

Débuter avec OpenID et PHP



par Vikram Vaswani Joris Crozier (Traducteur)

Date de publication : 17/09/2008

Dernière mise à jour : 17/01/2009

Cet article est la traduction de **Getting Started with OpenID and PHP** (disponible [ici](#)) et a pour but de vous faire découvrir OpenID et son utilisation.


| | |
|--|----|
| I - Mémoire insuffisante..... | 3 |
| II - Moi numérique..... | 3 |
| III - Assembler les pièces..... | 4 |
| IV - Première étape..... | 4 |
| V - Garder la simplicité..... | 8 |
| VI - Une question de stockage..... | 13 |
| VI - Une approche alternative..... | 15 |
| VIII - Le service avec le sourire..... | 16 |

I - Mémoire insuffisante

Quelque chose d'étrange est arrivé l'autre jour. Je surfais sur un site que je visite de façon irrégulière, je me suis authentifié avec mon login et password et cliqué sur le bouton submit. Après quelques secondes de clics et rectifications, la machine m'a informé que la vérification de mon password avait échouée. Perplexe, j'ai essayé encore plusieurs fois mais, ne rencontrant aucun succès, j'ai claqué des talons, prononcé quelques malédictions et suis retourné sur un travail plus productif. C'est seulement plus tard, après avoir répondu à quelques questions de sécurité et retrouvé mon password original, que j'ai réalisé ma gaffe, j'ai utilisé un password appartenant à un autre site.

Si cette histoire vous semble familière, c'est parce que c'est l'histoire du web d'aujourd'hui, trop de sites, trop de login et pas assez de capacité crânienne pour tous les classer précisément. Mais il y a de bonnes nouvelles à portée de la main : OpenID, un Framework libre et open source pour les "connexions singulières" à travers différents sites et applications. Les nouvelles encore meilleures ? Il existe déjà une liasse de widgets PHP qui permettent au développeur d'intégrer facilement OpenID dans les applications PHP et cet article va vous montrer comment les utiliser. Alors qu'est-ce que vous attendez ? Donnez un petit coup à la page et allons-y !

II - Moi numérique

Avant de plonger dans le code, passons quelques minutes à répondre à une question de base : Quelle est cette chose 'OpenID' et comment marche-t-il ? Selon le site Web officiel,  **OpenID** est :



"Une façon simple d'utiliser une seule identité numérique à travers l'Internet". Fondamentalement, un OpenID est une URL personnalisée, choisi par vous comme votre identité en ligne et enregistré avec un prestataire de services OpenID. Chaque fois qu'un site externe doit vérifier votre identité pour des buts d'établissement de la connexion, vous fournissez cette URL au lieu de votre username; le site entre alors en contact avec votre prestataire de services OpenID pour l'authentification. Quel est le bénéfice ? Simple car votre OpenID est stocké chez votre fournisseur de services OpenID et n'importe quel site peut contacter ce fournisseur pour vous authentifier. Il n'y a pas besoin de créer de multiples comptes ou de vous rappeler une multitude de login et password pour des sites différents, tout ce que vous avez besoin c'est un simple OpenID. Cela veut dire, bien sûr, que le site externe soutient la structure OpenID; l'adoption de celui-ci augmente progressivement, et le site Web OpenID a quelques informations intéressantes sur les grandes organisations diverses qui ont commencé à utiliser la structure.

Typiquement il y a deux parties à une transaction OpenID : Consommateur et Fournisseur. Un Fournisseur ressemble à un conservateur : il permet aux utilisateurs de créer et enregistrer des URL OpenID et gère leur compte OpenID, et il authentifie aussi l'utilisateur en tant que Consommateur sur demande. Un Consommateur (aussi parfois appelé un Parti d'Espoir) est un site Web OpenID permis.

La structure OpenID est complètement open source et n'importe quel site Web peut devenir un Consommateur ou un Fournisseur d'OpenID sans encourir n'importe quels coûts sur des honoraires de licence. En conséquence, il y a déjà un grand nombre de Fournisseurs OpenID sur le Web et un nombre croissant de sites Web ont commencé à permettre aux utilisateurs de souscrire à leurs services en utilisant OpenID.



Que se passe-t'il lors d'une transaction OpenID ? Bien, quand un utilisateur essaie d'enregistrer dans un site Grand public avec un OpenID, le Consommateur entre en contact avec le Fournisseur pour vérifier les lettres de créance de l'utilisateur avant la permission de celui-ci ou son accès. L'utilisateur peut être redirigé vers le Fournisseur et prié de s'enregistrer en arrivant sur son compte avec le Fournisseur en utilisant un mot de passe; une fois que c'est fait avec succès, le Fournisseur redirige automatiquement l'utilisateur sur le site Grand public, qui maintenant traite l'utilisateur comme vérifié et lui donne les autorisations nécessaires. Une clef partagée, connue des deux partis et fortement cryptée est utilisée pour maintenir l'intégrité de la transaction et évite les "escroqueries".

Si vous êtes nouveau a OpenID, les informations ci-dessus devraient être suffisantes pour expliquer le concept de base et assurer que vous pourrez suivre la matière ce qui suit. Quoi qu'il en soit si vous désirez/avez besoin d'informations détaillées, je vous recommande de jeter un oeil sur le site des développeurs de l'OpenID à l'adresse :

 <http://openid.net/developers/> ou les  **spécifications** de l'OpenID.

III - Assembler les pièces

Maintenant que vous avez (espérons-le) compris les bases du fonctionnement du Framework OpenID, tournons-nous vers une question plus précise : où est-ce que PHP intervient ? Bien, il y a un bon nombre de bibliothèques écrites pour PHP et faites pour aider le développeur à ajouter rapidement le support OpenID sur son application, ce tutoriel en montre deux d'entre elles :

- La bibliothèque PHP d'OpenID ( <http://www.openidenabled.com/>), maintenue par JanRain Inc (JanRain Inc opère aussi sur le site MyOpenID.com, un fournisseur populaire pour les identités OpenID). C'est une implémentation stable pour les clients et serveurs finaux d'une connexion OpenID, et est utilisée dans la plupart des exemples de ce tutoriel.
- Le package PEAR d'authentification OpenID ( <http://pear.php.net/pepr/pepr-proposal-show.php?id=500>) proposé par Pádraic Brady. A noter que ce package est encore au stade de la proposition à l'heure où j'écris ces lignes et devrait bientôt être considéré au stade alpha. Elle est utilisée brièvement dans ce tutoriel pour montrer une alternative à la bibliothèque de JanRain Inc

Dans le cas où vous ne les avez pas encore, vous avez aussi besoin des packages PEAR suivants :

- Le package PEAR DB ( <http://pear.php.net/package/DB>)
- Le package Crypt_HMAC2 ( http://pear.php.net/package/Crypt_HMAC2)
- Le package Crypt_DiffieHellman ( http://pear.php.net/package/Crypt_DiffieHellman)
- Le package Services_Yadis ( http://pear.php.net/package/Services_Yadis)

Vous pouvez installer ces packages à la main, ou en utilisant l'installateur PEAR comme suit :

```
shell> pear install Crypt_HMAC2
```

Dans le but d'essayer les exemples fournis dans le tutoriel, vous aurez aussi besoin de votre propre OpenID, allez en prendre un sur le site <http://www.myopenid.com/> ou chez n'importe quel autre fournisseur de service OpenID (et rappelez-vous que vous pouvez l'utiliser sur n'importe quel site relais d'OpenID) Si vous utilisez le service MyOpenID, votre OpenID sera certainement de la forme <http://vousmeme.myopenid.com> et sera généré pour vous gratuitement.

Une fois que vous avez toutes les pièces ensemble, vous êtes prêts. Mais avant je dois vous donner un avertissement important : je ne suis pas un expert en OpenID et ce tutoriel ne prétend pas à être une référence exhaustive sur l'implémentation de OpenID (les spécifications et les bibliothèques clients changent trop rapidement pour attendre un but complet). Quoiqu'il en soit il prétend à être une introduction générale pour les développeurs PHP qui sont débutants en OpenID, pour leur donner une idée générale de comment l'intégration PHP/OpenID fonctionne et monter leur niveau de confort avec ces technologies. Pour cette raison, j'ai gardé le code suivant très simple : souvenez-vous que vous pouvez trouver d'autres exemples plus complexes dans la documentation fournie avec les bibliothèques clientes proposées précédemment.

Cet avertissement donné, nous pouvons commencer.

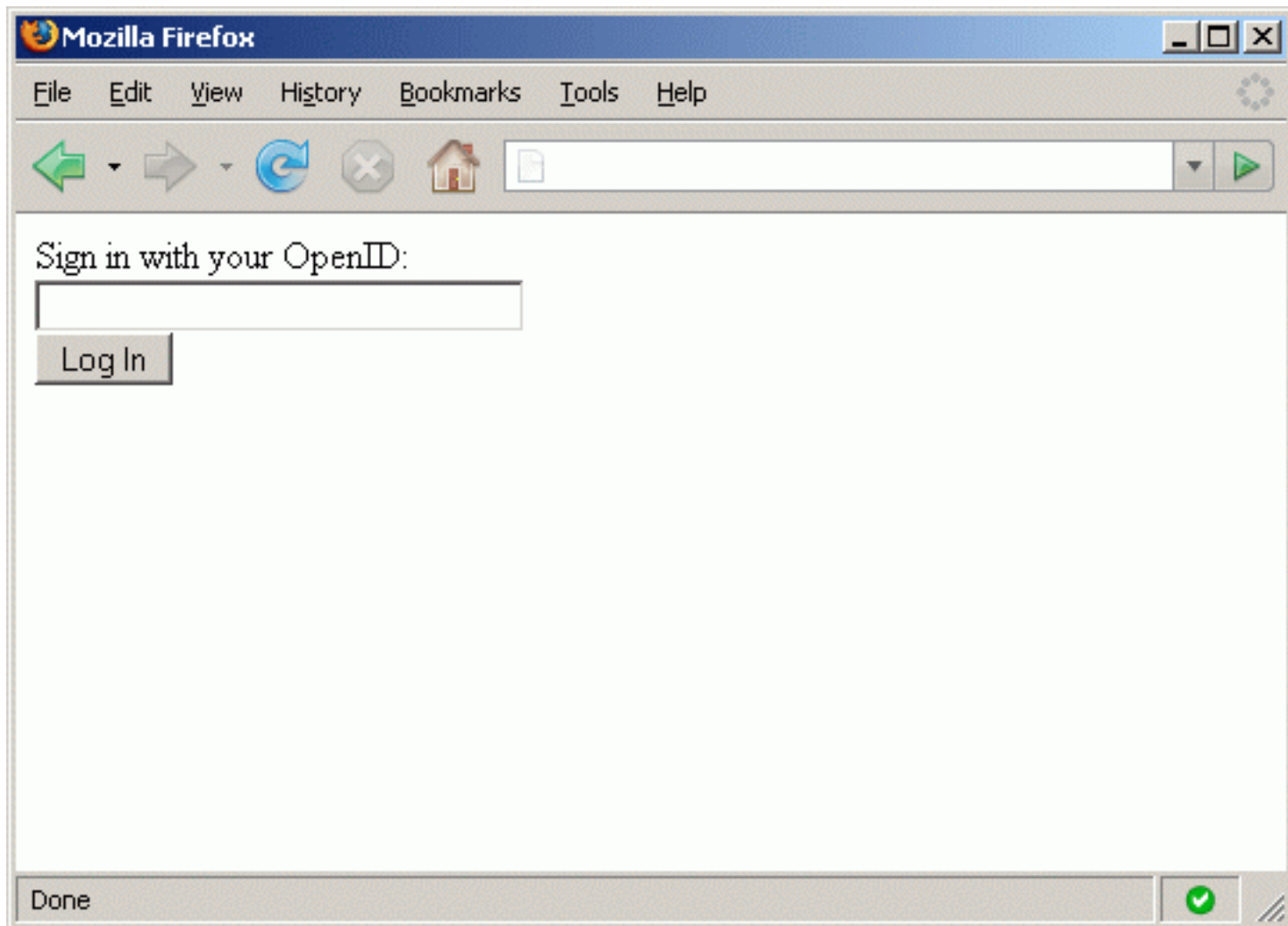
IV - Première étape

La première chose dont vous avez besoin, si vous voulez commencer à accepter OpenID sur votre site, c'est un formulaire d'authentification. Voici le code

```
<form method="post">
  Entrez votre OpenID: <br/>
  <input type="text" name="id" size="30" />
<br />
```

```
<input type="submit" name="submit" value="Log In" />
</form>
```

Voilà a quoi le formulaire ressemble:



Vous noterez que le formulaire d'authentification ne comporte pas de champs password. C'est parce que dans le Framework d'OpenID, l'authentification est gérée par le fournisseur OpenID: tout ce dont un utilisateur a besoin pour accéder à un site Consommateur est son OpenId.

Quand l'utilisateur valide ce formulaire avec son OpenID, le process du formulaire a besoin de localiser le fournisseur OpenID et de le rediriger vers celui-ci pour l'authentification. La bibliothèque PHP OpenID peut s'en charger pour vous. Regardez le code suivant, qui enveloppe le formulaire ci-dessus dans un test conditionnel et ajoute le code qui fonctionne sur la soumission du formulaire. (Je pars du principe que votre site - le site consommateur - est situé à l'adresse <http://consumer.example.com> mais soyez libre de le changer pour <http://localhost> pour vos essais.)

```
<?php
if (!isset($_POST['submit'])) {
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title></title>
```

```
</head>
<body>
  <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
    Entrez votre OpenID: <br/>
    <input type="text" name="id" size="30" />
    <br />
    <input type="submit" name="submit" value="Log In" />
  </form>
</body>
</html>
<?php
} else {
  // vérifie les valeurs du formulaire
  if (trim($_POST['id'] == '')) {
    die("ERROR: Entrez un OpenID valide svp.");
  }

  // fichiers inclus
  require_once "Auth/OpenID/Consumer.php";
  require_once "Auth/OpenID/FileStore.php";

  // démarrage de la session (requis pour YADIS)
  session_start();

  // crée une zone de stockage pour les données OpenID
  $store = new Auth_OpenID_FileStore('./oid_store');

  // crée un consommateur OpenID
  $consumer = new Auth_OpenID_Consumer($store);

  // commence le process d'authentification
  // crée une requête d'authentification pour le fournisseur OpenID
  $auth = $consumer->begin($_POST['id']);
  if (!$auth) {
    die("ERROR: Entrez un OpenID valide svp.");
  }

  // redirige vers le fournisseur OpenID pour l'authentification
  $url = $auth->redirectURL('http://consumer.example.com/', 'http://consumer.example.com/oid_return.php');
  header('Location: ' . $url);
}
?>
```

Comme expliqué précédemment, l'authentification avec OpenID est un process en deux étapes : premièrement le consommateur contacte le fournisseur avec l'OpenID de l'utilisateur et ensuite , le fournisseur procède à l'authentification et redonne le contrôle au consommateur pour compléter l'authentification.

Le script ci-dessus procède a la première étape de ce process. Une fois le formulaire soumis et validé, une nouvelle session PHP démarre et deux instances d'objets sont créées: Auth_OpenID_FileStore et Auth_OpenID_Consumer. Auth_OpenID_FileStore représente un emplacement sur le disque que la bibliothèque PHP OpenID utilisera pour stocker les données relatives a la tentative d'authentification courante. Le nom du répertoire à utiliser devrait être passé au constructeur de l'objet. (Cela tentera de créer le répertoire si il n'existe pas). L'objet Auth_OpenID_Consumer représente un consommateur OpenID et l'objet Auth_OpenID_FileStore génère précédemment doit être passé a son constructeur.

Pour commencer le processus d'authentification, le script appelle la méthode begin() de l'objet Auth_OpenID_Consumer, passé par l'OpenID de l'utilisateur. La valeur de retour de cette méthode est un objet Auth_OpenID_AuthRequest, qui représente une requête d'authentification. La méthode redirectURL() de cet objet est ensuite invoquée avec deux arguments : l'URL utilisée pour identifier votre site chez le fournisseur OpenID, et l'URL a laquelle le fournisseur OpenID doit rendre le contrôle après l'authentification. La valeur de retour de redirectURL() est une URL , utilisée pour rediriger le navigateur de l'utilisateur vers le fournisseur OpenID du site.

A ce point, le contrôle transfère vers le fournisseur OpenID, qui a besoin que l'utilisateur s'authentifie avec son password. Une fois ce processus complété, le fournisseur OpenID redirige le navigateur de l'utilisateur vers l'URL passée en second argument de la méthode `redirectURL()` - dans cet exemple le script `oid_return.php` du domaine consommateur. Typiquement, le fournisseur OpenID va aussi attacher quelques informations à la chaîne demandée en tant que paramètres GET, celles-ci sont utilisées par le consommateur pour terminer l'authentification.

Regardons maintenant ce que le script `oid_return.php` fait :

```
<?php
// Fichiers inclus
require_once "Auth/OpenID/Consumer.php";
require_once "Auth/OpenID/FileStore.php";

// démarre la session (requis pour YADIS)
session_start();

// crée une zone de stockage pour les données OpenID
$store = new Auth_OpenID_FileStore('./oid_store');

// crée un consommateur OpenID
// Lit la réponse du fournisseur OpenID
$consumer = new Auth_OpenID_Consumer($store);
$response = $consumer->complete('http://consumer.example.com/oid_return.php');

// renseigne les valeurs en fonction de celles de l'authentification
if ($response->status == Auth_OpenID_SUCCESS) {
    $_SESSION['OPENID_AUTH'] = true;
} else {
    $_SESSION['OPENID_AUTH'] = false;
}

// redirige vers la zone restreinte
header('Location: restricted.php');
?>
```

La première moitié de ce script est similaire à ce que nous avons déjà vu. Il initialise l'espace de stockage et crée l'objet `Auth_OpenID_Consumer`. Il appelle ensuite la méthode `complete()` de l'objet en lui passant l'URL de redirection du fournisseur d'accès. La valeur de retour de la méthode `complete()` de l'objet `Auth_OpenID_ConsumerResponse` qui représente la réponse du fournisseur OpenID à la requête d'authentification. Quatre codes de réponse sont possibles :

- `Auth_OpenID_SUCCESS` qui indique que l'authentification a été un succès
- `Auth_OpenID_FAILURE` qui indique que l'authentification a échoué
- `Auth_OpenID_CANCEL` qui indique que l'authentification a été abandonnée par l'utilisateur
- `Auth_OpenID_SETUP_NEEDED` qui apparaît seulement si le serveur OpenID devait authentifier de façon non interactive et en a été incapable.

Dans notre exemple ci-dessus, une fois l'authentification complète, une variable de session nommée `$_SESSION['OPENID_AUTH']` est initialisée avec un booléen indiquant si l'authentification a réussi ou non. Il est maintenant simple d'utiliser cette variable de session pour les pages d'authentification en surveillant celle-ci sur toutes les pages et seulement autoriser l'accès de l'utilisateur si celle-ci est à `True`. Voici un exemple simple sur la façon de l'implémenter sur une page restreinte (`restricted.php`)

```
<?php
// Vérifie le statut de l'authentification
session_start();
if (!isset($_SESSION['OPENID_AUTH']) || $_SESSION['OPENID_AUTH'] !== true) {
    die ('Vous n'avez pas le droit d'accéder à cette page! Loggez-vous svp.');
```

```

"DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title></title>
  </head>
  <body>
    <h2>Page restreinte</h2>
    <p>Vous voyez cette page seulement car l'authentification OpenID a fonctionnée.</p>
  </body>
</html>

```

Si un utilisateur essaie d'accéder a cette page sans avoir authentifié son OpenID , \$_SESSION['OPENID_AUTH'] devrait ne pas exister ou être a False et l'utilisateur verra simplement un message d'erreur. C'est seulement après une authentification réussie avec le fournisseur OpenID que \$_SESSION['OPENID_AUTH'] passe a True et l'utilisateur sera en mesure de voir la page restreinte.

V - Garder la simplicité

Quand vous créez pour un OpenID, il y a des chances pour qu'on vous demande pour des informations personnelles optionnelles, incluant votre nom, votre adresse email, votre langue et votre pays de résidence. Les spécifications OpenID incluent des dispositions pour le consommateur pour récupérer ces informations chez le fournisseur pendant le process d'authentification. Cette extension appelée d'enregistrement simple est pleinement supportée dans la bibliothèque PHP d'OpenID, et la révision suivante du code précédant illustre ceci :

```

<?php
// fichiers requis
require_once "Auth/OpenID/Consumer.php";
require_once "Auth/OpenID/FileStore.php";
require_once "Auth/OpenID/SReg.php";

// démarra de session (requis pour YADIS)
session_start();

// crée une zone de stockage pour les données OpenID
$store = new Auth_OpenID_FileStore('./oid_store');

// crée un consommateur OPenID
// lit la reponse depuis le fournisseur OpenID
$consumer = new Auth_OpenID_Consumer($store);
$response = $consumer->complete('http://consumer.example.com/oid_return.php');

// crée une variable de session qui dépend de l'authentification
if ($response->status == Auth_OpenID_SUCCESS) {
    $_SESSION['OPENID_AUTH'] = true;

    // récupère les informations d'enregistrement
    $sreg = new Auth_OpenID_SRegResponse();
    $obj = $sreg->fromSuccessResponse($response);
    $data = $obj->contents();

    // fais quelques choses avec lesdites informations
    // ...
} else {
    $_SESSION['OPENID_AUTH'] = false;
}

// redirige vers la page a accès restreint
header('Location: restricted.php');
?>

```

Dans une requête simple d'enregistrement , un consommateur peut demander pour n'importe laquelle des 8 informations : pseudo , adresse email , nom complet, date de naissance au format YYYY-MM-DD, genre , code postal, pays de résidence , langage , zone de temps. Chacune de ces informations est représentée par une clef. Par exemple 'dob' pour date de naissance ou 'email' pour l'adresse email. Pour créer une requête simple d'enregistrement en utilisant la bibliothèque PHP d'OpenID, appelez la méthode build() de la classe statique Auth_OpenID_SRegRequest avec deux tableaux en tant qu'arguments. Le premier tableau liste les clefs requises tandis que l'autre liste els clefs optionnelles. Dans l'exemple ci dessus, le nom complet de l'utilisateur, la date de naissance et le langage sont requis. Le pseudo de l'utilisateur est optionnel. Supposons l'authentification réussie, les attributs du profil demandé, si disponible chez le fournisseur OpenID, sont retournés au consommateur, avec les autres paramètres demandés. Elles peuvent être ensuite retirées en tant que tableau associatif en initialisant une instance de la classe Auth_OpenID_SRegResponse avec le paquet de réponse et en appelant la méthode contents() de l'instance, comme illustré ci dessous :

```
<?php
// fichiers requis
require_once "Auth/OpenID/Consumer.php";
require_once "Auth/OpenID/FileStore.php";
require_once "Auth/OpenID/SReg.php";

// démarra de session (requis pour YADIS)
session_start();

// crée une zone de stockage pour les données OpenID
$store = new Auth_OpenID_FileStore('./oid_store');

// crée un consommateur OpenID
// lit la reponse depuis le fournisseur OpenID
$consumer = new Auth_OpenID_Consumer($store);
$response = $consumer->complete('http://consumer.example.com/oid_return.php');

// crée une variable de session qui dépend de l'authentification
if ($response->status == Auth_OpenID_SUCCESS) {
    $_SESSION['OPENID_AUTH'] = true;

    // récupère les informations d'enregistrement
    $sreg = new Auth_OpenID_SRegResponse();
    $obj = $sreg->fromSuccessResponse($response);
    $data = $obj->contents();

    // fais quelques choses avec lesdites informations
    // ...
} else {
    $_SESSION['OPENID_AUTH'] = false;
}

// redirige vers la page a accès restreint
header('Location: restricted.php');
?>
```

Il est maintenant facile d'utiliser ces informations de profil , dans le workflow du site consommateur - par exemple pour enregistrer automatiquement un utilisateur avec son nom ou pour envoyer un mail a son adresse. Pour illustrer considérons cette version améliorée du script précédent qui utilise l'adresse mail récupérée via le fournisseur OpenID pour vérifier si l'utilisateur a déjà un compte chez le fournisseur OpenID. Si la réponse est oui, un message d'accueil personnalisé est affiché avec l'adresse mail de l'utilisateur. Sinon un nouveau formulaire d'inscription est affiché dont les champs sont pré-remplis avec les informations récupérées depuis le profil de l'utilisateur.

Voici le code :

```
<?php
// fichiers inclus
require_once "Auth/OpenID/Consumer.php";
```

```

require_once "Auth/OpenID/FileStore.php";
require_once "Auth/OpenID/SReg.php";

// démarrage de session (requis pour YADIS)
session_start();

// crée une zone de stockage pour les données OpenID
$store = new Auth_OpenID_FileStore('./oid_store');

// crée un consommateur OpenID
// lit la réponse depuis le fournisseur OpenID
$consumer = new Auth_OpenID_Consumer($store);
$response = $consumer->complete('http://consumer.example.com/oid_return.php');

// crée une variable de session qui dépend de l'authentification
if ($response->status == Auth_OpenID_SUCCESS) {
    $_SESSION['OPENID_AUTH'] = true;

    // récupère les informations d'enregistrement
    $sreg = new Auth_OpenID_SRegResponse();
    $obj = $sreg->fromSuccessResponse($response);
    $data = $obj->contents();

    if (isset($data['email'])) {
        // Si l'adresse mail est disponible
        // Vérifie si l'utilisateur a déjà un compte sur le système

        // ouvre une connexion à la base
        $conn = mysql_connect('localhost', 'user', 'pass') or die('ERROR: Connexion serveur impossible');
        mysql_select_db('test') or die('ERROR: Impossible de sélectionner une base');

        // exécute la requête
        $result = mysql_query("SELECT DISTINCT COUNT(*) FROM users WHERE email = '" . $data['email'] . "'") or die('ERR

        $row = mysql_fetch_array($result);
        if ($row[0] == 1) {
            // si oui affiche un message personnalisé
            $newUser = false;
            echo 'Bonjour et bienvenue, ' . $data['email'];
            exit();
        } else {
            // si non avertit que l'utilisateur est nouveau
            $newUser = true;
        }

        // ferme la connexion
        mysql_free_result($result);
        mysql_close($conn);
    } else {
        // si l'adresse email n'est pas disponible
        // avertit que l'utilisateur est nouveau
        $newUser = true;
    }
} else {
    $_SESSION['OPENID_AUTH'] = false;
    die ('Vous n'avez pas la permission d'accéder à cette page! Re-loggez vous svp.');
```

```

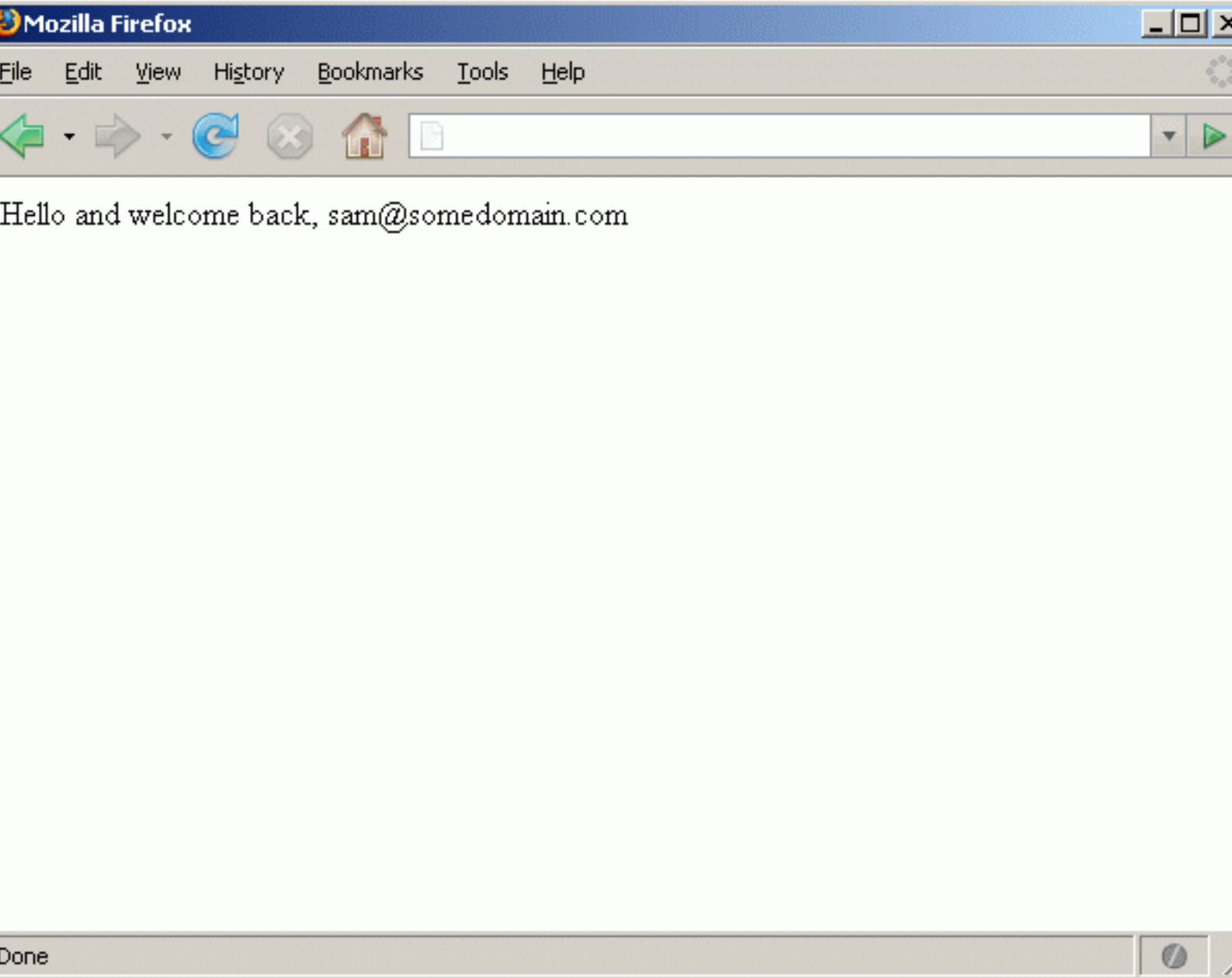
}

// Si l'utilisateur n'a pas de compte,
// ou si l'adresse email n'est pas disponible
// avertit que c'est un nouvel enregistrement
// affiche un formulaire d'inscription avec les champs pré-remplis
if ($newUser == true) {
    ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title></title>
    </head>

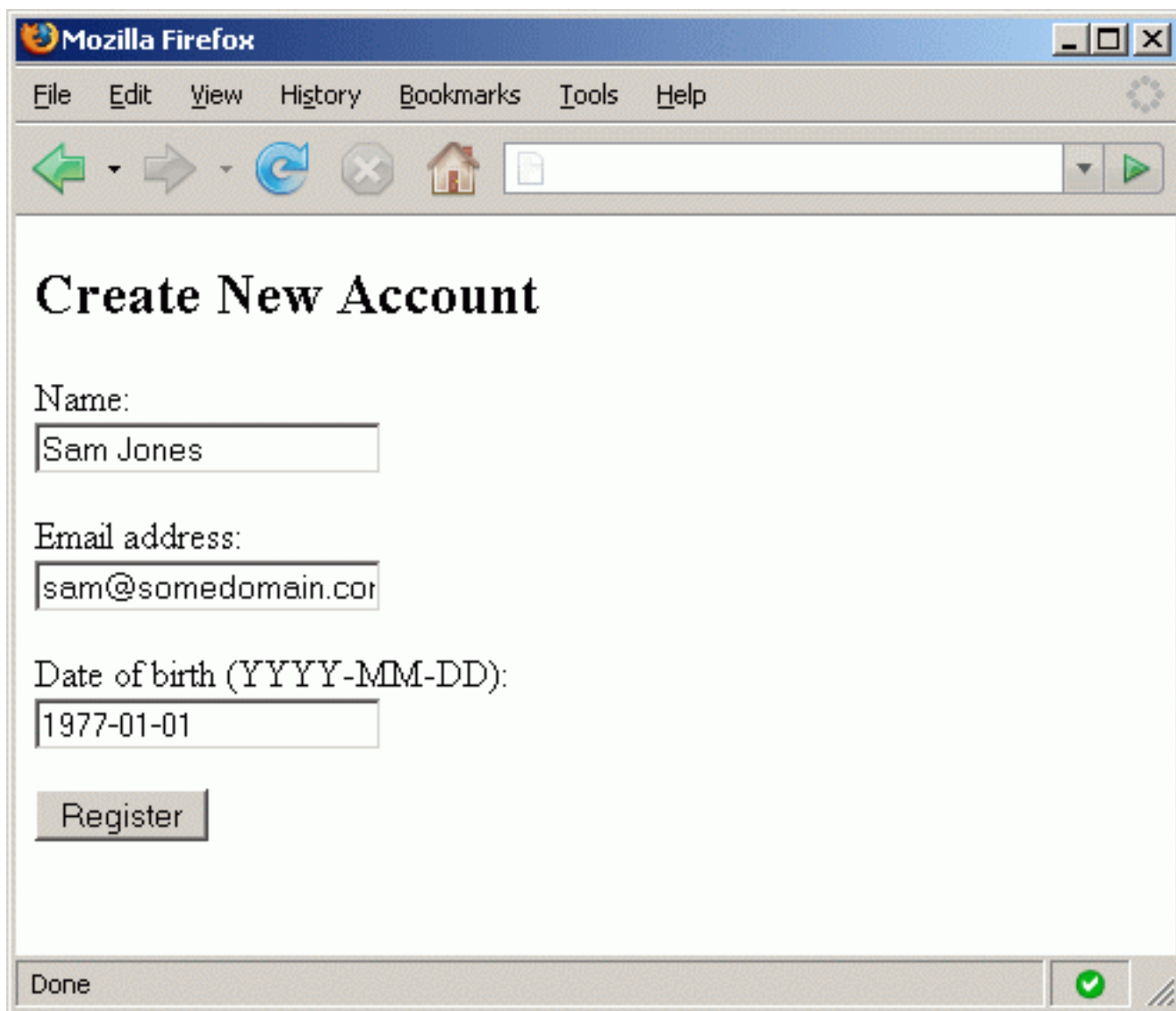
```

```
<body>
<h2>Créer un nouveau compte</h2>
<form method="post" action="register.php">
  Name: <br/>
  <input type="text" name="name" value="<?php echo @$data['fullname']; ?>" />
  <p />
  Adresse mail: <br/>
  <input type="text" name="email" value="<?php echo @$data['email']; ?>" />
  <p />
  Date de naissance (YYYY-MM-DD): <br/>
  <input type="text" name="dob" value="<?php echo @$data['dob']; ?>" />
  <p />
  <input type="submit" name="submit" value="Enregistrer" />
</form>
</body>
</html>
<?php
}
?>
```

Voici ce qu'un utilisateur avec un compte sur le système doit voir après identification OpenID



Et voici ce qu'un nouvel utilisateur doit voir après identification OpenID



VI - Une question de stockage

L'exemple précédent a utilisé entièrement la classe de stockage de la bibliothèque PHP d'OpenID pour le stockage des données OpenID. Si ce n'est pas à votre goût, vous pouvez aussi stocker vos données OpenID dans une base MySQL, PostgreSQL ou SQLite en remplaçant l'objet Auth_OpenID_FileStore par les objets respectifs Auth_OpenID_MySQLStore, Auth_OpenID_PostgreSQLStore ou Auth_OpenID_SQLiteStore.

L'exemple suivant illustre l'utilisation d'une base MySQL

```
<?php
<?php
if (!isset($_POST['submit'])) {
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title></title>
  </head>
  <body>
    <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
      Entrez votre OpenID: <br/>
```

```

        <input type="text" name="id" size="30" />
        <br />
        <input type="submit" name="submit" value="Log In" />
    </form>
</body>
</html>
<?php
} else {
    // verifie les entrées du formulaire
    if (trim($_POST['id'] == '')) {
        die("ERROR: Veuillez entrer un OpenID valide.");
    }

    // fichier inclus
    require_once "Auth/OpenID/Consumer.php";
    require_once "Auth/OpenID/MySQLStore.php";
    require_once 'DB.php';

    // démarrage de session (requis pour YADIS)
    session_start();

    // crée un DSN pear pour MySQL
    $dsn = 'mysql://user:pass@localhost/openid';
    $options = array(
        'debug' => 2,
        'portability' => DB_PORTABILITY_ALL,
    );

    // Ouvre la connexion
    $db =& DB::connect($dsn, $options);
    if (PEAR::isError($db)) {
        die($db->getMessage());
    }

    // crée une zone de stockage pour les données OpenID
    $store = new Auth_OpenID_MySQLStore($db);
    $store->createTables();

    // crée un consommateur OpenID
    $consumer = new Auth_OpenID_Consumer($store);

    // débute le process d'authentification
    // crée une requête d'authentification auprès du fournisseur OpenID
    $auth = $consumer->begin($_POST['id']);
    if (!$auth) {
        die("ERROR: Entrez un OpenID valide svp.");
    }

    // redirige vers le fournisseur OpenID
    $url = $auth->redirectURL('http://consumer.example.com/', 'http://consumer.example.com/
oid_return.php');
    header('Location: ' . $url);
}
?>

```

Pour utiliser une base MySQL pour le stockage, initialisez un objet `Auth_OpenID_MySQLStore` et passez une connexion PEAR DB au constructeur de l'objet. Appeler la méthode `createTables()` de l'objet prend soin de créer les tables nécessaires et le reste du scénario continue comme auparavant. Quand vous utilisez les données retournées par le fournisseur OpenID dans le script de retour `oid_return.php`, n'oubliez pas de réutiliser l'objet `Auth_OpenID_MySQLStore` au lieu de `Auth_OpenID_FileStore`.

À propos, si vous voulez utiliser une bibliothèque d'abstraction quelconque au lieu de PEAR DB, vous pouvez le faire en surchargeant la classe `Auth_OpenID_DatabaseConnection` et en l'utilisant avec votre toolkit d'abstraction. Similairement si vous voulez utiliser un mécanisme de stockage autre que par fichier ou base SQL comme base de vos sources personnalisées. Pour plus d'information là-dessus, regardez dans la documentation de la bibliothèque PHP d'OpenID de JanRain.

VI - Une approche alternative

Une implémentation consommateur alternative d'OpenID est fournie par le package PEAR Authentication::OpenID_Consumer, à l'heure où j'écris il est en version alpha, mais il est intéressant de le voir en action malgré tout. (Notez que vous devez mettre le niveau de report d'erreur de PHP pour ignorer les avertissements et les notes renvoyées par le package. Ceci devrait être corrigé dans la version finale). Considérons le code suivant qui est équivalent au premier exemple du tutoriel.

```

<?php
error_reporting(E_ERROR);
if (!isset($_POST['submit'])) {
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title></title>
</head>
<body>
    <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
        Entrez votre OpenID: <br/>
        <input type="text" name="id" size="30" />
        <br />
        <input type="submit" name="submit" value="Log In" />
    </form>
</body>
</html>
<?php
} else {
    // vérifie les valeurs du formulaire
    if (trim($_POST['id'] == '')) {
        die("ERROR: Entrez un OpenId valide svp.");
    }

    // fichier requis
    require_once 'OpenID/Consumer.php';
    require_once 'OpenID/Store/File.php';

    // démarrage de session
    session_start();

    // création du stockage
    $store = new OpenID_Store_File('./oid_store');

    // création du consommateur
    $consumer = new OpenID_Consumer($store);

    // débute le process d'authentification
    // crée une requête d'authentification vers le fournisseur OpenID
    $auth = $consumer->start($_POST['id']);
    if (!$auth)
        die {Error: Entrez un OpenID valide svp.");
    }

    // redirige vers le fournisseur OpenID pour authentification
    $auth->redirect('http://consumer.example.com/oid_return.php', 'http://consumer.example.com/');
}
?>

```

Mis à part les différences de notation, ceci est remarquablement similaire à ce que vous avez vu dans l'exemple précédent, le script crée un fichier de stockage, initialise un objet consommateur à partir du stockage, et appelle la méthode start() de l'objet Consommateur avec l'OpenID de l'utilisateur pour commencer le process d'authentification auprès du fournisseur OpenID. La méthode redirect() de l'objet Autorisation résultant qui accepte l'URL de retour et l'identifiant du site, prend soin ensuite de rediriger le navigateur de l'utilisateur jusqu'au site du fournisseur OpenID.

Une fois que l'utilisateur a complété son authentification, le fournisseur OpenID redonne le contrôle au script consommateur `oid_return.php`

```
<?php
error_reporting(E_ERROR);
// fichiers inclus
require_once 'OpenID/Consumer.php';
require_once 'OpenID/Store/File.php';

// démarrage de la session
session_start();

// création stockage
$store = new OpenID_Store_File('./oid_store');

// création consommateur
$consumer = new OpenID_Consumer($store);

// crée la variable de session en fonction de l'authentification
if (isset($_GET)) {
    $response = $consumer->finish($_GET);
    $result = $response->getResult();
    if ($result == 'success') {
        $_SESSION['OPENID_AUTH'] = true;
    } else {
        $_SESSION['OPENID_AUTH'] = false;
    }
}




// redirige vers la page restreinte
header('Location: restricted.php');
?>
```

Ici, la méthode `finish()` de l'objet consommateur est utilisée pour compléter le process d'authentification. Le résultat de la méthode `finish()` est un objet réponse qui représente la réponse envoyée par le fournisseur OpenID vers la requête d'authentification du consommateur. La méthode `getResult()` de cet objet peut être utilisée pour tester le résultat du process d'authentification, et effectue les actions appropriées qui dépendent du résultat succès ou non.

VIII - Le service avec le sourire

Maintenant que vous avez vu deux implémentations différentes du consommateur OpenID, tournons notre attention vers la fin de connexion du fournisseur, d'abord nous devons noter que la bibliothèque OpenID PHP de JanRain utilisée dans le tutoriel intègre une version complète du serveur OpenID que vous pouvez utiliser pour créer votre code personnalisé de fournisseur OpenID. Un exemple de serveur est inclus dans le package pour vous aider à commencer.

Dans la plupart des cas, vous pouvez vous en tirer sans coder votre propre serveur OpenID spécialement si vos besoins sont simples. Il existe plusieurs packages open-source qui vous permettent d'installer et de gérer votre propre serveur OPENID, en voici une petite liste :

-  **phpMyID**
-  **SimpleID**
-  **Clamshell**

Et ce sera tout pour cet article. A travers ces quelques lignes je vous ai donné un aperçu d'OpenID expliquant ce que c'était et comment ça fonctionnait. Je vous ai aussi emmené dans l'installation de quelques bibliothèques PHP pour OpenID et montré comment vous pouviez implémenter OpenID dans vos applications. Je vous ai aussi montré l'extension d'enregistrement simple d'OpenID, vous permettant de récupérer des informations sur le profil

d'un utilisateur et les réinjecter dans votre application. Finalement je vous ai montré une alternative basée sur l'implémentation PEAR de consommateur d'OpenID, et redirigé vers des packages faciles d'utilisation pour installer votre propre serveur OpenID

Si vous voulez en savoir plus sur PHP et OpenID, vous pouvez regarder ces ressources :

-  **An OpenID and PHP primer**
-  **The OpenID developer site**
-  **Plaxo's guide for OpenID-Enabling a Web site**
-  **OpenID for Non-Superusers**
-  **OpenID implementations for other programming languages**

Amusez-vous bien, et bon code !