ECPHP-CAS-Bundle documentation

Release 1.0.0

Jul 19, 2022

Contents

A Central Authentication Service bundle for Symfony 4 & 5.

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 the HTTP client.

CHAPTER 1

Requirements

1.1 PHP

PHP greater or equal to 7.2.5.

1.2 Symfony

The minimal required version of Symfony is 5.

1.3 Extensions

These PHP extensions are required:

- json
- libxml
- simplexml

1.4 Packages

In order to get the CAS bundle working, you will require some dependencies.

To give a maximum freedom to the users using, each required dependencies is a well defined standardized PHP class.

See the PHP-FIG framework group for more information.

Dependency	PSR	Implementations	Example package
Logger	PSR-3	log-implementation	monolog/monolog
Cache	PSR-6	cache-implementation	symfony/cache
HTTP factories	PSR-17	http-factory-implementations	nyholm/psr7
HTTP Client	PSR-18	http-client-implementations	symfony/http-client

You are free to use any package you want, as long as they are implementing the proper requirement.

CHAPTER 2

Installation

This package has a Symfony Flex recipe that will install configuration files for you.

Default configuration files will be copied in the *dev* environment.

2.1 Step 1

The recommended way to install it is with Composer :

```
composer require ecphp/cas-bundle
```

Warning: If you use API Platform and Symfony < 5.2, then it's possible that some URLs contains parameters with a dot inside. By default, Symfony mangles url parameters having dot with an underscore, which can lead in huge inconsistencies if you heavily rely on query parameters like in API Platform.

In order to fix that issue, the optional package loophp/unaltered-psr-http-message-bridge-bundle can be installed.

```
composer require loophp/unaltered-psr-http-message-bridge-bundle
```

2.2 Step 2

This is the crucial part of your application's security.

Edit the security settings of your application, usually in *config/packages/security.yaml*.

```
security:
    firewalls:
        main:
            anonymous: ~
            guard:
```

(continues on next page)

(continued from previous page)

```
                provider: cas
                authenticators:
                    - cas.guardauthenticator

    access_control:
        - { path: ^/api, role: ROLE_CAS_AUTHENTICATED }
        - { path: ^/admin, role: ROLE_CAS_AUTHENTICATED }
```

If you're using Symfony >= 5.4, it is already possible to use the new security system (more information). This is the default in Symfony >= 6.

```
security:
    enable_authenticator_manager: true
    firewalls:
        main:
            custom_authenticator: cas.authenticator

    access_control:
        - { path: ^/api, role: ROLE_CAS_AUTHENTICATED }
        - { path: ^/admin, role: ROLE_CAS_AUTHENTICATED }
```

This configuration example will trigger the authentication on paths starting with */api* or */admin*, therefore make sure that at least such paths exists.

Feel free to change these configuration to fits your need. Have a look at the Symfony documentation about security and Guard authentication.

2.3 Step 3

The CAS protocol requires HTTPS on both side (client and server) in order to communicate.

Whilst it is not possible to configure the behavior of the CAS server, it is possible to configure the HTTP client in use in this bundle in order to relax the requirement and to disable SSL checks when communicating from the client to the server.

> **Warning:** Keep in mind that the following is only for development setup, not for production.

On step 3, while copying the configuration files, the file `config/packages/dev/cas_framework.yaml` is copied over. That file is useful when developing, it will disable some verifications required when using SSL protocol.

Those particular settings are specific to the default HTTP client that is installed, which is symfony/http-client.

The `User-Agent` HTTP header used on the `dev` environment is `SymfonyCasBundle`. Feel free to customize it or remove it when switching to another environment.

If you plan to change the HTTP client, those settings will most probably need to be updated accordingly.

2.4 Step 4

The default configuration of this bundle comes with a configuration for authenticating with a real CAS server setup for testing and demo purposes at https://casserver.herokuapp.com/cas/.

Warning: It is important to note that this is the Apereo official public demo cas server, used by the project for basic showcases. They may go up and down as the project needs without notice, see this page for further information.

The credentials to use for authentication are the following:

- User: `casuser`
- Password: `Mellon`

Modifying the configuration file is key in this bundle and requires some understanding of the CAS protocol. See more on the dedicated *Configuration* page for that.

The aforementioned server provided by Apereo does not support Proxy authentication.

If you need a server with Proxy authentication, edit the `cas_bundle.yaml` and replace `https://casserver.herokuapp.com/cas/` with `https://heroku-cas-server.herokuapp.com/cas/`. Make sure to enable the property `pgtUrl` which is by default in comment. The source of that server are hosted on Github.

If you prefer using a local CAS server, you can choose to build your own using the tool you prefer. The quickest solution for a working CAS server on any platform is this Docker project.

CHAPTER 3

Configuration

Hereunder an example of configuration for CAS Bundle.

Tip: Based on this configuration, the behavior of the bundle can change.

```
base_url: https://heroku-cas-server.herokuapp.com/cas
protocol:
  login:
    path: /login
    allowed_parameters:
      - service
      - renew
      - gateway
    default_parameters:
      service: https://my-app/homepage
  serviceValidate:
    path: /p3/serviceValidate
    allowed_parameters:
      - format
      - pgtUrl
      - service
      - ticket
    default_parameters:
      format: JSON
      pgtUrl: https://my-app/casProxyCallback
  logout:
    path: /logout
    allowed_parameters:
      - service
    default_parameters:
      service: https://my-app/homepage
  proxy:
    path: /proxy
```

(continues on next page)

(continued from previous page)

```
    allowed_parameters:
      - targetService
      - pgt
  proxyValidate:
    path: /proxyValidate
    allowed_parameters:
      - format
      - pgtUrl
      - service
      - ticket
    default_parameters:
      format: JSON
      pgtUrl: https://my-app/casProxyCallback
```

CHAPTER 4

Usage

4.1 Step 1

Follow the *Installation* procedure.

4.2 Step 2

Configure the configuration files accordingly and the security of your Symfony application.

4.3 Step 3

Once setup, if you try to reach a path that is protected by the firewall, you will be automatically redirected to the CAS server login page.

If you try to reach a path using Ajax (*X-Requested-With: XMLHttpRequest*), there won't be any redirection but an error **401** (*Unauthorized*).

Once authenticated on the CAS server, you will be redirected back to the Symfony application and continue the authentication process.

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  1/13: SecurityChecker... ✓
Running task  2/13: Composer... ✓
Running task  3/13: ComposerNormalize... ✓
Running task  4/13: YamlLint... ✓
Running task  5/13: JsonLint... ✓
Running task  6/13: PhpLint... ✓
Running task  7/13: TwigCs... ✓
Running task  8/13: PhpCsAutoFixerV2... ✓
Running task  9/13: PhpCsFixerV2... ✓
Running task 10/13: Phpcs... ✓
Running task 11/13: PhpStan... ✓
Running task 12/13: Phpspec... ✓
Running task 13/13: Infection... ✓
```

CHAPTER 6

Contributing

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

CHAPTER 7

Development

7.1 Maintainers

See the MAINTAINERS.txt file.

7.2 Documentation

Documentation can be built locally using Sphinx.

To render the documentation locally do the following steps:

- `docker-compose up`
- Navigate to http://127.0.0.1:8100/

7.3 Contributors

See the Github insights page.