Secure Software Oauth2

Framework and libraries used

Composer

Useful to manage dependencies and package of a web application, I used it to install every library I’ve used in this project.

Slim

Slim is a very useful routing framework allowing to build easily REST api or just route the request of a website. In this project I use two instances of Slim routing app, one to handle API requests and the other to manage routes on my website.

Twig

As for the view rendering, I chose to use Twig which lets me define templates view to fill with datas. The view are stored in a directory in the public folder of each website.

Pure

CSS framework, easy to use to give a plus to the look, not really much to say about it.

SessionManager

To improve security of session management, I use a special class to create, delete and get session. You can find the description by following this link :

http://blog.teamtreehouse.com/how-to-create-bulletproof-sessions

Oauth 2.0 Server PHP

A library wrapping all the logic of Oauth inside different classes, it makes the implementation easier and cleaner. I implemented it on my authorization server and my API.

Architecture

API (Resources Server)

The API manages the access to the database via some classes to access specific datas, these classes are called “Model” there are three of them :

- UserModel : to get user data, register user and make authetication

- ToDoListModel : to get the ToDoLists, add one, delete one.
- TaskModel : to get tasks of a list, add one, delete one.

Classes called controller are used to manage the different API calls via slim routing, get the data by using the Model and sending the JSON encoded datas response. In addition a controller is available to manage user login and registration to the API (the API handle its session with the session manager described in the library list).

The API calls are using a specific url to be called : todolistapi.io/api/NameOfTheAction

For example a call to get the lists of the todoList of the users : todolistapi.io/api/ToDoList

We’ll take later about the access_token part.

Authorization server

The authorization server is implemented in the same application as the API, just following a different url : todolistapi.io/Oauth/

Therefore it’s using the same slim router to handle the request, there are three routes :

- Authorize using GET method, to display the authorization form (if the user is connected to the API website, it’s just a yes/no form, but if the user isn’t connected, it shows a login form to let the user connects).
- Authorize using POST method, which will handle the result of the preceding form and starting Oauth process.
- Token using POST method, used to handle the token exchange and refresh token exchange.

Website (Client)

This one is another app which uses Oauth as a client to get access to API informations. It uses also a slim router to handle the request and twig templating system to render the data from the API in a view.

It also has its own session manager (same as described in the libraries section), the session is used to store informations about the connected user and the token informations.

To use it, it’s necessary to authorize this application with your your API account to access your data. To do so, there is a button on the main page letting you chose to connect with your API account, redirecting you to the authorization server form page. Once the user authorized the website to get the data, the authorization server will send a code to the redirect uri on the website (redirect uri is in this case securesoftware.io/code). This page can then send the code to the authorization server to get a token and a refresh token.

The token is then used to make the API calls, a check of the token expiration is made at every call, if the token has expired, a request for a new token using the refresh token is made.

Databases

There are two MySQL databases both used by the API and authorization server, one contains all API data when the other is used for Oauth to hold the client list, tokens, refresh tokens, ..etc

Stored procedures are used to realize operations on the API data (addition, suppression,..).

Additional classes

- TokenManager : class to generate unique token, mainly used for CSRF tokens, it has a method to generate a token and store it Session variable and another to check if the token given as argument is the name as the one in the session.
- ApiCaller : wrapper of curl to realize API calls using the access token automatically if this one is present in the Session variable
- RefreshToken : check if the access token has expired, if yes makes an API call to the authorization server to get a new one using the refresh token stored in the Session variable.
- SessionManager: manage session creation and destruction, as well as checking for possible session hijack.

Complete connection flow

1. User go to “http://securesoftware.io” which show him a button to ask to connect with his API account.
2. By pressing this button, user is redirected to the authorization server, showing him a form to ask him if he lets the website accessing his API data. (http://todolistapi/Oauth/Authorize?response_type=code&client_id=todolist&state=Nonce&redirect_uri=http://securesoftware.io/code)
3. When he accepts it makes a POST request to the same url as he was.
4. Then he’s redirected with a code by the authorization server to the website page used to handle this code. (http://todolistapi/Oauth/code?code=HisCode&state=Nonce)
5. The website makes a curl request to the Authorization server using his id, his secret and the code to ask for a token. (http://todolistapi/Oauth/Token?client_id=todolist&client_secret=todolist&state=Nonce&code=HisCode&grant_type=authorization_code)
6. Authorization server checks the code and the nonce (check if the nonce is the same as it was sent in the second step) and the client id and secret. If everything is good, he creates an access token and a refresh token and send them back to the website.
7. The website save the token, its expiration date and the refresh token in a session variable and makes and API call to get user info to put them in the session too. User is then redirected to the connected section of the website.

8. Authorization complete, user now use the website to make API calls to get his API datas. He’s considered as login on the website using his API datas.

API call

1. The user wants to add a new To Do List, so he enters the name of the list and click “Add”.
2. Website slim router handles the post request, check the CSRF token validity, check if the user is connected, and then makes a curl API calls to the add url (todolistapi.io/api/ToDoList/add) with the post parameters (name of the list) and the access_token as a url parameter.
3. API slim router handles the post request, first check the token validity, get the token informations (the user_id associated with the token) then call the ToDoListModel method to add a list with the user ID and list name as parameter
4. ToDoListModel uses a prepared statement to call a stored procedure to add the list and save the result of the operation (success (1) or failed (0)).
5. The result is encoded in JSON and sent as a response of the API.
6. The website decode the response of the API and send the result to with a template name to render a view using these datas.
7. The website refreshes the list of To Do List.