Arcella Documentation

Release 0.0.1 pre-alpha

nplhse

November 22, 2016

Basics

The web is all about publishing and distributing content. Unfortunately most of the time you end up fighting against a content management system, rather than enjoying your content. The idea behind Arcella is to provide an easy to use environment that allows the user to focus on publishing and distributing their content.

CHAPTER 1

What is Arcella?

The web is all about publishing and distributing content. Unfortunately most of the time you end up fighting against a content management system, rather than enjoying your content. The idea behind Arcella is to provide an easy to use environment that allows the user to focus on publishing and distributing their content.

CHAPTER 2

Requirements

Arcella has been designed to have as few requriements as possible, which means that it can be used on most of the Web hosting providers. All you need is:

- Webserver (Apache, Nginx, LiteSpeed, IIS, etc.) with PHP 5.6 or higher
- MySQL version 5.6 or higher

CHAPTER 3

Installation

3.1 From GitHub

1. Launch a terminal or console and navigate to the webroot folder. Clone the Arcella repository from https://github.com/nplhse/Arcella to a folder in the webroot of your server, e.g. ~/webroot/arcella.

```
cd ~/webroot
git clone https://github.com/nplhse/Arcella.git
```

2. Install the vendor bundles by using Composer:

```
cd ~/webroot/arcella
composer install
```

3. Setup the database

```
php bin/console doctrine:database:create
php bin/console doctrine:schema:create
```

4. Install all the node bundles by using npm

```
npm install
```

5. And finally you might want to populate the database with some (fake) data, but this is an totally optional step

```
php bin/console doctrine:fixtures:load
```

CHAPTER 4

License

Arcella is published under the MIT License.

CHAPTER 5

Architecture

5.1 Hexagonal architecture

One of the main concepts behind the architecture of Arcella is Hexagonal Architecture, which defines several conceptual layers of code responsibility and helps to decouple code between those layers. It's main purpose is to ensure that our application expresses itself and uses the Symfony framework to accomplish tasks, instead of being our application itself.

That results in three differend layers which are: * *Domain layer* * *Application layer* * *Framework layer*

5.1.1 Domain layer

The Domain layer defines the core functionality of Arcella, but it only defines interfaces because it doesn't care about which implementations the Application layer uses.

5.1.2 Application layer

5.1.3 Framework layer

CHAPTER 6

Utilities

Adds several useful tools and utilities to Arcella applications.

6.1 TokenValidator

The TokenValidator is a service that allows you to add an extra level of confirmation to certain actions. For example you could use it to validate that a user actually has control of the email address he/ she provided within the registration process.

6.1.1 Configuration

Inside the main config you can define the following parameters for your Tokens:

- the lifetime (in seconds)
- the length (in characters)
- and the keyspace that will be used during the creation.

6.1.2 Usage

You should always use the TokenValidator as a service via the DI-container which can be accessed via **@arcella_utility_tokenvalidator**.

Creating a new Token

To create a new Token entity you just need to call the **generateToken()** function.

```
$token = $this->tokenValidator->generateToken();
```

The returned value is the key by which the Token entity could be identified.

Optional you could also provide additional parameters or a life time in seconds to be added to the Token entity like this

```
// Parameters need to be stored inside an associative array.
$params = array(
    'foo' => $bar,
);
```

```
// The lifespan is set in seconds as integer.
$lifespan = 60;

$token = $this->tokenValidator->generateToken($params, $lifespan);
```

Validating a Token

If you want to validate a Token you just need to call the **validateToken()** function.

```
$this->tokenValidator->validateToken($key);
```

This function will either return *true* if the provided key represents a valid Token or *false* if this is not the case. If there is no Token entity matching the key this function will throw a *EntityNotFound* Exception.

If the validation was successful and there are any additional parameters set they can be accessed via the TokenValidator.

```
$params = $this->tokenValidator->getTokenParams();
```

Removing a Token

After a successful validation you should remove the Token entity from the TokenRepository.

```
$this->tokenValidator->removeToken($key);
```

CHAPTER 7

How to contribute

Any contribution to Arcella is appreciated, whether it is related to fixing bugs, suggestions or improvements. Feel free to take your part in the development of Arcella!

However you should follow these simple guidelines for your contribution to be properly recived:

- Arcella uses the GitFlow branching model for the process from development to release.
- Because of GitFlow we accept contributions only via pull requests on Github.
- Please keep in mind, that Arcella tries to follow SemVer v2.0.0.
- In order to foster an inclusive, kind, harassment-free, and cooperative community, you should follow our Code of Conduct.
- Also you should make sure to follow the PHP Standards Recommendations and the Symfony best practices.

CHAPTER 8

Code of Conduct

8.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

8.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

8.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

8.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project lead at mail@nplhse.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident.

Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

8.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4

CHAPTER 9

Pull requests

To make a long story short you should at first fork and install Arcella from this repo. Now you make sure all the tests pass. Make your changes to the code and add tests for your changes. If all the tests pass push to your fork and submit a pull request to the development branch.

- **Add tests** - None of your code will be accepted if it doesn't have proper tests.
- **Stick to the standards** - Make sure to follow the Symfony coding standards.
- **Document any change in behaviour** - Make sure the Readme and any other relevant documentation are kept up-to-date.
- **Create feature branches** - Don't ever ask us to pull from your master branch.
- **One pull request per feature** - If you want to contribute more than one thing, please send multiple pull requests.
- **Send coherent history** - Make sure each individual commit in your pull request is meaningful. If you had to make multiple intermediate commits while developing, please squash them before submitting.

CHAPTER 10

Quality assurance

10.1 Make use of Gulp

Of cause you can run the tests and validate the code manually, but there is a also more elegant way: Arcella supports Gulp with automates and enhances the workflow. And there are a lot of useful standard tasks included with Arcella. The most simple way is just to execute the default task and then gulp will run all tests, validate the code and watch if something changes. And if you make some changes to the code it will automatically run all the tests and validate all the code again. Pretty awsome, isn't it?

```
cd ~/webroot/arcella
gulp
```

10.2 Run the tests

As Arcella uses PHPUnit for testing it's quite easy to run the tests. Just navigate to the project folder, e.g. ~/webroot/arcella and run PHPunit.

```
cd ~/webroot/arcella
php vendor/bin/phpunit
```

Or do the same thing with Gulp:

```
cd ~/webroot/arcella
gulp test
```

10.3 Validate the code

To make sure that our code does not violate the Symfony coding standards we're using PHPCodeSniffer that automatically detects violations to these coding standards.

```
cd ~/webroot/arcella
php vendor/bin/phpcs src -v --standard=Symfony2
```

Or make use of Gulp for this task:

```
cd ~/webroot/arcella
gulp validate
```

If this command fails with an error, saying that the Symfony2 coding standard is not available than just add it with the following command: