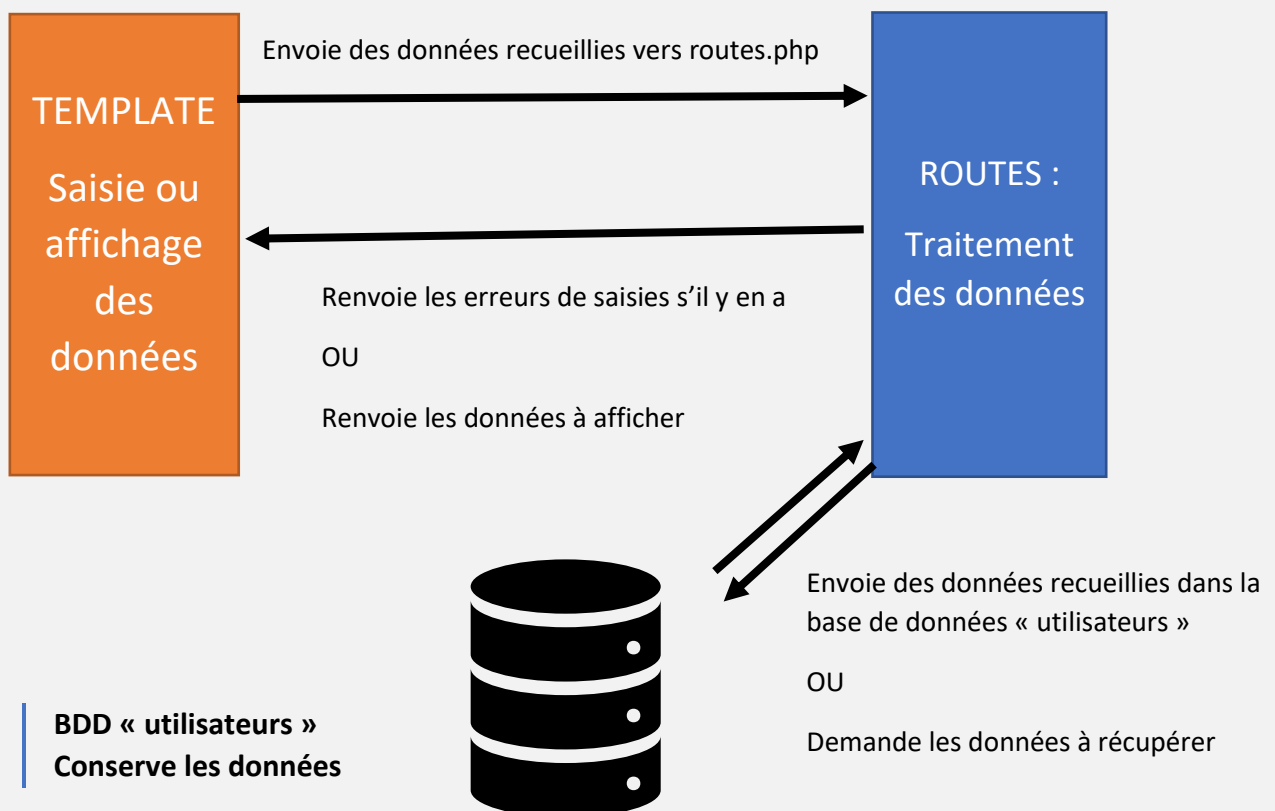
Le dossier « routes » contenant le fichier routes.php

Le dossier « templates » contenant tous les templates Smarty.

Deux fichiers extérieurs, dont l'index.php.

La base de données est dans le fichier data de laragorn

Le fichier pdo.php se trouve plus haut dans l'arbre de dossier.



6.Conclusion

Ce TP a permis l'apprentissage du fonctionnement des services web côté serveur.
L'utilisation de smarty a permis l'élaboration de ces derniers de manière simple et efficace.

De plus, j'ai eu l'occasion de comprendre comment était fait le système de session sur serveur, ainsi que celui de la gestion des mots de passe cryptés.

Parmis les services que l'on pourrais rajouter, nous pourrions développer une page pour modifier les données de la base de données.