Login-Authorization

Generated by Doxygen 1.7.6.1

Wed Jul 2 2014 13:15:10

Contents

Chapter 1

Getting Started

1.1 Installation

To get started right away, read the chapter on Installation .

1.2 Oauth 2.0

If you want to learn the basics of OAuth 2.0 consider reading the Overview of OAuth 2.0
.

1.3 API Documentation

To see a full API documentation have a look at API Documenation .

Chapter 2

API Documenation

2.1 API

All requests need to be made to api/foo, for example:
`http://example.com/Login-Authorization/api/request-token`

2.1.1 Authentication

2.1.1.1 request-token

Redirect here with these query parametes:

- client_id
- scope
- redirect_uri (optional if only one is registered)
- user_id

A approval dialog will open, if the user grants authorization
the user will be redirected to redirect_uri with these query parameters:

- code
- state

2.1.1.2 access-token

Send a POST request with these parametes to receive an access token:

- client_id
- client_secret
- redirect_uri (same as in request-token or the only registered one)
- code (from request-token)

The response will be a JSON:

```
{"access_token": string,
 "expires_in": int,
 "token_type": string,
 "scope": string}
```

2.1.2 Resource

2.1.2.1 validate

Send a POST request with these parametes to check if a token is still valid:

- access_token

The response will be a JSON:

```
{"success": bool}
```

2.1.2.2 get-login-data

Send a POST request with these parametes to get the data of the user associated with the token:

- access_token

The response will be a JSON:

```
{"user_name": string,
 "user_id": int,
 "foaf_uri": string}
```

Chapter 3

Installation

1. Create a mysql-database with any name you like
2. Copy config.php-dist to config.php and configure it according to your mysql data
3. Create all required tables, clients and the first admin by running _install.php (in the _installation folder) from your browser
4. If everything worked, you should be able to login with:
    - Username: admin
    - Password: admin

Chapter 4

Overview of OAuth 2.0

4.1 Introduction

OAuth 2.0 is an open authorization protocol which enables applications to access each others data. For example, a game application can access a users data in the Facebook application etc. The following sections should give a basic overview of the OAuth 2.0 protocol.

4.2 Roles

OAuth 2.0 defines four basic roles:

- The resource owner (or user):

  The person (or application) that owns the data that is to be shared

- The resource server:

  The server responsible for storing and managing the resources

- The client applictation (or just client):

  The application that requests access to the rescources stored on the resource server

- The authorization server:

  The server responsible for authorizing the client to access the protected resources, can be the same as the resource server

Their interaction is illustrated in the following diagram:

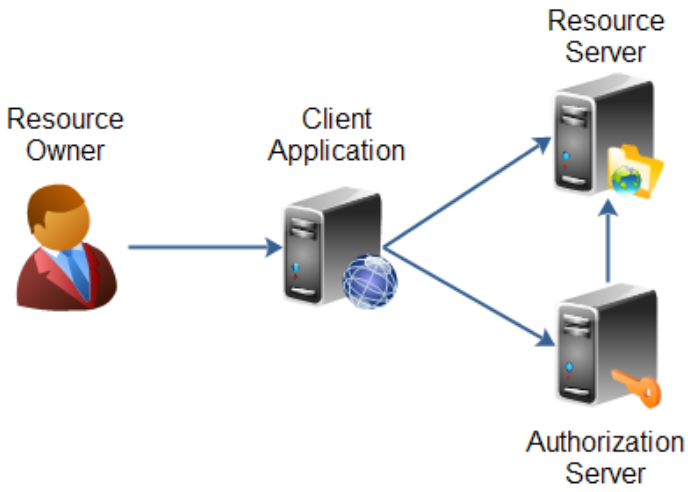

Figure 4.1: Basic roles defined by OAuth 2.0

4.3 Abstract Protocol Flow

The basic flow of OAuth 2.0 (as shown in the diagram below) is as follows:

To get access to protected resources a client sends an authorization request to the resource owner(user). If the user grants authorization, the client gets an authorization grant, this grant can take different forms (see Authorization Grant)

The client exchanges the grant for an access token at the authorization server.

The access token can then be used by the client to access the protected resources.

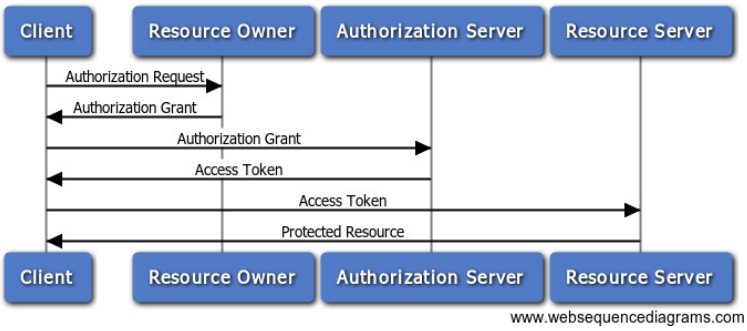

Figure 4.2: Abstract protocol flow

4.4 Authorization

4.4.1 Registration

Before the client can request access to protected resources, the client needs to register with the authorization server.

During that process the client is assigned a unique client ID and a client secret.

The client also registers a redirect URI, which is where the user is redirected to, after granting/denying authorization.

4.4.2 Authorization Grant

OAuth 2.0 specifies the following four types of authorization grants:

4.4.2.1 Authorization Code

Since this is the grant implemented in this application a more detailed explanation is required.

The basic idea of this grant is as follows:

- The user accesses the client application (1)
- The client app asks the user to log in via an authorization server (2)
- The user is redirected to the authorization server by the client, the client also sends its client ID along (3)
- After successful login via authorization server the user is asked if he/she wants to grant authorize the client to access the user data.

  The user is redirected to the registered redirect URI of the client along with an authorization code (4, 5)
- At the redirect URI the client connects directly to the authorization server and sends its client ID, client secret and the authorization code.

  If all of the data is valid, the authorization server sends back an access token (6, 7)
- This token is sent with every recource request and serves as authentication of the client and authorization to access the recources (10-13)

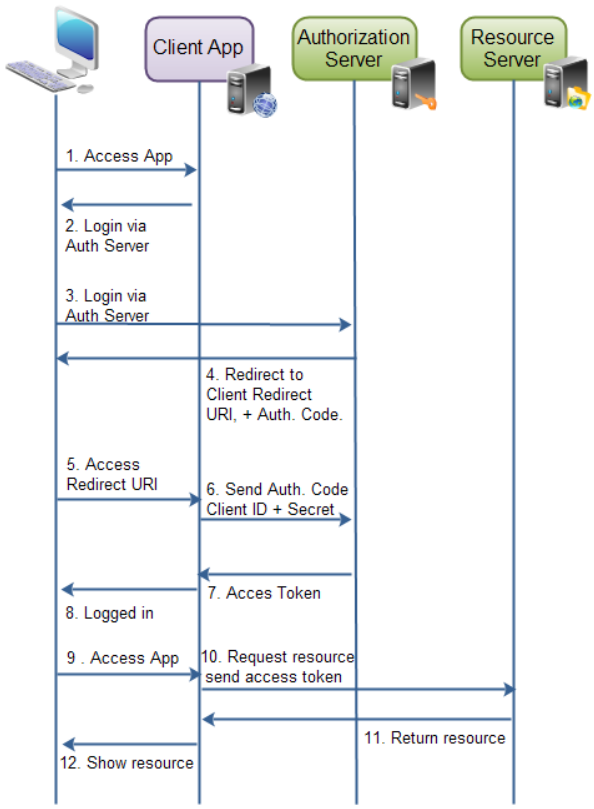

Figure 4.3: authorization code grant flow

4.4.2.2 Implicit

The implicit grant is similar to authorization code grant, the difference being that the authorization server sends an access token

to the client immediatly after the client logged in and authorized the client.

4.4.2.3 Resource Owner Password Credentials

This grant requires the user to give the own credentials to the client application, so that the client can use them to access the resources.

4.4.2.4 Client Credentials

The authorization server exchanges an access token for client ID and client secret directly

4.4.3 Endpoint

An endpoint is usually a URI on a server, in the case of OAuth 2.0 these are:

- The authorization endpoint:

  The endpoint where the user logs in and grants/denies authorization to the client

- The redirect endpoint:

  The endpoint where the user is redirected to after granting/denying authorization

- The token endpoint:

  The endpoint where the client exchanges client ID, client secret and authorization code for an access token

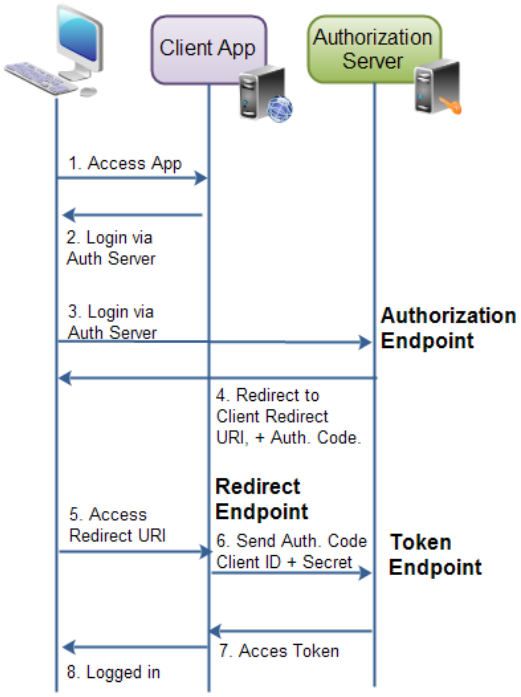

Figure 4.4: Endpoints defined by OAuth 2.0

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

Data Structure Documentation

6.1 Database Class Reference

Contains functions to simplify database requests.

Public Member Functions

- __construct ()
- getUserDataName ($user_name)
- getUserDataId ($user_id)
- getLoginData ($user_name)
- userExists ($user_name)
- addUser ($user_name, $user_password_hash, $foaf_uri=null)
- getUserId ($user_name)
- getUserName ($user_id)
- getUserType ($user_name)
- changeEntry ($user_id, $key, $value)
- getScope ($user_name)

Private Attributes

- **$db_connection** = null

6.1.1 Detailed Description

Contains functions to simplify database requests.

Author

Sebastian Ankerhold

6.1.2 Constructor & Destructor Documentation

6.1.2.1 __construct ( )

Constructor creates/checks database connection

6.1.3 Member Function Documentation

6.1.3.1 addUser ($ *user_name,* $ *user_password_hash,* $ *foaf_uri* = `null`)

Adds a new user and password to the database

Parameters

string	*$user_name*	The name of a user
string	*$user_password_hash*	The password for that user

Returns

true if adding was successful, else false

6.1.3.2 changeEntry ($ *user_id,* $ *key,* $ *value*)

Changes an entry of a given user to a new value

Parameters

int	*$user_name*	The name of a user
string	*$key*	The column that needs changing
string	*$value*	The new value of that column

Returns

true if change was successful, false else

6.1.3.3 getLoginData ($ *user_name*)

Returns data necessary for a login

Parameters

string	*$user_name*	The name of a user

Returns

login relevant data

6.1.3.4 getScope ($ *user_name*)

Checks the type a given user and returns string of scopes

Parameters

string	*$user_name*	

Returns

a string which contains all the scopes the user was permitted

6.1.3.5 getUserDataId ($ *user_id*)

Returns all available data of the corresponding user id

Parameters

int	*$user_id*	The ID of a user

Returns

all available user data

6.1.3.6 getUserDataName ($ *user_name*)

Returns all available data of the corresponding user name

Parameters

string	*$user_name*	The name of a user

Returns

all available user data

6.1.3.7 getUserId ($ *user_name*)

Returns the user id of a given user name

Parameters

string	*$user_name*	The name of a user

Returns

The ID of that user

6.1.3.8 getUserName ($ *user_id*)

Returns the user name of a given user id

Parameters

int	*$user_id*	The Id of a user

Returns

The name of that user

6.1.3.9 getUserType ($ *user_name*)

Returns the user type of a given user name

Parameters

string	*$user_name*	The name of a user

Returns

The type of that user

6.1.3.10 userExists ($ *user_name*)

Checks if a user with a given name is already registered

Parameters

string	*$user_name*	The name of a user

Returns

true if user name already exists, otherwise false

The documentation for this class was generated from the following file:

- Database.php

6.2 OAuthLogin Class Reference

Handles user's login and logout process.

Public Member Functions

- __construct ($user_name=null)
- doLogout ()
- isUserLoggedIn ()

Data Fields

- **$errors** = array()
- **$messages** = array()

Private Member Functions

- checkLoginOptions ($user_name=null)
- dologinWithPostData ()
- doLoginWithRegistrationData ($user_name)
- getAuthCode ()
- getAccessToken ()

Private Attributes

- **$database** = null

6.2.1 Detailed Description

Handles user's login and logout process.

Author

Sebastian Ankerhold

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __construct ($ *user_name* = `null`)

Constructor Checks and performs the possible login actions

6.2.3 Member Function Documentation

6.2.3.1 checkLoginOptions ($ *user_name* = null) [private]

Checks the GET and POST data and performs the corresponding actions.

Parameters

string	*$user_name*	The name of a user

6.2.3.2 dologinWithPostData ( ) [private]

Log in with post data by checking the validity of posted data and store essential data into PHP SESSION

6.2.3.3 doLoginWithRegistrationData ($ *user_name*) [private]

If a user registered a new account, this function uses the entered data to log in the user

Parameters

6.2.3.4 doLogout ( )

Performs the logout by resetting the $_SESSION array and displays a logout message

6.2.3.5 getAccessToken ( ) [private]

Request an access token by sending the received auth.code to the token endpoint. If successful the token will be stored in SESSION

6.2.3.6 getAuthCode ( ) [private]

Redirects to the authorization endpoint to request an authorization code If successful the user will be redirected back to this location with a code

6.2.3.7 isUserLoggedIn ( )

Simply return the current state of the user's login

Returns

boolean user's login status

The documentation for this class was generated from the following file:

- OAuthLogin.php

6.3 Registration Class Reference

Handles the user registration.

Public Member Functions

- __construct ()

Data Fields

- **$errors** = array()
- **$messages** = array()

Private Member Functions

- registerNewUser ()

Private Attributes

- **$database** = null

6.3.1 Detailed Description

Handles the user registration.

Author

Sebastian Ankerhold

6.3.2 Constructor & Destructor Documentation

6.3.2.1 __construct ( )

Constructor

6.3.3 Member Function Documentation

6.3.3.1 registerNewUser ( ) [private]

Handles the entire registration process. Checks all error possibilities ,creates a new user in the database and performs a login for the new user if everything is fine.

The documentation for this class was generated from the following file:

- Registration.php