Arcella Documentation

Release 0.0.1

nplhse

November 13, 2016

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.

The main documentation for Arcella is organized into some sections:

- basic-docs
- *Concept Documentation*
- *Developer Documentation*
- *Source Documentation*

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 hautelook_alice: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

Building blocks

Arcella is based on the shoulders of giants and makes use of well established and best-in-class technologies as building blocks. This is to ensure that Arcella is as good as possible, while being simple to use and easy to extend. Some of these key technologies include:

- Symfony Full Stack Framework as foundation for the whole thing
- Doctrine ORM as abstraction for the database
- Markdown as markup for Github
- reStructuredText as markup for the documentation
- **'YAML'_** in the configuration files

„ _YAML: http://yaml.org/

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

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 9

Quality assurance

9.1 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
```

9.2 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
```

9.3 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:

CHAPTER 10

AppBundle

This section of Arcellas documentation provides information about the source code of the **AppBundle**, which is a demo bundle that is included within the standard edition of the symfony framework.

10.1 AppBundle

class `AppBundle`
Class AppBundle @package AppBundle

10.2 Controller/Default

10.3 Entity/Dummy

class `Dummy`
Dummy

@ORMTable(name="dummy") @ORMEntity(repositoryClass="AppBundleRepositoryDummyRepository")

`getId`()
Get id

@return int

`setName`(*$name*)
Set name

@param string $name

@return Dummy

`getName`()
Get name

@return string

10.4 Repository/Dummy

class `DummyRepository`
DummyRepository

This class was generated by the Doctrine ORM. Add your own custom repository methods below.

CHAPTER 11

Domain

This section of Arcellas documentation provides information about the source code within the **Domain**, which defines the code functionality of Arcella.

11.1 NodeInterface

interface `NodeInterface`

Nodes are the main structure that holds contents within Arcella, they consist of an numeric Id, a title and an array containing the contents, where "text" is the default content. Also there are timestamps from the creation and the last update included.

@package ArcellaDomain

`getId`()

Returns the Id of the node.

@return integer Id of the node.

`getTitle`()

Returns the Title of the node.

@return string Title of the node.

`setTitle`(*string $title*)

Updates the Title of the node and returns the changed node with the new title.

@param string $title The new title for the node.

@return mixed The node with the new title.

`getSlug`()

Returns the Slug of the node.

@return string Slug of the node.

`setSlug`(*string $slug*)

Updates the Slug of the node and returns the changed node with the new slug.

@param string $slug The new slug for the node.

@return mixed The node with the new slug.

`getContent`()

Returns the Content of the node.

@return array All the contents inside a associated array, with "text" as default.

setConcent(*array $content*)
Sets Content of the node.

@param array $content The new content for the node.

@return mixed The node with the new content.

getCreated()
Returns the timestamp of the creation of the node.

@return integer Timestamp of the nodes creation.

getUpdated()
Returns the timestamp of the last update of the node.

@return integer Timestamp of the nodes last update.

CHAPTER 12

Indices and tables

- genindex
- search