PHP Authentication 1.3

You need a PHP authentication library for your new AJAX web site? Look no further. PHP Authentication is a web service accessed via POST requests, producing XML or JSON output. Completely customizable, easily integrates with the client side. Source code included.

Features:

- standard user registration,
- invite-only registration,
- user invitations,
- many more.

Table of Contents

1 Installation instructions

Step by step instructions:

1.1 Extract the zip file

1.2 Set database connection parameters

PHP Authentication service uses database to store data. All most frequently used databases are supported (MySQL, PostgreSQL, SQLite). Database is not accessed directly but through EZPDO, lightweight data persistence library that comes with the product. It is not necessary for you to know anything about the mentioned library, you just need to edit the following configuration file:

```
./config.xml
```

Look for `<default_dsn>` tags and uncomment the one depending on the database type that you use. Put the right username, password and database name in connection string. Please note that the database already has to be created and user has to be given the right credentials, including the one necessary for table creation. The following MySQL statements might be helpful in that task:

```
CREATE DATABASE databasename DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, INDEX ON databasename . * TO
'username'@'localhost';

SET PASSWORD FOR 'username'@'localhost'=PASSWORD('password');
```

Syntax might differ slightly if you are using database other than MySQL.

1.3 Call `_buildTables.php` script

If you have set everything from the previous step correctly, this script will create the following tables: `LastLogin`, `MaskedCookieData`, `TempInvitation`, `TempVerification` and `User`. Also in local file system, folder `compiled` and file `ezpdo.log` will be created. After you are done with this step, you can delete `_buildTables.php` script.

1.4 Customize `verify.php` file

Link to this file is sent to user via e-mail upon registration as a request for account verification. You can customize this file and give it a desired look, just be sure to keep the `include_once` tag at the top and `$userMessage` variable wherever you want.

1.5 Customize `properties.ini` file

Properties file contains several sections, we'll describe each one right here.

1.5.1 Section `[General]`:

PHP Authentication comes with multi-language support. You select your preferred language by setting the `DefaultLanguage` parameter. You can add your own translation (or customize existing) by modifying the `messages.php` file.

Each registered user can send registration invitations to friends. Number of invitations that user can send is defined by parameter `NumberOfInvitations`.

If `CheckForRegistrationCode` parameter is set to 1, registration code is sent along with each invitation and new users can register only by providing that code. Think of it as of 'closed' web site support - new users can register only if they are invited by existing users. If `CheckForRegistrationCode` is set to 0, invitations can still be sent around but they will not contain registration code, since they are not needed now.

If `SendVerificationEMail` parameter is set to 1, e-mail with verification link is sent to user after his registration. User will not be able to log in until he verifies his account by opening this link. If parameter is set to 0, user's account will be verified on registration and verification mail will not be sent.

With `MessageFormat` parameter you define the format of PHP Authentication's output messages. The following two formats are available: `XML` (default) and `JSON`.

1.5.2 Section `[UserName]`:

`MinLength` and `MaxLength` define username minimum and maximum lengths, respectively. Maximum length can not be greater than 64 and username must be alphanumeric string. If `AllowEMailAddress` is set to 1, username can have e-mail address syntax and length constraints do not apply.

1.5.3 Section `[VerificationMail]`:

Upon registration, e-mail is sent to user containing the link for account verification. In order to verify the account, user is required to visit the link before the first login.

Link URL is defined with the parameter `Link`. Enter your hostname and path to `verify.php` script, leave the remaining part untouched.

Parameter `MailSubject` defines the subject of verification request e-mail, while parameters `MailBodyFile` and `MailHeadersFile` define paths to mail body and mail headers templates respectively. Customize templates as you wish, just don't forget to leave `{$link}` string inside the mail body.

1.5.4 Section `[LoginDataMail]`:

In case user forgets his username or password, he can make a request for forgotten credentials. Username and new password will be sent via e-mail to address that he used when he registered his account.

Parameter `MailSubject` defines the subject of e-mail, while parameters `MailBodyFile` and `MailHeadersFile` define paths to mail body and mail headers templates respectively. Customize templates as you wish, just don't forget to leave `{$userName}` and `{$password}` strings inside the mail body.

1.5.5 Section `[InvitationMail]`:

This section defines the look of invitation mail.

Parameter `MailSubject` defines the subject of e-mail which will contain inviter's name if the `{$name}` string is included. Parameter `MailBodyTailFile` defines the tail that will be appended to custom message that is inviter sending. If registration codes are used, this tail should contain `{$registrationCode}` string. Finally, parameter `MailHeadersFile` defines path to mail headers template file.

1.5.6 Section `[DaysToExpire]`:

In this section you can set the number of days after which the invitations and unverified accounts will expire. For that purpose you use `TempInvitation` and `TempVerification` parameters respectively. When the value is set to 0, data is not expired. Deletion of old data is actually done by the `deleteExpiredData.php` script.

1.5.7 Section `[EZPDO]`:

You don't need to modify this section unless you want to move EZPDO directory somewhere else, or you already have it somewhere on your system. In that case you should customize `RelativePath` parameter and make it point wherever you want. Just keep in mind that it is relative path, so keep the starting `./` and omit the trailing slash.

1.6 Set up cron to run `deleteExpiredData.php` once a day

Instructions for this step are different on each hosting environment. If you don't know how to set up scheduler to run the script in regular intervals, the best would be to ask your hosting provider for help.

That should be it! You are ready to start using PHP Authentication web service. Continue by checking the API.

2 API

2.1 Script name: `register.php`

2.1.1 POST Parameters:

`userName` - alphanumeric string. Minimum and maximum length are defined in `properties.ini` file.

`password` - alphanumeric string. Length 32 characters. It should be MD5 hash of user password obtained on client side.

`eMailAddress` - string, syntactically valid e-mail address.

`registrationCode` - alphanumeric string up to 10 characters long. It is needed only if it is defined so in `properties.ini` file.

2.1.2 Modified tables:

`TempInvitation` - if new user has been invited by one of the existing users, or registration code is mandatory, entry containing e-mail address or registration code is deleted from this table.

`TempVerification` - new entry is created containing verification code if the parameter `SendVerificationEMail` in section `[General]` of file `properties.ini` is set to 1.

`User` - new entry is created.

2.1.3 XML output on success:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="UserRegistration">
  <Error>false</Error>
  <UserName>anonymous</UserName>
  <Message>Account verification request sent to your e-mail
address</Message>
</XMLMessage>
```

or

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="UserRegistration">
    <Error>false</Error>
    <UserName>anonymous</UserName>
    <Message>User registered</Message>
</XMLMessage>
```

2.1.4 Possible XML outputs on failure:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="UserRegistration">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>Invalid input</Message>
</XMLMessage>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="UserRegistration">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>Unknown registration code</Message>
</XMLMessage>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="UserRegistration">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>User name not available or e-mail address already registered
in system</Message>
</XMLMessage>
```

2.1.5 JSON output on success:

```
{
 "type":"UserRegistration",
 "error":false,
 "userName":"anonymous",
 "message":["Account verification request sent to your e-mail address"]
}
```

or

```
{
  "type":"UserRegistration",
  "error":false,
  "userName":"anonymous",
  "message":["User registered"]
}
```

2.1.6 Possible JSON outputs on failure:

```
{
 "type":"UserRegistration",
 "error":true,
 "userName":"anonymous",
 "message":["Invalid input"]
}
```

```
{
 "type":"UserRegistration",
 "error":true,
 "userName":"anonymous",
 "message":["Unknown registration code"]
}
```

```
{
 "type":"UserRegistration",
 "error":true,
 "userName":"anonymous",
 "message":["User name not available or e-mail address already registered
in system"]
}
```

2.2 Script name: `unregister.php`

2.2.1 POST Parameters: None

2.2.2 Modified tables:

`LastLogin` - user's entry is deleted.

`User` - user's entry is deleted.

2.2.3 XML output on success:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="UnregisterUser">
  <Error>false</Error>
  <UserName>anonymous</UserName>
  <Message>User unregistered</Message>
</XMLMessage>
```

2.2.4 XML output on failure:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="UnregisterUser">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>In order to close account, you have to be logged in</Message>
</XMLMessage>
```

2.2.5 JSON output on success:

```
{
 "type":"UnregisterUser",
 "error":false,
 "userName":"anonymous",
 "message":["User unregistered"]
}
```

2.2.6 JSON output on failure:

```
{
 "type":"UnregisterUser",
 "error":true,
 "userName":"anonymous",
 "message":["In order to close account, you have to be logged in"]
}
```

2.3 Script name: `verify.php`

2.3.1 POST Parameters: None

2.3.2 GET Parameters:

`verificationCode` - string, 21 characters long.

2.3.3 Modified tables:

`TempVerification` - entry containing given verification code is deleted.

`User` - for specific user, a verification flag is set to true.

2.3.4 Output:

Script doesn't give any output, in fact it is customizable web page. More about customization can be read here.

2.4 Script name: `logIn.php`

2.4.1 POST Parameters:

`userName` - alphanumeric string. Minimum length 2, maximum 10 characters.

`password` - alphanumeric string. Length 32 characters. It should be MD5 hash of user password obtained on client side.

2.4.2 Modified tables:

`LastLogin` - upon login, timestamp and user's IP address is stored.

`MaskedCookieData` - user's login identifier that is kept in cookie is stored here as well.

2.4.3 XML output on success:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="LogIn">
  <Error>false</Error>
  <UserName>username</UserName>
  <Message>Logged in</Message>
</XMLMessage>
```

2.4.4 Possible XML outputs on failure:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="LogIn">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>Invalid input</Message>
</XMLMessage>

<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="LogIn">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>You haven't verified your account. Please visit the
verification link that has been sent to your e-mail address.</Message>
</XMLMessage>

<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="LogIn">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>Wrong username and/or password</Message>
</XMLMessage>
```

2.4.5 JSON output on success:

```
{
"type":"LogIn",
"error":false,
"userName":"username",
"message":["Logged in"]
}
```

2.4.6 Possible JSON outputs on failure:

```
{
"type":"LogIn",
"error":true,
"userName":"anonymous",
"message":["Invalid input"]
}


{
"type":"LogIn",
"error":true,
"userName":"anonymous",
"message":["You haven't verified your account. Please visit the
verification link that has been sent to your e-mail address."]
}

{
"type":"LogIn",
"error":true,
"userName":"anonymous",
"message":["Wrong username and/or password"]
}
```

2.5 Script name: *logOut.php*

2.5.1 POST Parameters: None

2.5.2 Modified tables:

MaskedCookieData - entry containing user's login identifier is deleted.

2.5.3 XML output:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="LogOut">
  <Error>false</Error>
  <UserName>anonymous</UserName>
  <Message>Logged out</Message>
</XMLMessage>
```

2.5.4 JSON output:

```
{
 "type":"LogOut",
 "error":false,
 "userName":"anonymous",
 "message":["Logged out"]
}
```

2.6 Script name: *requestLoginData.php*

2.6.1 POST Parameters:

eMailAddress - string, syntactically valid e-mail address.

2.6.2 Modified tables: None

2.6.3 XML output on success:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="RequestLoginData">
  <Error>false</Error>
  <UserName>anonymous</UserName>
  <Message>Account data sent to your e-mail address</Message>
</XMLMessage>
```

2.6.4 Possible XML outputs on failure:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="RequestLoginData">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>Invalid input</Message>
</XMLMessage>

<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="RequestLoginData">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>Your request can not be currently fulfilled. Please try again a
bit later.</Message>
</XMLMessage>

<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="RequestLoginData">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>Unknown e-mail address</Message>
</XMLMessage>
```

2.6.5 JSON output on success:

```
{
 "type":"RequestLoginData",
 "error":false,
 "userName":"anonymous",
 "message":["Account data sent to your e-mail address"]
}
```

2.6.6 Possible JSON outputs on failure:

```
{
 "type":"RequestLoginData",
 "error":true,
 "userName":"anonymous",
 "message":["Invalid input"]
}

{
 "type":"RequestLoginData",
 "error":true,
 "userName":"anonymous",
 "message":["Your request can not be currently fulfilled. Please try again
a bit later."]
}

{
 "type":"RequestLoginData",
 "error":true,
 "userName":"anonymous",
 "message":["Unknown e-mail address"]
}
```

2.7 *Script name:* *getCurrentUserName.php*

2.7.1 POST Parameters: None

2.7.2 Modified tables: None

2.7.3 XML output:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="GetCurrentUserName">
  <Error>false</Error>
  <UserName>username</UserName>
  <Message></Message>
</XMLMessage>
```

or:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="GetCurrentUserName">
  <Error>false</Error>
  <UserName>anonymous</UserName>
  <Message></Message>
</XMLMessage>
```

2.7.4 JSON output:

```
{
 "type":"GetCurrentUserName",
 "error":false,
 "userName":"username",
 "message":[]
}
```

or:

```
{
 "type":"GetCurrentUserName",
 "error":false,
 "userName":"anonymous",
 "message":[]
}
```

2.8 Script name: getUserName.php

2.8.1 POST Parameters:

`id` - user's id.

2.8.2 Modified tables: None

2.8.3 XML output on success:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="GetUserName">
  <Error>false</Error>
  <UserName>username with assigned id</UserName>
  <Message></Message>
</XMLMessage>
```

2.8.4 XML output on failure:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="GetUserName">
  <Error>true</Error>
  <UserName>username</UserName>
  <Message>User ID unknown</Message>
</XMLMessage>
```

2.8.5 JSON output on success:

```
{
"type":"GetUserName",
"error":false,
"userName":"username with assigned id",
"message":[]
}
```

2.8.6 JSON output on failure:

```
{
"type":"GetUserName",
"error":true,
"userName":"username",
"message":["User ID unknown"]
}
```

2.9 *Script name:* `setPrivateData.php`

2.9.1 POST Parameters:

`currentPassword` - alphanumeric string. Length 32 characters. It should be MD5 hash of user's current password obtained on client side.

`newPassword` - alphanumeric string. Length 32 characters. It should be MD5 hash of user's new password obtained on client side.

`eMailAddress` - string, syntactically valid e-mail address.

`gender` - integer, 0 = not set, 1 = male, 2 = female

`birthYear` - integer, 0 = not set, otherwise user must be between 5 and 120 years old.

2.9.2 Modified tables:

`User` - user entry is updated with given data.

2.9.3 XML output on success:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="SetPrivateData">
  <Error>false</Error>
  <UserName>username</UserName>
  <Message>Personal data changed</Message>
</XMLMessage>
```

2.9.4 Possible XML outputs on failure:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="SetPrivateData">
  <Error>true</Error>
  <UserName>username</UserName>
  <Message>Invalid input</Message>
</XMLMessage>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="SetPrivateData">
  <Error>true</Error>
  <UserName>username</UserName>
  <Message>Wrong current password</Message>
</XMLMessage>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="SetPrivateData">
  <Error>true</Error>
  <UserName>username</UserName>
  <Message>Personal data not changed - error occurred while saving the
changes. It is possible that new e-mail address has already been
registered.</Message>
</XMLMessage>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="SetPrivateData">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>In order to change personal data, you have to be logged
in</Message>
</XMLMessage>
```

2.9.5 JSON output on success:

```
{
"type":"SetPrivateData",
"error":false,
"userName":"username",
"message":["Personal data changed"]
}
```

2.9.6 Possible JSON outputs on failure:

```
{
"type":"SetPrivateData",
"error":true,
"userName":"username",
"message":["Invalid input"]
}
```

```
{
"type":"SetPrivateData",
"error":true,
"userName":"username",
"message":["Wrong current password"]
}
```

```
{
"type":"SetPrivateData",
"error":true,
"userName":"username",
"message":["Personal data not changed - error occurred while saving the
changes. It is possible that new e-mail address has already been
registered."]
}
```

```
{
"type":"SetPrivateData",
"error":true,
"userName":"anonymous",
"message":["In order to change personal data, you have to be logged in"]
}
```

2.10 Script name: `getPrivateData.php`

2.10.1 POST Parameters: None

2.10.2 Modified tables: None

2.10.3 XML output on success:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="GetPrivateData">
  <Error>false</Error>
  <UserName>username</UserName>
  <Message></Message>
  <PrivateData>
   <EMailAddress>e-mail address</EMailAddress>
   <Gender>0 (not set), 1 (male) or 2 (female)</Gender>
   <BirthYear>birth year</BirthYear>
  </PrivateData>
</XMLMessage>
```

2.10.4 XML output on failure:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="GetPrivateData">
  <Error>true</Error>
  <UserName>anonymous</UserName>
  <Message>In order to get personal data, you have to be logged
in</Message>
</XMLMessage>
```

2.10.5 JSON output on success:

```
{
"type":"GetPrivateData",
"error":false,
"userName":"username",
"message":[],
"privateData":
 {
 "eMailAddress":"e-mail address"
 "gender":"0 (not set), 1 (male) or 2 (female)"
 "birthYear":birth year
 }
}
```

2.10.6 JSON output on failure:

```
{
"type":"GetPrivateData",
"error":true,
"userName":"anonymous",
"message":["In order to get personal data, you have to be logged in"]
}
```

2.11 Script name: *sendInvitation.php*

2.11.1 POST Parameters:

`name` - string, at least 1 character long. As registered users send invitations to their friends, this could be inviter's first name or nick name.

`eMailAddress` - string, syntactically valid e-mail address. It should be invitees address.

`message` - string, at least 1 character long. Custom invitation message. It will be appended with the registration instructions. Read more about the invitation mail settings here.

2.11.2 Modified tables:

`TempInvitation` - new entry is created with e-mail address of the invited user and (optionally) registration code.

`User` - number of remaining invitations for particular user is decremented.

2.11.3 XML output on success:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="SendInvitation">
  <Error>false</Error>
  <UserName>username</UserName>
  <Message>Invitation with registration instructions is sent to given e-
mail address</Message>
</XMLMessage>
```

2.11.4 Possible XML outputs on failure:

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="SendInvitation">
  <Error>true</Error>
  <UserName>username</UserName>
  <Message>Invalid input</Message>
</XMLMessage>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="SendInvitation">
  <Error>true</Error>
  <UserName>username</UserName>
  <Message>Person with given e-mail address is already
registered</Message>
</XMLMessage>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLMessage type="SendInvitation">
  <Error>true</Error>
  <UserName>username</UserName>
  <Message>Person with given e-mail address is already invited</Message>
</XMLMessage>
```

2.11.5 JSON output on success:

```
{
 "type":"SendInvitation",
 "error":false,
 "userName":"username",
 "message":["Invitation with registration instructions is sent to given e-mail address"]
}
```

2.11.6 Possible JSON outputs on failure:

```
{
 "type":"SendInvitation",
 "error":true,
 "userName":"username",
 "message":["Invalid input"]
}

{
 "type":"SendInvitation",
 "error":true,
 "userName":"username",
 "message":["Person with given e-mail address is already registered"]
}

{
 "type":"SendInvitation",
 "error":true,
 "userName":"username",
 "message":["Person with given e-mail address is already invited"]
}
```

2.12 Script name: *`getRemainingInvitations.php`*

2.12.1 POST Parameters: None

2.12.2 Modified tables: None

2.12.3 XML output:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<XMLMessage type="RemainingInvitations">
  <Error>false</Error>
  <UserName>username</UserName>
  <Message></Message>
  <RemainingInvitations>number of remaining
invitations</RemainingInvitations>
</XMLMessage>
```

2.12.4 JSON output:

```
{
"type":"RemainingInvitations",
"error":false,
"userName":"username",
"message":[]
"remainingInvitations":number of remaining invitations
}
```

2.13 Script name: ***`deleteExpiredData.php`***

2.13.1 POST Parameters: None

2.13.2 Modified tables:

`TempInvitation` - all unused invitations that have expired are deleted from the table. Expiration periods are set in `properties.ini` file.

`TempVerification` - all verification codes that have expired are deleted from the table. Expiration periods are set in `properties.ini` file.

`User` - all user accounts that haven't been verified are deleted from the table. Expiration periods are set in `properties.ini` file.

2.13.3 Output: None (the script is supposed to by run by cron)

3 DB Tables

3.1 `LastLogin`

Field	Type

e_oid	int(12)
user_ID	int(12)
time	int(16)
ipAddress	varchar(15)

3.2 *MaskedCookieData*

Field	Type
e_oid	int(12)
user_ID	int(12)
masked_ID	varchar(21)

3.3 *TempInvitation*

Field	Type
e_oid	int(12)
registrationCode	varchar(10)
eMailAddress	varchar(64)
creationTime	int(16)

3.4 *TempVerification*

Field	Type
e_oid	int(12)
user_ID	int(12)
verificationCode	varchar(21)
creationTime	int(16)

3.5 *User*

Field	Type
e_oid	int(12)
userName	varchar(64)
password	varchar(32)
eMailAddress	varchar(64)
gender	varchar(1)
birthYear	int(16)
memberSince	int(16)
isVerified	tinyint(1)
remainingInvitations	int(12)