EULogin-Bundle Documentation

Release 1.0.0

Pol Dellaiera

Jan 14, 2020

Contents

A Central Authentication Service bundle for Symfony 4.

The Central Authentication Service (CAS) is an Open-Source single sign-on protocol for the web. Its purpose is to permit a user to access multiple applications while providing their credentials only once. It also allows web applications to authenticate users without gaining access to a user's security credentials, such as a password. The name CAS also refers to a software package that implements this protocol.

In order to foster a greater adoption of this bundle, it has been built with interoperability in mind. It only uses PHP Standards Recommendations interfaces.

- PSR-3 for logging,
- PSR-4 for classes autoloading,
- PSR-6 for caching,
- PSR-7 for HTTP messages (requests, responses),
- PSR-12 for coding standards,
- PSR-17 for HTTP messages factories,
- PSR-18 for HTTP client.

CHAPTER 1

Requirements

1.1 PHP

PHP greater or equal to 7.1.3 for Symfony 4.

PHP greater or equal to 7.2.5 for Symfony 5.

1.2 Symfony

The minimal required version of Symfony is 4.

CHAPTER 2

Installation

This package does not yet have a Symfony Flex recipe. Installation steps must be done manually.

Default configuration files will be copied in the *dev* environment except for the file defining the services.

2.1 Step 1

The easiest way to install it is through Composer

```
composer require drupol/eulogin-bundle:^4.4
```

2.2 Step 2

Make sure that the bundle is enabled in *config/bundles.php*.

You should see a line that looks like the following:

```
drupol\\EuloginBundle\\EuloginBundle::class => ['all' => true],
```

2.3 Step 3

As this package depends on the package *drupol/cas-bundle*, you will need to copy some configuration files from that package first.

```
cp -ar vendor/drupol/cas-bundle/Resources/config/* config/
```

Then, copy the configuration files from the bundle *drupol/eulogin-bundle* in your application

```
cp -ar vendor/drupol/eulogin-bundle/Resources/config/* config/
```

Warning: It is important to play those commands in the proper order.

2.4 Step 4

Edit the configuration file *config/packages/dev/cas_bundle.yaml* and make the necessary changes to fit your needs.

See more on the dedicated *Configuration* page.

CHAPTER 3

Configuration

```
cas:
  base_url: https://webgate.ec.europa.eu/cas
  protocol:
    login:
      path: /login
      allowed_parameters:
        - service
        - renew
        - gateway
      default_parameters:
        service: cas_bundle_homepage
    serviceValidate:
      allowed_parameters:
        - service
        - ticket
        - pgtUrl
        - renew
        - format
        - userDetails
        - ticketTypes
      path: /serviceValidate
      default_parameters:
        userDetails: "true"
        format: XML
        #pgtUrl: cas_bundle_proxy_callback
    logout:
      path: /logout
      allowed_parameters:
        - service
      default_parameters:
        service: cas_bundle_homepage
    proxy:
      path: /proxy
      allowed_parameters:
```

(continues on next page)

(continued from previous page)

```
        - targetService
        - pgt
    proxyValidate:
      path: /proxyValidate
      allowed_parameters:
        - service
        - ticket
        - userDetails
        - pgtUrl
        - format
      default_parameters:
        userDetails: "true"
        format: XML
        #pgtUrl: cas_bundle_proxy_callback
```

CHAPTER 4

Usage

CHAPTER 5

Tests, code quality and code style

Every time changes are introduced into the library, Travis CI and Github Actions run the tests written with PHPSpec.

PHPInfection is also triggered used to ensure that your code is properly tested.

The code style is based on PSR-12 plus a set of custom rules. Find more about the code style in use in the package drupol/php-conventions.

A PHP quality tool, Grumphp, is used to orchestrate all these tasks at each commit on the local machine, but also on the continuous integration tools (Travis, Github actions)

To run the whole tests tasks locally, do

```
composer grumphp
```

or

```
./vendor/bin/grumphp run
```

Here's an example of output that shows all the tasks that are setup in Grumphp and that will check your code

```
$ ./vendor/bin/grumphp run
GrumPHP is sniffing your code!
Running task  1/10: Composer... ✓
Running task  2/10: ComposerNormalize... ✓
Running task  3/10: YamlLint... ✓
Running task  4/10: JsonLint... ✓
Running task  5/10: PhpLint... ✓
Running task  6/10: TwigCs... ✓
Running task  7/10: PhpCsAutoFixerV2... ✓
Running task  8/10: PhpCsFixerV2... ✓
Running task  9/10: Phpcs... ✓
Running task 10/10: PhpStan... ✓
$
```

CHAPTER 6

Contributing

See the file CONTRIBUTING.md but feel free to contribute to this library by sending Github pull requests.

CHAPTER 7

Development

In order to test efficiently, is to test the library against a real CAS server.

If you're not able to use one, the best is to work with a local CAS server.

If you want to setup your own local CAS server in less than 2 minutes, use the repo crpeck/cas-overlay-docker and you'll have something working really quickly.

Don't forget to setup the HTTPS certificates because the communication between the CAS server and your application MUST be in HTTPS, and I haven't found a way yet to disable this for testing purposes.

If you prefer to use your local machine, there are already some documentation on Github.