Majority of web sites use sign-up with social networks as alternate for traditional registration/login. Buttons &#65533;Login with: Facebook, Twitter, Google,... &#65533; are present together with a login form very often now. Using social networks for login simplifies a user experience and saves his time. The author used social login on different web sites as well. Based on experience it was decided to build a PHP class/package that summarise the social login process on an abstract level and allows to implement the process with minimum coding. <h1>Social login in common</h1> For a user a social login looks like identical process in different social networks. Usually, it looks like: a user licks on a &#65533;login with ...&#65533; button. Is redirected to a social network website (http://facebook.com etc.) , clicks &#65533;Allow&#65533; on that site and, finally, is redirected back to an original website where his social login is processes and a session for a user is created. However, on a back-end there is some difference. Some social networks APIs don't need to connect to an api endpoint before to redirect a user a login page. Other has to connect to an API to init a login process, to get some special redirect url, to check some settings etc. Next difference is that some APIs will require saving a specific data (request token etc) between 2 steps of an auth process &#65533; get a redirect url and complete login. For others no need to save something. All such differences must be taken into account during a social login process. <h1 class="western" style="margin-bottom: 0in; line-height: 100%;">Introduction gelembjuk/auth PHP package to unify the social login process.</h1> The package is available in Composer. It allows to login with Facebook, Twitter, Google, LinkedIn with unified process and minimum coding. Additionally, the package can be easy extended with new social networks and same process will be used. Installation It is just normal installation with Composer {codecitation style="brush:bash;"} composer require gelembjuk/auth {/codecitation} &#65533; &#65533; Configuration

To use the package you need to get API settings for each social network you want to use. A config array can look like this:
{codecitation style="brush:php;"}
$integrations = array(
 'facebook' => array(
 'api_key' => 'fake facebook api key',
 'secret_key' => 'fake facebook secret key'
 ),
 'twitter' => array(
 'consumer_key' => 'fake twitter consumer key',
 'consumer_secret' => 'fake twitter consumer secret'
 ),
 'linkedin' => array(
 'api_key' => 'fake linkedin api key',
 'api_secret' => 'fake linkedin api secret'
 ),
 'google' => array(
 'application_name' => 'Your application name',
 'client_id' => 'fake google api client id',
 'client_secret' => 'fake google api client secret'
)
 );
{/codecitation}
The readme file on Github (https://github.com/Gelembjuk/auth) describes where to get API keys for each integration. You need to create new application for each API and copy your keys.

Basic usage

Create an object {codecitation style="brush:php;"} $socialnetwork = $_REQUEST['network'];

// here you could check if $socialnetwork has correct allowed value.
// If no check and it is wrong then you will get an exception

// create social network login object.
// The second argument is array of API settings for a social network
$network = GelembjukAuthAuthFactory::getSocialLoginObject($socialnetwork,$integrations[$socialnetwork]); &#65533; {/codecitation}

Start the login processes

Prepare a redirect url. In some APIs this url must be also registered in an application settings. It must be url to your social

login complete script.
{codecitation style="brush:php;"} $redirecturl = 'http://'.$_SERVER['HTTP_HOST']
.dirname($_SERVER['REQUEST_URI']).'/completelogin.php'; &#65533;
{/codecitation}

Get social login auth page url. You will need to redirect a user to this page. {codecitation style="brush:php;"}

$url = $network->getLoginStartUrl($redirecturl); {/codecitation}

Before redirect remember social login object state somewhere (usually in a $_SESSION) to restore it later.

{codecitation style="brush:php;"}
$_SESSION['socialloginsate_'.$socialnetwork] = $network->serialize();
{/codecitation}
And remember what network was used. You can skip this line if current network is part of your $redirecturl or if you use only one network.
In other words, on social login complete you need to know what network was used to create that object again {codecitation style="brush:php;"} $_SESSION['socialloginnetwork'] = $socialnetwork; {/codecitation} And redirect a user to a social netork login page {codecitation style="brush:php;"} header("Location: $url",true,301);
exit; {/codecitation} Complete a social login process

On a social network side a user will have to login to his account and then accept your request to grant permissions on basic access to his profile. If user cancels or disallow this then, anyway, he will be forwarded to your &#65533;login complete&#65533; page, same as in case of success.

On a login page we need to create same object as on start (it can be same script actually, depends on your implementation). {codecitation style="brush:php;"} // get a network name from previous page
$socialnetwork = $_SESSION['socialloginnetwork'];

// create object, same as above
$network = GelembjukAuthAuthFactory::getSocialLoginObject(
 $socialnetwork,$integrations[$socialnetwork]); {/codecitation} Next step is to read an input arguments pasted from a social network. {codecitation style="brush:php;"} $arguments = array();

 foreach ($network->getFinalExtraInputs() as $key) {
 $arguments[$key] = $_REQUEST[$key];
 } {/codecitation} This is required because classes from the package don't have access to $_GET,$_REQUEST etc arrays directly (because of attempt to create a nice arhitecture).

Restore a state of the object to be same as before redirect
{codecitation style="brush:php;"}
$network->unSerialize($_SESSION['socialloginsate_'.$socialnetwork]);
{/codecitation} Complete login and get a user profile from a social network
{codecitation style="brush:php;"} $profile = $network->completeLogin($arguments); {/codecitation} The profile is an array with keys `userid`,`name`,`email`,`imageurl`.

This is done! Now you can remember the profile in a session to know that a user is already loged in your web site.
{codecitation style="brush:php;"} $_SESSION['user'] = $profile;
{/codecitation}

On practice, you rather will find DB record for such user by $profile['userid'] + $socialnetwork in your users table and remember user's internal ID in a session. If no user in the DB then he did his login for first time and you have to add him to the table.