Smart OAuth2 Server

PHP Library for OAuth2

Lightweight simple and flexible OAuth2 Server

With support of the JSON Web Token

Customizable user authentication and storage of the token records

Designed based on IoC

Key features

1. Support of the JSON WEB Token.
2. Support of the signing algorithms - HS256, HS384, HS512, RS256, RS384, RS512.
3. Support of the refresh tokens.
4. Customizable user authentication and storage of the token records. It can be implemented for a classical database, nosql database, a memcache solution like Redis etc.
5. Support of the immediate invalidation of the access token and refresh token in the case of theft.

Directory Structure

docs

src

  OAuth2

    Interfaces

Directory Structure (part 2)

docs

This directory contains the documentation about classes, interfaces and functions of the *SmartFactory OAuth2 Server*.

src

This is the root directory for all classes and interfaces. The class loader is implemented based on PSR4 approach. You have no need to add additional class loader function for your classes.

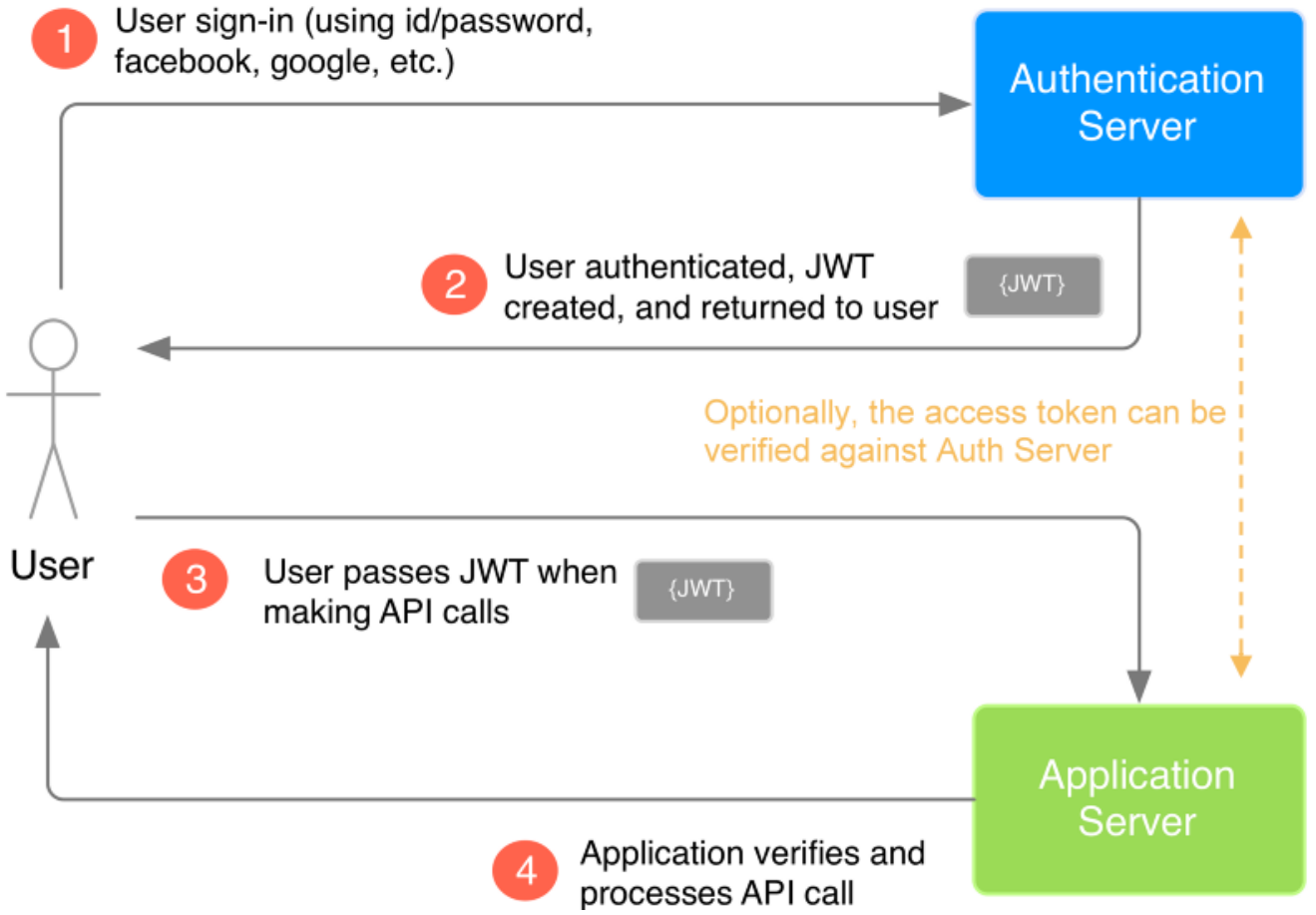
src/OAuth2

This directory contains the core classes and interfaces of the *SmartFactory OAuth2 Server*.

src/OAuth2/Interfaces

This directory contains the core interfaces of the *SmartFactory OAuth2 Server*.

Workflow and principle of work



Workflow and principle of work (Details)

1. The user sends his credentials to the authentication server.
2. The authentication server verifies the credentials, issues the access and refresh tokens and sets the expiration time for them. The access token is wrapped into JWT token and signed with a private key, so that nobody can manipulate it.
3. The client receives the response from the authentication server and stores the token information locally. The client should take care of the expiration and renew the tokens on proper time. As long as the refresh token is valid, the access token can be refreshed without using the credentials, so that the credentials need not be sent over the network. And only when the refresh token is about to expire, it has to be renewed by sending the credentials.
4. The client needs not to decode the JWT access token, but should just store it locally and send it to the application server in each API request. The application server has the public key, with which it verifies the signature of the JWT access token. Optionally, the application server can extract the access token and verify it against the authentication server, whether it is still valid.

OAuthManager

This is the core class offering all necessary methods.

Authentication with credentials and getting the tokens:

```
$oauth = singleton(IOAuthManager::class);
$oauth->authenticateUser($credentials, &$response);
```

Refreshing the access and refresh tokens:

```
$oauth->refreshTokens($refresh_token, $user_id, $client_id, &$response);
```

Verification of the access token:

```
$oauth->verifyJwtAccessToken($jwt_access_token, $check_on_server);
```

Invalidation of the token records:

```
$oauth->invalidateUser($user_id, $client_id, $refresh_token);
$oauth->invalidateClient($user_id, $client_id, $refresh_token);
$oauth->invalidateJwtAccessToken($jwt_access_token);
$oauth->invalidateRefreshToken($refresh_token);
```

IUserAuthenticator

You should implement this interface and specify the authenticator by the initialization of the *OAuthManager*.

```
IUserAuthenticator::authenticateUser($credentials);
```

This method should e.g. access the user database and return the user id in the case of the successful authentication, otherwise it should throw an exception with an error message.

ITokenStorage

You should implement this interface and specify the authenticator by the initialization of the *OAuthManager*. It can be implemented for a classical database, nosql database, a memcache solution like Redis etc.

The following methods should be implemented:

Save the token information:

```
public function saveTokenRecord(&$token_record);
```

Verify the access token (record must exist and not expired):

```
public function verifyAccessToken($access_token, $user_id, $client_id);
```

Verify the refresh token (record must exist and not expired):

```
public function verifyRefreshToken($refresh_token, $user_id, $client_id);
```

Delete the record (by user_id, client_id, access_token, refresh_token):

```
public function deleteTokenRecordByKey($key, $value);
```

Next steps

One of the goals of this presentation was to give you an overview over the possibilities and functions of the *SmartFactory OAuth2 Server*.

To get familiar with the *SmartFactory Core and OAuth2* Server do the following:

- View and study the API documentation in the folder docs or here API documentation.
- Study the core code of the library *SmartFactory* and that of the *SmartFactory OAuth2 Server*.
- Study the Demo Application that shows usage of the *SmartFactory* and *SmartFactory OAuth2 Server*.

How to start

1) Git-clone the demo application *SmartFactoryDemo* and run 'composer update'.
2) Study the directory structure of the demo application and the code.
3) Implement the interfaces *IUserAuthenticator* and *IUserAuthenticator*.
4) Bind you classes to the interfaces in the file *intialization_inc.php* to be able to use the IoC approach for creating objects offered by the library *SmartFactory*.
5) Implement you business logic in the root directory or any subdirectory.
6) Implement the JSON API request handlers.
7) Add translation texts for your application over the *localization/edit.php* or directly into the JSON file *localization/texts.json*. Use the script *localization/check.php* to check your localization texts for missing translations.