



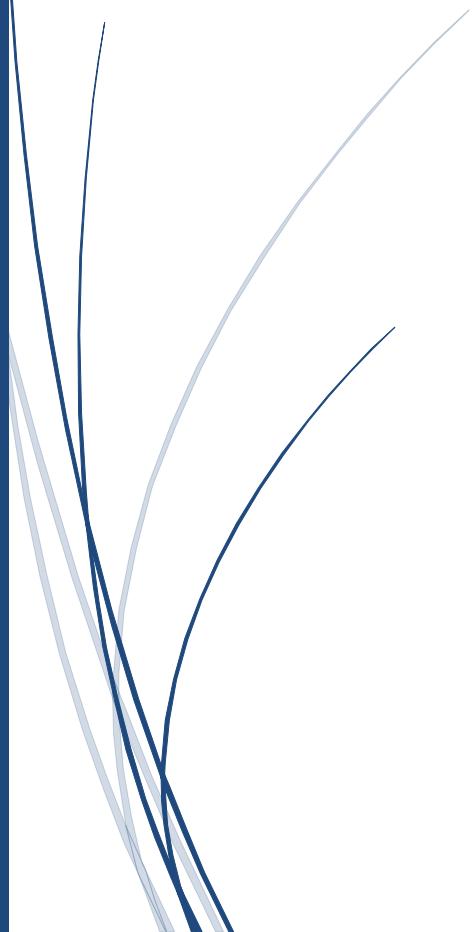
2021 г

FlyCubePHP

Development Guide RUS
ver. 1.4.0

GitHub: <https://github.com/AnthonySnow887/FlyCubePHP>

MIT License



Anthony Snow

Оглавление

Предисловие	7
1. bin/fly_cube_php - инструмент генерации кода и управления проектом	9
2. Создание нового проекта FlyCubePHP и описание его структуры	13
3. Конфигурация проекта и переменные окружения.....	16
4. Контроллеры и их возможности.....	20
4.1. Базовый класс контроллера	20
4.2. Базовый класс графического контроллера.....	23
4.3. Базовый класс API контроллера.....	24
4.4. Создание нового графического контроллера.....	24
4.5. Создание нового API контроллера	28
5. Система маршрутизации приложения	30
6. Asset Pipeline и работа с ресурсами.....	36
6.1. Работа с *.js и *.js.php файлами	37
6.2. Работа с *.css и *.scss файлами	43
6.2.1. Вспомогательные функции системы компиляции Sass Stylesheets	47
6.3. Работа с изображениями.....	48
7. FlashMessages - передача данных между запросами	49
8. Работа с cookie.....	50
8.1. Стандартные cookie.....	51
8.2. Подписанные cookie (signed cookie)	53
8.3. Шифрованные cookie (encrypted cookie)	55
9. Работа с сессиями.....	57
10.Работа с базами данных	58
10.1. Настройка доступа к базам данных	60
11.Модели данных	62
11.1. Создание новой модели данных.....	62
11.2. Возможности моделей данных.....	63
11.2.1. tableName / setTableName	63
11.2.2. primaryKey / setPrimaryKey	64
11.2.3. hasColumnMapping / columnMapping / setColumnMapping	64
11.2.4. passwordColumn / setPasswordColumn.....	66
11.2.5. isAuthenticated	67
11.2.6. isReadOnly / setReadOnly	67

11.2.7.	save	68
11.2.8.	destroy	69
11.2.9.	all	69
11.2.10.	destroyAll	70
11.2.11.	first	70
11.2.12.	limit	71
11.2.13.	count	71
11.2.14.	find	72
11.2.15.	findBy	72
11.2.16.	findBySql	73
11.2.17.	where	74
11.2.18.	exists	74
11.2.19.	existsSome	75
11.2.20.	selectSome	75
11.2.21.	Функции обратного вызова (callbacks)	76
11.2.22.	Специальные функции обработки	77
12.	Миграции Active Record	78
12.1.	Создание новой миграции	79
12.2.	Возможности миграций	80
12.2.1.	up	80
12.2.2.	down	81
12.2.3.	adapterName	81
12.2.4.	createSchema	82
12.2.5.	dropSchema	82
12.2.6.	createTable	83
12.2.7.	renameTable	84
12.2.8.	dropTable	85
12.2.9.	addColumn	85
12.2.10.	renameColumn	86
12.2.11.	changeColumn	87
12.2.12.	changeColumnDefault	87
12.2.13.	changeColumnNull	88
12.2.14.	dropColumn	89
12.2.15.	addIndex	89
12.2.16.	renameIndex	90
12.2.17.	dropIndex	90
12.2.18.	setPrimaryKey	91

12.2.19.	dropPrimaryKey	92
12.2.20.	addForeignKey	92
12.2.21.	addForeignKeyPKey	93
12.2.22.	dropForeignKey	94
12.2.23.	dropForeignKeyPKey.....	95
12.2.24.	execute.....	95
12.3.	Управление базами данных	96
12.3.1.	--db-create.....	96
12.3.2.	--db-create-all	97
12.3.3.	--db-drop	98
12.3.4.	--db-drop-all.....	99
12.3.5.	--db-setup	100
12.3.6.	--db-reset.....	101
12.3.7.	--db-schema-dump.....	102
12.3.8.	--db-schema-load	103
12.3.9.	--db-seed	104
12.4.	Управление миграциями.....	105
12.4.1.	--db-migrate	105
12.4.2.	--db-migrate-redo.....	107
12.4.3.	--db-rollback	108
12.4.4.	--db-rollback-all.....	108
12.4.5.	--db-migrate-status.....	109
12.4.6.	--db-version	109
13.	Классы каталога Network	110
13.1.	HttpClient	110
13.1.1.	HttpResponse.....	113
13.2.	HttpCodes.php	114
14.	Вспомогательные классы	115
14.1.	CoreHelper.php	115
14.2.	Enum.php	120
14.3.	MimeTypes.php.....	121
15.	Вспомогательные методы шаблонов представлений (views helpers)	121
15.1.	Вставка данных в блок HTML Head.....	122
15.2.	params - доступ к параметрам	122
15.3.	flash (FlashMessages) - передача данных между запросами	123
15.4.	router (RouteCollector) - доступ к методам системы маршрутизации.....	124
15.5.	AssetTagHelper.php	124

15.5.1. stylesheet_link_tag	125
15.5.2. javascript_include_tag	125
15.5.3. auto_discovery_link_tag.....	126
15.5.4. favicon_link_tag	127
15.5.5. preload_link_tag	128
15.5.6. image_tag.....	129
15.5.7. link_to.....	130
15.5.8. add_plugin_view_javascripts	131
15.5.9. add_plugin_view_stylesheets	131
15.6. AssetUrlHelper.php.....	132
15.6.1. asset_path.....	132
15.6.2. asset_url	133
15.6.3. javascript_path.....	134
15.6.4. javascript_url	135
15.6.5. stylesheet_path.....	136
15.6.6. stylesheet_url	137
15.6.7. image_path.....	138
15.6.8. image_url	139
15.6.9. image_alt	140
15.6.10. make_valid_url.....	140
15.7. FormTagHelper.php	141
15.7.1. label	141
15.7.2. form_with	141
15.7.3. raw	143
15.7.4. button_tag	143
15.7.5. submit	144
15.7.6. file_field.....	144
15.7.7. hidden_field	145
15.7.8. check_box	145
15.7.9. color_field.....	146
15.7.10. date_field.....	146
15.7.11. datetime_field.....	147
15.7.12. datetime_local_field	147
15.7.13. email_field.....	148
15.7.14. month_field	148
15.7.15. number_field	149
15.7.16. password_field	149

15.7.17.	telephone_field.....	150
15.7.18.	phone_field.....	150
15.7.19.	radio_button	151
15.7.20.	range_field.....	151
15.7.21.	search_field	152
15.7.22.	text_area	152
15.7.23.	text_field.....	153
15.7.24.	time_field	153
15.7.25.	url_field	154
15.7.26.	week_field	154
15.8.	JavascriptTagHelper.php.....	155
15.8.1.	javascript_tag.....	155
15.9.	StylesheetTagHelper.php	156
15.9.1.	stylesheet_tag.....	156
15.10.	ProtectionTagHelper.php	157
15.10.1.	csrf_meta_tags.....	157
15.10.2.	csp_meta_tag	157
16.	Дополнительные инструменты управления проектом	158
16.1.	--secret - генерация секретных ключей приложения	158
16.2.	--check-dirs - проверка каталогов проекта.....	159
16.3.	--assets-precompile - сборка ресурсов.....	159
16.4.	--clear-cache - очистка кэша приложения	160
16.5.	--clear-logs - очистка журналов функционирования приложения.....	160
16.6.	--clear-php-sessions - очистка PHP сессий	161
17.	Менеджер компонентов (плагинов) и его архитектура	162
17.1.	Архитектура компонентов (плагинов)	163
17.2.	Зависимости плагинов	165
17.3.	Структура каталогов плагина	166
17.3.1.	Создание консольного плагина и описание структуры его каталогов	167
17.3.2.	Создание графического плагина и описание структуры его каталогов	168
18.	API Doc - создание описаний для API.....	171
18.1.	Загрузка API Doc и доступ к их объектам	173
18.2.	Разделы API Doc и их PHP классы	174
18.3.	Описание метода API и его PHP класс	176
18.4.	Описание параметра API и его PHP класс	179
18.5.	Описание группы параметров API и её PHP класс	181
18.6.	Описание требуемых входных HTTP заголовков API.....	183

18.7. Описание ответа сервера API и его PHP класс	183
18.8. Описание ошибки сервера API и её PHP класс	185
18.9. Описание примера использования API и его PHP класс	187
19. Система логирования FlyCubePHP	190
20. Система обработки ошибок FlyCubePHP	193
21. Система расширений FlyCubePHP	195
21.1. Расширение для системы доступа к базе данных	195
21.2. Расширение для системы миграций	197
21.3. Расширение вспомогательных методов шаблонов представлений (views helpers)	198
21.4. Расширение для системы обработки ошибок	199
22. Обновление проекта до последней версии ядра FlyCubePHP	200
22.1. Обновление проекта с версией ядра FlyCubePHP ниже 1.3.0	200
22.2. Обновление проекта с версией ядра FlyCubePHP 1.3.0 или старше	201
23. Рекомендации по настройке и развертыванию приложения	202

Предисловие

FlyCubePHP – это MVC Web Framework, разработанный на языке PHP и повторяющий идеологию и принципы построения WEB приложений, заложенный в Ruby on Rails. Основной задачей, поставленной при разработке, являлся быстрый и гибкий перенос кода проектов с Ruby on Rails на PHP с внесением минимального количества изменений в кодовую базу проектов, и сохранением производительности приложений.

Дополнительным функционалом, добавленным в ядро FlyCubePHP, стала поддержка механизма плагинов, позволяющий расширять или изменять функционал Вашего приложения, учитывая зависимости плагинов при работе. По своей структуре плагин частично повторяет архитектуру приложения, внося в него необходимые изменения.

На текущий момент FlyCubePHP поддерживает:

- создание проектов, архитектурно повторяющих проекты Ruby on Rails 5:
 - создание контроллеров (gui / api);
 - создание моделей данных;
 - создание шаблонов WEB страниц;
 - создание вспомогательных классов для доступа из шаблонов WEB страниц;
 - создание JavaScript файлов с автоматической сборкой в *.min.js в production mode и обработкой директив « = require ... » и « = require_tree ... »;
 - создание Sass Stylesheets файлов с автоматической сборкой в CSS (или *.min.css в production mode) и обработкой директив « = require ... » и « = require_tree ... »;
 - создание api-doc файлов для контроллеров на базе json;
 - доступ к ресурсам проекта (изображения, файлы и т.д.) осуществляется с использованием Asset Pipeline;
 - для ускорения загрузки ресурсов реализовано:
 - автоматическое сжатие ресурсов с помощью «gzip» и «deflate»;
 - автоматическая проверка HTTP заголовок «ETag».
- создание плагинов, расширяющих или изменяющих функционал Вашего приложения;
- создание миграций базы данных и широкий спектр инструментов по работе с миграциями;
- создание расширений ядра FlyCubePHP позволяющих:
 - добавить поддержку необходимой СУБД;
 - добавить поддержку необходимой СУБД для системы миграций;
 - добавить дополнительные вспомогательные классы для доступа из шаблонов WEB страниц;
 - добавить поддержку необходимых инструментов по предобработке и сборке JavaScript файлов;
 - добавить поддержку необходимых инструментов по предобработке и сборке Stylesheets файлов.

FlyCubePHP поддерживает следующие СУБД:

- SQLite 3;
- PostgreSQL 9.6 или старше;
- MySQL/MariaDB (tested on MariaDB 10.2.36).

Сторонние зависимости, необходимые для корректного функционирования фреймворка:

- Twig-2.x - используется для формирования и рендеринга шаблонов web-страниц (<https://github.com/twigphp/Twig/archive/refs/tags/v2.12.5.zip>);
- JShrink - используется для сборки *.min.js файлов в production mode (<https://github.com/tedious/JShrink/archive/refs/tags/v1.4.0.zip>);
- ScssPhp - используется для сборки Sass Stylesheets файлов в css (<https://github.com/scssphp/scssphp/archive/refs/tags/v1.8.1.zip>);
- Monolog - используется для ведения журналов функционирования приложения (логирования) (<https://github.com/Seldaek/monolog/archive/refs/tags/1.26.1.zip>).

ПРИМЕЧАНИЕ: необходимые сторонние зависимости будут автоматически скачаны при создании нового проекта FlyCubePHP. Данное поведение можно отключить, задав входной аргумент «--download-requires=false».

Основные системные требования:

- PHP >= 7.0

Дополнительные необходимые PHP модули:

- php7-ctype
- php7-curl
- php7-json
- php7-mbstring
- php7-mysql
- php7-openssl
- php7-pdo
- php7-pgsql
- php7-posix
- php7-sqlite
- php7-xmlreader
- php7-xmlwriter
- php7-zip
- php7-zlib

Операционные системы, на которых производилось тестирование:

- OpenSUSE 15.1 и старше
- CentOS 8
- Astra Linux SE 1.6 (Linux kernel: 4.15.3-1)

Лицензия: MIT License.

1. bin/fly_cube_php - инструмент генерации кода и управления проектом

Для создания нового проекта FlyCubePHP или его компонентов (контроллеры, миграции данных, модели данных и т.д.) в проекте присутствует исполняемый файл bin/fly_cube_php, позволяющий производить все необходимые манипуляции с проектом.

Перечень поддерживаемых команд исполняемого файла bin/fly_cube_php, расположенного в ядре FlyCubePHP:

```
FlyCubePHP/bin> php ./fly_cube_php --help
```

```
Usage: ./fly_cube_php [options]
```

```
Options include:
```

--help	Show this message [-h, -?]
--new	Create new FlyCubePHP project
--name=[VALUE]	Set new project name
--path=[VALUE]	Set new project root path (optional; default: user home)
--download-requires=[true/false]	Download FlyCubePHP requires in new project (optional; default: true)
--version	Print the version [-v]
--latest-version	Select latest version from GitHub [-lv]
--upgrade	Update FlyCubePHP core version of your project to the latest version
--force	Forced update flag

```
Examples:
```

1. Create new FlyCubePHP project:

```
./fly_cube_php --new --project --name=MyProject
```

2. Create new FlyCubePHP project without requires:

```
./fly_cube_php --new --project --name=MyProject --download-requires=false
```

3. Create new FlyCubePHP project specifying the installation directory:

```
./fly_cube_php --new --project --name=MyProject --path=/home/test/projects
```

4. Upgrade an existing FlyCubePHP project:

```
./fly_cube_php --upgrade --path=/home/test/projects/MyProject
```

5. Force upgrade an existing FlyCubePHP project:

```
./fly_cube_php --upgrade --force --path=/home/test/projects/MyProject
```

При создании нового проекта FlyCubePHP в него автоматически будет скопирован данный исполняемый файл, но уже с более широким функционалом. Перечень поддерживаемых команд исполняемого файла bin/fly_cube_php, расположенного в новом проекте (где «MyProject» - название нового проекта; создание нового проекта будет рассмотрено позже):

```

MyProject/bin> ./fly_cube_php -h

Usage: ./fly_cube_php [options]

Options include:

--help                                Show this message [-h, -?]
--version                             Print the version [-v]
--latest-version                      Select latest version from GitHub [-lv]
--output=[true/false]                  Show output (optional)
--env=[VALUE]                           Set current environment (production/development;
                                         default: development)

--upgrade                             Update FlyCubePHP core version of your project
                                         to the latest version

--force                               Forced update flag

--new [option]                          Create new object (project / controller / model /
                                         migration / plugin / ...etc)

--project                            Create new FlyCubePHP project
--controller                         Create new controller
--controller-api                     Create new API controller
--model                               Create new model
--migration                          Create new migration
--plugin                             Create new FlyCubePHP plugin
--plugin-gui                         Create new FlyCubePHP Gui plugin
--plugin-controller                  Create new FlyCubePHP plugin controller
--plugin-controller-api              Create new FlyCubePHP plugin API controller
--plugin-model                       Create new FlyCubePHP plugin model
--plugin-migration                   Create new FlyCubePHP plugin migration
--path=[VALUE]                        Set new project root path (optional; default: user home)
--download-requires=[true/false]      Download FlyCubePHP requires in new project
                                         (optional; default: true)

--name=[VALUE]                        Set new object name
--plugin-name=[VALUE]                 Set plugin name
--actions=[VALUE( ,VALUE...)]         Set new controller actions list

--db-create                           Create database for current environment
--db-create-all                      Create databases for all environments
                                         (development and production)

--db-drop                            Drop database for current environment
--db-drop-all                        Drop databases for all environments
                                         (development and production)

--db-setup                           Init database for current environment
--db-reset                           Re-Init database for current environment
--db-seed                            Load database Seed.php file
--db-migrate                         Start database migrations
--db-migrate-redo                    Start re-install last database migration
--db-migrate-status                  Select migrations status
--db-rollback                        Start uninstall last database migration
--db-rollback-all                   Start uninstall all database migrations
--db-version                         Select current database migration version
--db-schema-dump                    Create database schema dump
--db-schema-load                     Re-Create database and load schema dump
--to-version=[VALUE]                 Set needed migration version
                                         (optional; if 0 - uninstall all migrations)

--step=[VALUE]                        Set needed number of steps for uninstall (re-install)
                                         migrations (optional; default: 1)

--routes                             Show application routes

--secret                            Create application secret key
--secret-length=[VALUE]               Set application secret key length (value > 0)

--check-dirs                          Check the project catalogs and create the missing or
                                         change their rights to the necessary

--assets-precompile                  Build assets cache
--clear-cache                        Clear all application cache
--clear-logs                          Clear all application logs
--clear-php-sessions                 Clear all php sessions

```

Examples:

1. Create new FlyCubePHP project:
./fly_cube_php --new --project --name=MyProject
2. Create new FlyCubePHP project without requires:
./fly_cube_php --new --project --name=MyProject --download-requires=false
3. Create new controller without actions:
./fly_cube_php --new --controller --name=Example
4. Create new controller with actions:
./fly_cube_php --new --controller --name=Example --actions=act_1,act_2
5. Create new plugin:
./fly_cube_php --new --plugin --name=ExamplePlugin
6. Create new gui plugin:
./fly_cube_php --new --plugin-gui --name=ExamplePlugin
7. Create new plugin controller:
./fly_cube_php --new --plugin-controller --plugin-name=ExamplePlugin --name=ExampleController
8. Install all migrations:
./fly_cube_php --db-migrate
9. Install needed migrations:
./fly_cube_php --db-migrate --to-version=20210309092620
10. Uninstall last migration:
./fly_cube_php --db-rollback
11. Uninstall last N-steps migrations:
./fly_cube_php --db-rollback --step=3
12. Upgrade current FlyCubePHP project:
./fly_cube_php --upgrade
13. Upgrade an existing FlyCubePHP project:
./fly_cube_php --upgrade --path=/home/test/projects/MyProject
14. Force upgrade an existing FlyCubePHP project:
./fly_cube_php --upgrade --force --path=/home/test/projects/MyProject

В случае возникновения ошибок при выполнении команд исполняемый файл bin/fly_cube_php выводит подробную информацию с указанием места ошибки, если удаётся его определить. Пример вывода ошибки при сборке ресурсов:

```
MyProject/bin> ./fly_cube_php --assets-precompile

==== FlyCubePHP: Assets precompile ====
[Render] ApiTestController->test:

==== FlyCubePHP: ERROR =====
Error title: FlyCubePHP Error (in NetworkBase.php)

Cannot modify header information - headers already sent by (output started at
/home/test/FlyCubePHProjects/MyProject/bin/fly_cube_php:2537)

File: /vendor/FlyCubePHP/FlyCubePHP/Core/Controllers/Extensions/NetworkBase.php (line: 237)
Type: E_WARNING
PHP 7.2.5 (Linux)

Code:
-----
227 |     http_response_code($status);
228 |     $cLength = false;
229 |     foreach ($headers as $key => $value) {
230 |         if (strcmp("content-type", strtolower(trim($key))) === 0)
231 |             continue;
```

```

232 |             if (strcmp("content-length", strtolower(trim($key))) === 0)
233 |                 $cLength = true;
234 |                 header("$key: $value");
235 |
236 |             if ($cLength === false)
237 |                 header("Content-Length: ".mb_strlen($body));
238 |
239 |             header("Content-Type: $contentType; charset=".strtoupper($encoding));
240 |
241 |             if (ob_get_level())
242 |                 ob_end_clean();
243 |
244 |             echo $body;
245 |
246 |         }
247 |     /**
-----  

Backtrace:  

-----
1 | /app/controllers/ApiTestController.php(14): FlyCubePHP\Core\Controllers\BaseController->send_response()
2 | /vendor/FlyCubePHP/FlyCubePHP/Core/Controllers/BaseActionControllerAPI.php(67):
ApiTestController->test()
3 | /bin/fly_cube_php(2265): FlyCubePHP\Core\Controllers\BaseActionControllerAPI->renderPrivate('test')
-----  

==== FlyCubePHP: END ERROR =====

```

Подробное описание использования каждой из команд исполняемого файла `bin/fly_cube_php` будет рассмотрено позже в отдельных главах.

2. Создание нового проекта FlyCubePHP и описание его структуры

Для создания нового проекта FlyCubePHP требуется выполнить следующую команду генератора «./fly_cube_php --new --name=MyProject»:

```
FlyCubePHP/bin> php ./fly_cube_php --new --name=MyProject

==== FlyCubePHP: Create new project "MyProject" ====
- Create project dir: OK
- Create project tree: OK
- Copy FlyCubePHP core: OK
- Copy FlyCubePHP templates: OK
- Download requires [Twig]: OK
- Download requires [JShrink]: OK
- Download requires [ScssPhp]: OK
- Download requires [Psr/Log]: OK
- Download requires [Monolog]: OK
- Unzip requires [Twig]: OK
- Unzip requires [JShrink]: OK
- Unzip requires [ScssPhp]: OK
- Unzip requires [Psr/Log]: OK
- Unzip requires [Monolog]: OK
==== FlyCubePHP: Create project success. ====
==== FlyCubePHP: Dir: /home/test/FlyCubePHProjects/MyProject ===
```

Новый проект FlyCubePHP архитектурно представляет из себя классическое WEB приложение Ruby on Rails 5, с возможностью расширения функционала средствами плагинов. Механизм загрузки плагинов реализует библиотека менеджера компонентов, входящая в состав ядра FlyCubePHP (см. описание ниже).

Ниже представлена структура основных каталогов и файлов нового проекта FlyCubePHP:

- app [Каталог приложения]
 - assets
 - images [Изображения: png, jpg, jpeg, svg, gif]
 - javascripts [Скрипты: *.js, *.js.php]
 - application.js
 - stylesheets [Стили: *.css, *.scss]
 - application.css
 - controllers [Контроллеры]
 - ApplicationController.php [Базовый класс графического контроллера]
 - ApplicationControllerAPI.php [Базовый класс API контроллера]
 - helpers [Вспомогательные классы представлений (views)]
 - ApplicationHelper.php
 - models [Модели данных]
 - views [Представления: *.html, *.html.twig]
 - layouts
 - application.html
- bin
 - fly_cube_php
- config [Конфигурационные файлы приложения]
 - environments [Конфигурации режимов работы приложения]
 - development.php
 - production.php
 - initializers [Каталог файлов дополнительной инициализации]
 - application_env.conf [Файл переменных окружения для приложения]
 - database.json [Файл конфигурации доступа к БД]
 - ignore-list.conf [Файл списка игнорируемых плагинов приложения]
 - routes.php [Файл маршрутов приложения]
- db [Файлы баз данных приложения]
 - migrate [Файлы миграций]

- doc [Файлы генерации API-DOC и справки]
 - api [Файлы генерации API-DOC]
 - help [Файлы генерации справки]
 - images [Изображения для генерации справки]
- extensions [Файлы расширений ядра FlyCubePHP]
 - asset_pipeline [Файлы расширений Asset Pipeline]
 - css_builder [Файлы расширений Asset Pipeline - CSS Builder]
 - js_builder [Файлы расширений Asset Pipeline - JS Builder]
 - controller [Файлы расширений контроллеров]
 - helpers [Дополнительные вспомогательные классы представлений (views)]
 - db [Файлы расширений по работе с СУБД]
 - adapters [Расширения адаптеров по работе с СУБД]
 - migrators [Расширения системы миграций по работе с СУБД]
 - error_handling [Файлы расширений системы обработки ошибок]
- lib [Библиотеки, разработанные в рамках проекта]
 - assets
 - images
 - javascripts
 - stylesheets
 - log [Журналы функционирования приложения]
 - plugins [Плагины приложения]
 - tmp [Временные файлы приложения]
 - vendor [Библиотеки сторонних производителей]
 - FlyCubePHP
 - JShrink
 - Monolog
 - Psr
 - ScssPhp
 - Twig-2.x
 - assets
 - images
 - javascripts
 - stylesheets
 - index.php [Основная index страница приложения]
 - my_project.apache24.conf [Файл конфигурации приложения для Apache 2.4]

После генерации нового проекта рекомендуется скопировать его конфигурационный файл «my_project.apache24.conf» для web-сервера Apache 2.4 и проверить функционирование приложения (выполнить запрос в браузере: http://127.0.0.1:8080/my_project/).

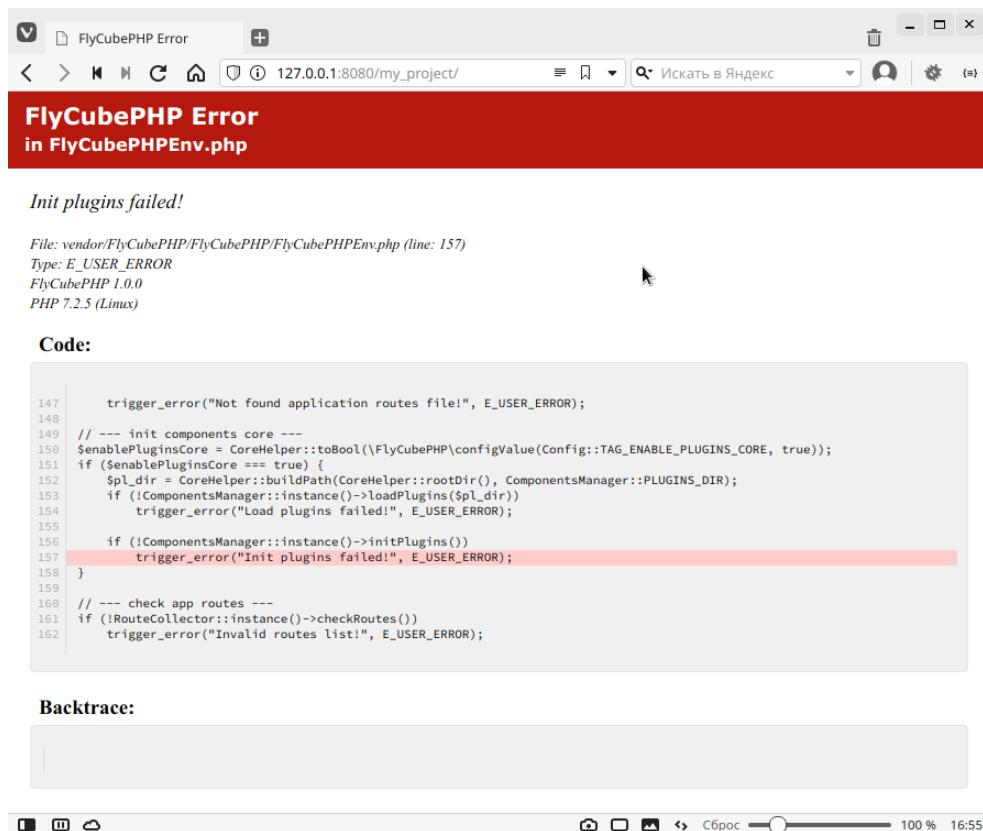


Рис. 1. Первая проверка функционирования нового проекта.

ПРИМЕЧАНИЕ: ошибка, информирующая о некорректной инициализации плагинов, говорит лишь о том, что в проекте отсутствуют плагины или ни один из них не был загружен.

Для того чтобы отключить проверку загрузки и инициализации плагинов требуется выставить переменные окружения «FLY_CUBE_PHP_ENABLE_PLUGINS_CORE: false» и «FLY_CUBE_PHP_CHECK_PLUGINS_COUNT: false» в файле конфигурации приложения «[MyProject]/config/application_env.conf» (где «MyProject» - название нового проекта):

```

#
# --- plugins system settings ---
#
# In production mode:
# - enable plugins core      - default true
# - check plugins count      - default true
#
# In development mode:
# - enable plugins core      - default true
# - check plugins count      - default true
#
FLY_CUBE_PHP_ENABLE_PLUGINS_CORE: false
FLY_CUBE_PHP_CHECK_PLUGINS_COUNT: false

```

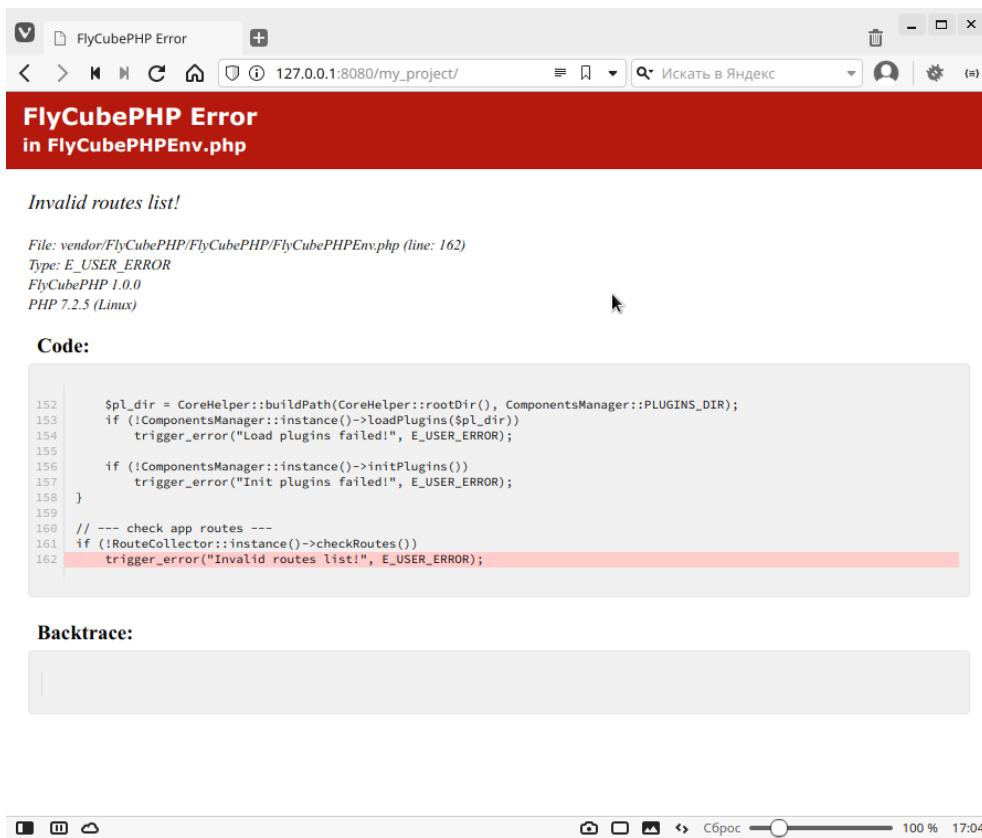


Рис. 2. Проверка функционирования нового проекта с отключенными плагинами.

ПРИМЕЧАНИЕ: ошибка, информирующая о некорректном списке маршрутов, говорит лишь о том, что в проекте отсутствуют контроллеры или не задан список маршрутов в файле «[MyProject]/config/routes.php» (где «MyProject» - название нового проекта).

3. Конфигурация проекта и переменные окружения

Все конфигурации проекта располагаются в каталоге «[MyProject]/config/». Если вывести содержимое данного каталога после генерации нового проекта, то должны получить список следующих файлов:

- `application_env.conf` - основной файл конфигураций проекта; содержит базовый набор переменных окружения;
- `database.json` - файл конфигурации доступа к базам данных;
- `ignore-list.conf` - файл конфигурации игнорируемы плагинов ядром приложения;
- `routes.php` - основной файл маршрутов приложения;
- `environments/development.php` - файл environments, загружаемый только в «development mode»;
- `environments/production.php` - файл environments, загружаемый только в «production mode»;
- `initializers/` - каталог дополнительных инициализаторов приложения; данные php-файлы загружаются после полной инициализации приложения в алфавитном порядке.

Так же каталог «[MyProject]/config/» должен содержать файл секретного ключа приложения «secret.key», который требуется для запуска приложения в «production mode». Для создания данного файла можно воспользоваться командой:

```
MyProject/bin> ./fly_cube_php --secret | grep -E --only-matching -e "[0-9a-f]{128}" > ../config/secret.key
```

Содержимое файла секретного ключа должно выглядеть следующим образом:

```
MyProject/bin> cat ../config/secret.key
```

```
ff46f41e287f005a659c08493f158315f25ff98f7ff1732599612630109edf879ba49c00cc157af0698f7fb3fbad43d14  
0c7c8c888c74aadd02548d14eab47e0
```

Основной файл конфигураций проекта «application_env.conf» содержит несколько разделов, содержащих различные настройки для функционирования приложения. Все эти переменные загружаются ядром конфигурирования при старте приложения. Получить доступ к необходимым параметрам конфигурирования можно используя класс «\FlyCubePHP\Core\Config\Config». Перечень доступных методов данного класса:

- `->secretKey()` - получить секретный ключ приложения;
- `->keys()` - получить список ключей для загруженных аргументов;
- `->args()` - получить массив загруженных аргументов;
- `->arg(string $key, $def = null)` - получить значение аргумента настроек, где «\$def» - базовое значение, возвращаемое при отсутствии искомого;
- `->setArg(string $key, $val)` - задать новое значение аргумента настроек;
- `->isProduction()` - проверка, является ли текущий режим работы приложения «production mode»;
- `->isDevelopment()` - проверка, является ли текущий режим работы приложения «development mode».

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Config/Config.php».

Так же можно воспользоваться вспомогательным файлом «ConfigHelper», упрощающим доступ к данным класса конфигурации. Перечень доступных методов файла:

- `\FlyCubePHP\setConfigValue(string $key, $val)` - задать новое значение аргумента настроек;
- `\FlyCubePHP\configValue(string $key, $def = null)` - получить значение аргумента настроек, где «\$def» - базовое значение, возвращаемое при отсутствии искомого;
- `\FlyCubePHP\isProduction()` - проверка, является ли текущий режим работы приложения «production mode»;
- `\FlyCubePHP\isDevelopment()` - проверка, является ли текущий режим работы приложения «development mode».

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Config/ConfigHelper.php».

Перечень доступных переменных окружения для настройки приложения приводится ниже в таблице 1.

Таблица 1.

Переменная	Рекомендуемое значение		Возможные значения	Описание
	production mode	development mode		
FLY_CUBE_PHP_ENV	production	development	production development	Режим работы приложения. Если не задан, то значение равно «development».
APP_URL_PREFIX			/ [APP URL]	URL префикс приложения.
FLY_CUBE_PHP_REBUILD_CACHE	false	true	true, false	Всегда пересобирать кэш приложения.
FLY_CUBE_PHP_REBUILD_TWIG_CACHE	false	true	true, false	Всегда пересобирать кэш шаблонизатора страниц Twig.
FLY_CUBE_PHP_CSRF_PROTECT	true	true	true, false	Использовать CSRF Protect.
FLY_CUBE_PHP_CSP_PROTECT	false	false	true, false	Использовать CSP Protect.
FLY_CUBE_PHP_COOKIE_SIGNED_SALT	Secret Key 16bit	"signed cookie"	Any string	Ключ шифрования подписанных cookie.
FLY_CUBE_PHP_COOKIE_ENCRYPTED_SALT	Secret Key 16bit	"authenticated encrypted cookie"	Any string	Ключ шифрования шифрованных cookie.
FLY_CUBE_PHP_ENABLE_ACTION_OUTPUT	false	true	true, false	Разрешить вывод из методов контроллеров (например, используя метод «echo»).
FLY_CUBE_PHP_ENABLE_TWIG_STRICT_VARIABLES	true	true	true, false	Включить проверку переменных для Twig (если требуемая переменная не найдена, то генерируется исключение).
FLY_CUBE_PHP_ENABLE_TWIG_DEBUG_EXTENSION	false	true	true, false	Включить поддержку расширения Twig-Debug.
FLY_CUBE_PHP_CHECK_DUPLICATE_HELPERS	false	true	true, false	Включить проверку дубликатов вспомогательных методов представлений.
FLY_CUBE_PHP_ENABLE_PLUGINS_CORE	true	true	true, false	Включить поддержку ядра системы плагинов.
FLY_CUBE_PHP_CHECK_PLUGINS_COUNT	true	true	true, false	Включить проверку количества загруженных и инициализированных плагинов (если их число равно «0», то генерируется исключение).
FLY_CUBE_PHP_ENABLE_EXTENSION_SUPPORT	false	false	true, false	Включить поддержку расширений ядра FlyCubePHP.
FLY_CUBE_PHP_EXTENSIONS_FOLDER	"extensions"	"extensions"	Directory name	Задать каталог расширений ядра FlyCubePHP в дереве каталогов проекта.
FLY_CUBE_PHP_ENABLE_LOG	true	true	true, false	Включить поддержку системы логирования.
FLY_CUBE_PHP_ENABLE_ROTATE_LOG	true	false	true, false	Включить систему ротации log-файлов (запись поверх старых файлов).
FLY_CUBE_PHP_LOG_ROTATE_MAX_FILES	10	10	Int number	Задать максимальное число log-файлов при ротации.
FLY_CUBE_PHP_LOG_ROTATE_FILE_DATE_FORMAT	"Y_m_d"	"Y_m_d"	PHP date format	Задать формат даты в имени log-файлов при ротации

FLY_CUBE_PHP_LOG_ROTATE_FILE_NAME_FORMAT	"{date}_{filename}"	"{date}_{filename}"	"{date}_{filename}" "{date}-{filename}" "{date}:{filename}" etc...	Задать формат именования log-файлов при ротации, где: • {date} - дата создания файла; • {filename} - имя файла.
FLY_CUBE_PHP_LOG_LEVEL	warning	debug	debug info warning error	Задать уровень логирования.
FLY_CUBE_PHP_LOG_FOLDER	"log"	"log"	Directory name	Задать каталог для log-файлов в дереве каталогов проекта.
FLY_CUBE_PHP_LOG_DATE_TIME_FORMAT	"d.m.Y H:i:s"	"d.m.Y H:i:s"	PHP date-time format	Задать формат даты-времени в log-файле.
FLY_CUBE_PHP_ENABLE_API_DOC	false	false	true, false	Включить поддержку и автоматический поиск API-Doc файлов.
FLY_CUBE_PHP_ENABLE_ASSETS_COMPRESSION	true	false	true, false	Включить поддержку сжатия ресурсов. Если данный функционал поддерживает клиент (браузер), то будут возвращаться сжатые файлы ресурсов.
FLY_CUBE_PHP_ASSETS_COMPRESSION_TYPE	gzip	gzip	gzip deflate	Задать тип сжатия ресурсов.

4. Контроллеры и их возможности

Контроллер - это объект, задачей которого является обработка запроса клиента и возврат кода ответа или результата в той или иной форме. Все контроллеры в FlyCubePHP наследуются от базового класса контроллера «\FlyCubePHP\Core\Controllers\BaseController», а маршрутизация описывается в файлах маршрутов приложения, например: «[MyProject]/config/routes.php» (где «MyProject» - название нового проекта). Вызов методов необходимого контроллера происходит автоматически в соответствии с заданными маршрутами при помощи объекта «RouteCollector», который является ядром системы маршрутизации FlyCubePHP.

4.1. Базовый класс контроллера

Базовый класс контроллера, который лежит в основе всех контроллеров приложения, представлен классом «\FlyCubePHP\Core\Controllers\BaseController». Данный класс предоставляет базовый набор методов, которые могут быть использованы в дочерних классах, и методы, обязательные к реализации.

Методы, обязательные к реализации в наследуемых классах:

- `->renderPrivate(string $action)` - базовый метод контроллера, для обработки входящего запроса; используется ядром FlyCubePHP для вызова методов контроллера, где «\$action» - название метода контроллера, который необходимо вызвать.

Вспомогательные методы базового контроллера:

- `->evalException(\Throwable $ex)` - метод обработки исключений; автоматически вызывается ядром FlyCubePHP при перехвате необработанного исключения от контроллера; ПРИМЕЧАНИЕ: для обработки перехваченных исключений требуется переопределить данный метод в вашем классе контроллера;
- `->appendBeforeAction(string $checkMethod)` - добавить обработчик перед вызовом основного метода контроллера, где «\$checkMethod» - название метода проверки текущего класса, который будет вызван;
- `->prependBeforeAction(string $checkMethod)` - добавить обработчик в начало очереди перед вызовом основного метода контроллера, где «\$checkMethod» - название метода проверки текущего класса, который будет вызван;
- `->skipBeforeAction(string $checkMethod, string|array $action)` - исключить обработчик (или несколько обработчиков) перед вызовом основного метода контроллера, где «\$checkMethod» - название метода проверки текущего класса, который должен быть отключен, «\$action» - название метода (массив с названиями методов), для которого применяется условие;

- `->appendAfterAction(string $checkMethod)` - добавить обработчик после вызова основного метода контроллера, где «\$checkMethod» - название метода проверки текущего класса, который будет вызван;
- `->prependAfterAction(string $checkMethod)` - добавить обработчик в начало очереди после вызова основного метода контроллера, где «\$checkMethod» - название метода проверки текущего класса, который будет вызван;
- `->skipAfterAction(string $checkMethod, string|array $action)` - исключить обработчик (или несколько обработчиков) после вызова основного метода контроллера, где «\$checkMethod» - название метода проверки текущего класса, который должен быть отключен, «\$action» - название метода (массив с названиями методов), для которого применяется условие;
- `->controllerClassName()` - имя класса текущего контроллера;
- `->controllerName()` - имя текущего контроллера;
- `->controllerDirectory()` - каталог текущего контроллера.
- `->redirect_to(array $options = [])` - отправить клиенту перенаправление на другой URL, где доступны следующие ключи массива «\$options»:
 - `status [int]` - задать HTTP код ответа (default: 303);
 - `url [string]` - задать URL перенаправления;
 - `controller [string]` - задать имя контроллера, на который нужно выполнить перенаправление;
 - `action [string]` - задать имя метода контроллера, на который нужно выполнить перенаправление; ПРИМЕЧАНИЕ: «url» и «controller» + «action» взаимоисключающие ключи!
- `->redirect_back(array $options = [])` - отправить клиенту перенаправление на предыдущий URL, если он задан, или использовать аргументы входных данных, где доступны следующие ключи массива «\$options»:
 - `status [int]` - задать HTTP код ответа (default: 303);
 - `url [string]` - задать URL перенаправления;
 - `controller [string]` - задать имя контроллера, на который нужно выполнить перенаправление;
 - `action [string]` - задать имя метода контроллера, на который нужно выполнить перенаправление; ПРИМЕЧАНИЕ: «url» и «controller» + «action» взаимоисключающие ключи!
- `->send_head(array $options = [])` - отправить клиенту только HTTP заголовки, где доступны следующие ключи массива «\$options»:
 - `status [int]` - задать HTTP код ответа (default: 200);
 - `headers [array]` - задать HTTP заголовки;
- `->send_response(array $options = [])` - отправить ответ клиенту, где доступны следующие ключи массива «\$options»:
 - `status [int]` - задать HTTP код ответа (default: 200);
 - `content-type [string]` - задать HTTP Content Type (default: text/html);
 - `encoding [string]` - задать HTTP Encoding (default: utf-8);
 - `headers [array]` - задать дополнительные HTTP заголовки;
 - `body [string]` - задать тело ответа.

- `->send_response_data(array $options = [])` - отправить ответ клиенту в форме бинарных данных, где доступны следующие ключи массива «\$options»:
 - `status [int]` - задать HTTP код ответа (default: 200);
 - `content-type [string]` - задать HTTP Content Type (default: get from mime-types);
 - `encoding [string]` - задать HTTP Encoding (default: utf-8);
 - `headers [array]` - задать дополнительные HTTP заголовки;
 - `data [string]` - задать передаваемые данные;
 - `filename [string]` - задать название передаваемого файла (может быть пустым).
- `->send_response_file(string $path, array $options = [])` - отправить файл клиенту, где доступны следующие ключи массива «\$options»:
 - `status [int]` - задать HTTP код ответа (default: 200);
 - `headers [array]` - задать дополнительные HTTP заголовки.
- `->send_response_file_data(string $path, array $options = [])` - отправить содержимое файла клиенту, где доступны следующие ключи массива «\$options»:
 - `status [int]` - задать HTTP код ответа (default: 200);
 - `headers [array]` - задать дополнительные HTTP заголовки.

ПРИМЕЧАНИЕ: методы, добавленные как обработчики «before/after action» могут возвращать тип «bool». Если данный метод вернет «false», то дальнейшая обработка будет остановлена и ядро FlyCubePHP попытается завершить выполнения скрипта. Если возвращен иной тип или метод вообще не возвращает результат (void), то результат игнорируется.

Пример использования метода «appendBeforeAction»:

```
class MyClass extends BaseController
{
    public function __construct() {
        if (RequestForgeryProtection::instance()->isProtectFromForgery())
            $this->appendBeforeAction("verifyAuthenticityToken");

        $this->appendBeforeAction("verifyAuthenticity");
    }

    /**
     * Метод верификации токена аутентификации
     */
    final private function verifyAuthenticityToken() {
        ...
    }

    /**
     * Метод верификации 2
     * @return bool
     */
    final private function verifyAuthenticity(): bool {
        ...
        return true;
    }
}
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/BaseController.php».

4.2. Базовый класс графического контроллера

Базовый класс графического контроллера предоставляет инструменты по рендерингу HTML страниц средствами шаблонизатора Twig. Он представлен классом «\FlyCubePHP\Core\Controllers\BaseActionController». Данный класс предоставляет базовый набор методов, которые могут быть использованы при обработке запросов и в дочерних классах, а так же методы, предоставляемые базовым классом контроллера «\FlyCubePHP\Core\Controllers\BaseController».

Вспомогательные методы базового контроллера:

- `->render(string $action, array $options)` - выполнить рендеринг страницы, где `«$action»` - название текущего метода контроллера (если не задан - определяется автоматически), `«$option»` - массив дополнительных свойств:
 - `layout_support [bool]` -рендерить или нет базовый слой контроллера (`default: true`);
 - `layout [string]` - задать базовый слой контроллера (должен располагаться в каталоге «app/views/layouts/»);
 - `skip_render [bool]` - пропустить отрисовку страницы.
- `->setLayout(string $name)` - задать базовый слой контроллера, где `«$name»` - название `*.html / *.html.twig` файла, расположенного в «app/views/layouts/».
- `->setLayoutSupport(bool $val)` - задать поддержку отрисовки базового слоя контроллера;
- `->skipRenderForAction(string $action)` - добавить название метода, для которого будет пропущен рендеринг страницы;
- `->skipRenderForActions(array $actions)` - добавить название методов, для которых будет пропущен рендеринг страницы;
- `->appendHelperMethod(string $name, array $settings = [])` - добавить вспомогательную функцию, расположенную в теле класса контроллера (разрешено использование только публичных функций).

ПРИМЕЧАНИЕ: вспомогательная функция не может обрабатывать входящие запросы и выступать в роли action-а в маршруте. Если ядро маршрутизации обрабатывает такой некрректный маршрут - выбрасывается исключение и выполнение всех скриптов прекращается.

Пример добавления вспомогательной функции, расположенной в теле контроллера:

ПРИМЕЧАНИЕ:

Разрешена регистрация только публичных (`public`) методов!

Публичные статические (`static public`) методы также разрешены.

ПРИМЕЧАНИЕ:

Метод контроллера, заданный как вспомогательная функция, не может быть методом отрисовки страницы и будет проигнорирован при формировании списка маршрутов!

```

===== Settings

- safe           - безопасная функция (вывод без экранирования) (default: false)
- need_context   - требуется twig context (default: false)
- needs_environment - требуется twig environment (default: false)

===== Examples

class MyClass extends BaseController
{
    public function __construct() {
        $this->appendHelperMethod('my_f_1', ['safe'=>true]);
        $this->appendHelperMethod('my_f_1_1', ['safe'=>true]);
        $this->appendHelperMethod('my_f_2', ['need_context'=>true]);
        $this->appendHelperMethod('my_f_3', ['need_context'=>true, 'needs_environment'=>true]);
    }

    public function my_f_1($a, $b) { ... }
    static public function my_f_1_1($a, $b) { ... }
    public function my_f_2($context, $a, $b) { ... }
    public function my_f_3(\Twig\Environment $env, $context, $string) { ... }
}

```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/BaseActionController.php».

4.3. Базовый класс API контроллера

Базовый класс API контроллера предоставляет инструменты по обработке входящих запросов без возможности рендеринга HTML страниц. Он представлен классом «\FlyCubePHP\Core\Controllers\BaseActionControllerAPI». Данный класс предоставляет только методы базового класса контроллера «\FlyCubePHP\Core\Controllers\BaseController».

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/BaseActionControllerAPI.php».

4.4. Создание нового графического контроллера

Для создания нового графического контроллера требуется выполнить следующую команду генератора «./fly_cube_php --new --controller --name=Test --actions=root,test,test_2»:

```

MyProject/bin> ./fly_cube_php --new --controller --name=Test --actions=root,test,test_2

==== FlyCubePHP: Create new controller ===
[Created] app/controllers/TestController.php
[Created] app/helpers/TestHelper.php
[Created] app/assets/javascripts/test.js
[Created] app/assets/stylesheets/test.scss
[Created] app/views/test
[Created] app/views/test/root.html
[Created] app/views/test/test.html
[Created] app/views/test/test_2.html
==== FlyCubePHP =====

```

ПРИМЕЧАНИЕ: указывать список методов контроллера в виде входного аргумента «`--actions=root,test,test_2`» необязательно и может быть проигнорировано при генерации.

После генерации контроллера все его методы автоматически добавляются в основной файл маршрутов приложения «`[MyProject]/config/routes.php`» (где «`MyProject`» - название нового проекта):

```
namespace FlyCubePHP;

// NOTE: Set routes here...
get("/root", [ "to" => "Test#root" ] ;
get("/test", [ "to" => "Test#test" ] ;
get("/test_2", [ "to" => "Test#test_2" ]);
```

Все представления нового контроллера, расположенные в каталоге «`app/views/test/*.html`», являются шаблонами страниц, рендеринг которых производится библиотекой Twig с дополнительно подключенными вспомогательными классами (`helpers`), которые предоставляют различные функции и инструменты для гибкой генерации web страниц. Более подробно доступные методы вспомогательных классов (`helpers`) будут рассмотрены позже.

ПРИМЕЧАНИЕ: на текущий момент фреймворк поддерживает следующие расширения для файлов представлений, при этом различий при рендеринге нет:

- `*.html;`
- `*.html.twig.`

Для проверки успешной работы требуется выполнить запрос в браузере к любому из созданных методов нового контроллера, например http://127.0.0.1:8080/my_project/root:

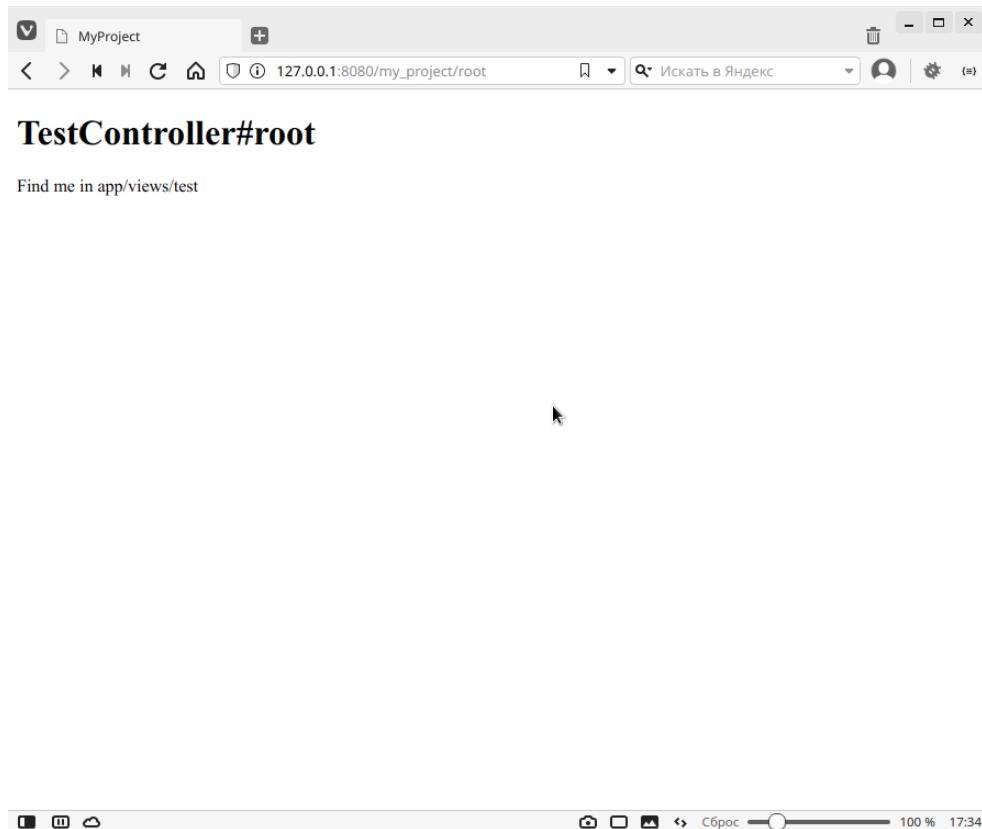


Рис. 3. Проверка функционирования метода «root» нового графического контроллера.

Для того чтобы сделать данную страницу корневой для всего приложения требуется изменить в файле маршрутов «[MyProject]/config/routes.php» (где «MyProject» - название нового проекта) строку «get("/root", ["to" => "Test#root"]);» на строку вида «root("Test", "root");» :

```
namespace FlyCubePHP;

// NOTE: Set routes here...
root("Test", "root");
get("/test", [ "to" => "Test#test" ] );
get("/test_2", [ "to" => "Test#test_2" ] );
```

Для проверки успешно заданной корневой страницы приложения требуется выполнить запрос в браузере http://127.0.0.1:8080/my_project/. Вывод не должен отличаться от предыдущего запроса, при этом доступ по адресу http://127.0.0.1:8080/my_project/root теперь должен вернуть ошибку:

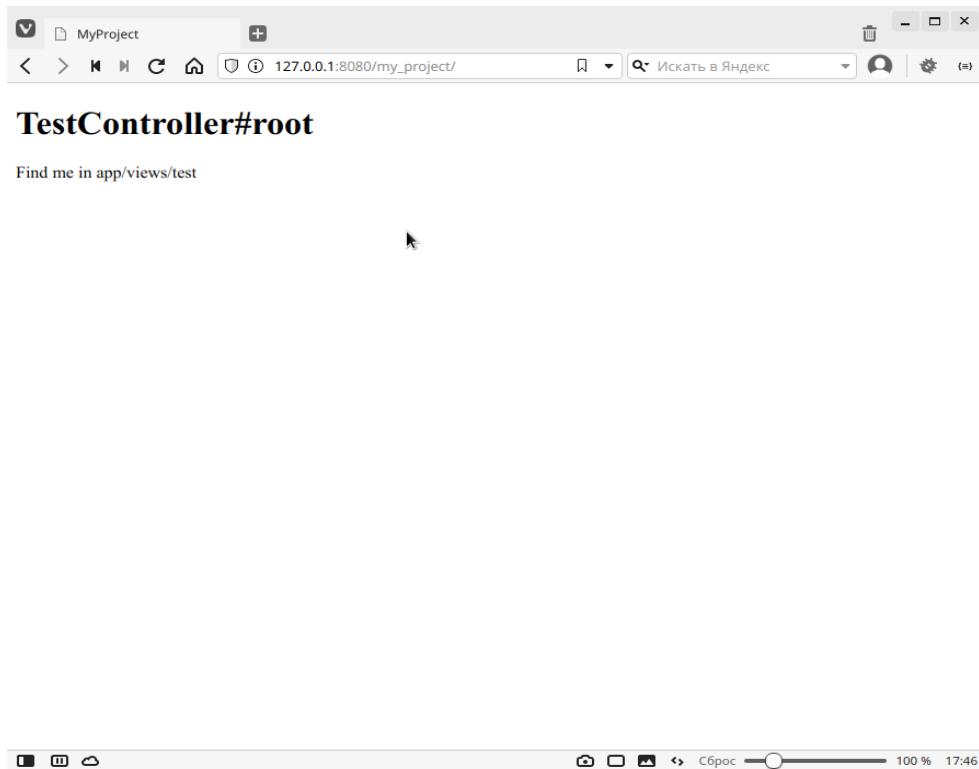


Рис. 4. Проверка функционирования корневой страницы приложения (http://127.0.0.1:8080/my_project/).

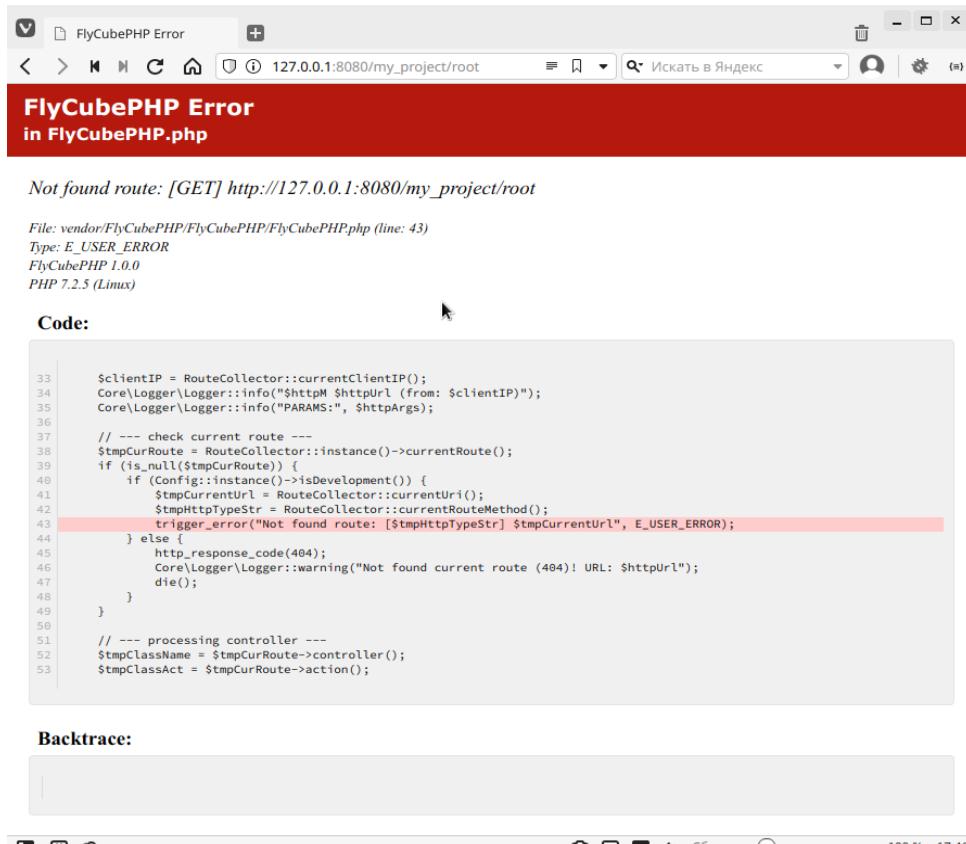


Рис. 5. Проверка вывода ошибки доступа по адресу http://127.0.0.1:8080/my_project/root

Так же проверить список доступных маршрутов приложения можно консольной командой «./fly_cube_php --routes»:

```
MyProject/bin> ./fly_cube_php --routes
== FlyCubePHP: Application routes ==
GET      /           TestController#root    as: root
GET      /test       TestController#test   as: test_test
GET      /test_2     TestController#test_2  as: test_test_2
== FlyCubePHP =====
```

ПРИМЕЧАНИЕ: в случае возникновения ошибок рекомендуется очистить каталог временных файлов приложения «[MyProject]/tmp/*» (где «MyProject» - название нового проекта).

4.5. Создание нового API контроллера

Для создания нового API контроллера требуется выполнить следующую команду генератора «./fly_cube_php --new --controller-api --name=Test --actions=test»:

```
MyProject/bin> ./fly_cube_php --new --controller-api --name=Test --actions=test
== FlyCubePHP: Create new controller ==
[Created] app/controllers/ApiTestController.php
== FlyCubePHP =====
```

ПРИМЕЧАНИЕ: указывать список методов контроллера в виде входного аргумента «--actions=test» необязательно и может быть проигнорировано при генерации.

После генерации контроллера все его методы автоматически добавляются в основной файл маршрутов приложения «[MyProject]/config/routes.php» (где «MyProject» - название нового проекта):

```
namespace FlyCubePHP;
// NOTE: Set routes here...
...
get("/api/test", [ "to" => "ApiTest#test" ]);
```

Для проверки успешной работы требуется выполнить запрос в браузере к созданному API методу нового API контроллера http://127.0.0.1:8080/my_project/api/test:

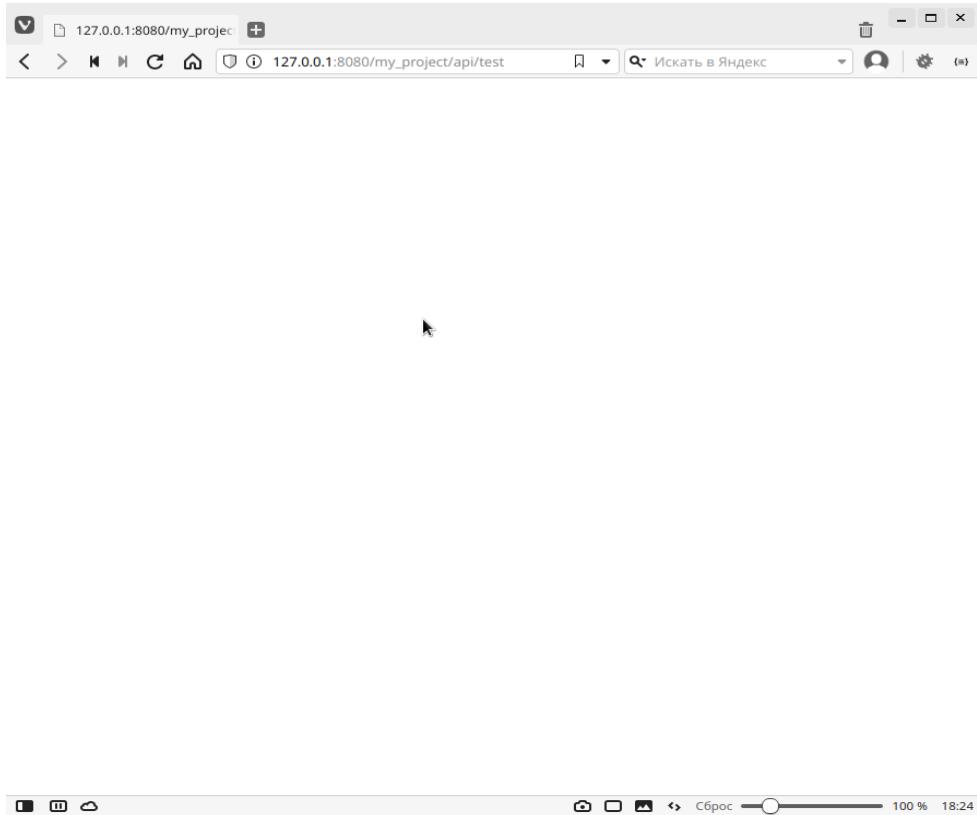


Рис. 6. Проверка функционирования тестовой API страницы приложения (http://127.0.0.1:8080/my_project/api/test).

Для проверки тестового вывода JSON данных из API метода добавим в метод контроллера «[MyProject]/app/controllers/ApiTestController::test()» (где «MyProject» - название нового проекта) следующий код:

```
<?php
class ApiTestController extends ApplicationControllerAPI
{
    public function test() {

        // Add this code:
        $this->send_response([
            'status' => 200,
            'content-type' => "application/json",
            'body' => "[{\\"uuid\\":\\"54d0e3df-13c3-43f4-80ea-d45064620197\\",\\"name\\":\\"Admin\\",\\"user_type\\":\\"admin\\",\\"auto_logout\\":15,\\"description\\":\\"Default Admin\\"},{\\"uuid\\":\\"286530bd-03bc-4fc6-b490-7d87ac5b5e44\\",\\"name\\":\\"Developer\\",\\"user_type\\":\\"developer\\",\\"auto_logout\\":15,\\"description\\":\\"Default Developer\\"},{\\"uuid\\":\\"87335d3b-845b-44d7-9823-29bd8e89f5a4\\",\\"name\\":\\"test\\",\\"user_type\\":\\"regular\\",\\"auto_logout\\":10,\\"description\\":\\"test user\\"}]";
    }
}
```

Выполнив повторный запрос в браузере к измененному API методу нового API контроллера http://127.0.0.1:8080/my_project/api/test должен быть возвращен следующий результат работы:

The screenshot shows a browser window with the URL http://127.0.0.1:8080/my_project/api/test. The page displays a JSON array of three user objects. Each object has properties: uuid, name, user_type, auto_logout, and description. The first user is an Admin, the second is a Developer, and the third is a test user.

```

1 // 20210929183354
2 // http://127.0.0.1:8080/my_project/api/test
3
4 [
5   {
6     "uuid": "54d0e3df-13c3-43f4-80ea-d45064620197",
7     "name": "Admin",
8     "user_type": "admin",
9     "auto_logout": 15,
10    "description": "Default Admin"
11  },
12  {
13    "uuid": "286530bd-03bc-4fc6-b490-7d87ac5b5e44",
14    "name": "Developer",
15    "user_type": "developer",
16    "auto_logout": 15,
17    "description": "Default Developer"
18  },
19  {
20    "uuid": "87335d3b-845b-44d7-9823-29bd8e89f5a4",
21    "name": "test",
22    "user_type": "regular",
23    "auto_logout": 10,
24    "description": "test user"
25  }
26 ]

```

Рис. 7. Проверка функционирования измененной API страницы приложения (http://127.0.0.1:8080/my_project/api/test).

5. Система маршрутизации приложения

За маршрутизацию в приложении FlyCubePHP отвечает класс ядра маршрутизации «\FlyCubePHP\Core\Routes\RouteCollector» и класс, описывающий маршрут «\FlyCubePHP\Core\Routes\Route». Методы доступа, которые поддерживает объект маршрута, полностью соответствуют требованиям REST API, и могут быть:

- HTTP GET - применяется для запроса данных ресурса;
- HTTP POST - применяется для создания новых объектов ресурса;
- HTTP PUT - применяется для обновления существующего ресурса;
- HTTP PATCH - применяется для частичного обновления существующего ресурса;
- HTTP DELETE - применяется для удаления объекта ресурса.

Основными задачами ядра маршрутизации являются:

- формирование списка маршрутов на основании объектов «Route»;
- анализ списка маршрутов на актуальность и корректность данных, содержащихся в объекте «Route» (ПРИМЕЧАНИЕ: если в объекте «Route» задано некорректное имя контроллера или его метода, маршрут будет удален из списка);
- анализ списка маршрутов и поиск необходимого, основываясь на методе HTTP запроса и URL для дальнейшей обработки;

- обработка входных HTTP параметров и данных и формирование единого массива аргументов для контроллеров.

Перечень доступных маршрутов приложения задается в файлах маршрутов «routes.php». Возможны несколько вариантов расположения этих файлов:

- в каталоге конфигурации приложения «[MyProject]/config/routes.php» - это основной файл маршрутов приложения, без которого работа всего приложения невозможна, где «MyProject» - название нового проекта;
- в каталогах конфигурации плагинов «[MyProject]/plugins/*/config/routes.php» - данные дополнительные файлы маршрутов могут только расширить список доступных маршрутов приложения, где «MyProject» - название нового проекта.

Основные методы ядра маршрутизации, доступные для использования:

- `->appendRoute(Route &$route)` - добавить объект маршрута в ядро маршрутизации (ПРИМЕЧАНИЕ: в случае добавления дубликата маршрута - он игнорируется);
- `->containsRoute(string $method, string $uri)` - проверка наличия маршрута в ядре маршрутизации, где «\$method» - HTTP метод доступа, «\$uri» - URL;
- `->containsRootRoute()` - проверка наличия корневого маршрута в ядре маршрутизации;
- `->routeByMethodUri(string $method, string $uri)` - получить объект маршрута по методу доступа «\$method» и URL «\$uri» (ПРИМЕЧАНИЕ: если необходимый маршрут не найден, возвращается «null»);
- `->routeByControllerAct(string $controller, string $action)` - получить объект маршрута по названию контроллера «\$controller» и его метода «\$action» (ПРИМЕЧАНИЕ: если необходимый маршрут не найден, возвращается «null»);
- `->rootRoute()` - получить объект корневого маршрута (ПРИМЕЧАНИЕ: если необходимый маршрут не найден, возвращается «null»);
- `->allRoutes()` - получить массив объектов, описывающих все маршруты приложения;
- `->currentRoute()` - получить объект маршрута для текущего HTTP запроса (ПРИМЕЧАНИЕ: если необходимый маршрут не найден, возвращается «null»).

Основные статические методы ядра маршрутизации, доступные для использования:

- `RouteCollector::applicationUrlPrefix()` - получить URL префикс для текущего приложения (ПРИМЕЧАНИЕ: URL префикс задается в файле конфигурации «[MyProject]/config/application_env.conf» (где «MyProject» - название нового проекта) ключом «APP_URL_PREFIX»)

```
#  
# --- application settings ---  
#  
APP_URL_PREFIX: /my_project
```

- `RouteCollector::currentHostUri()` - получить текущий URL хоста (пример: <https://127.0.0.1:8080>);

- `RouteCollector::currentClientIP()` - получить IP адрес текущего клиента;
- `RouteCollector::currentUri()` - получить текущий URL, включая входные аргументы (пример: `http://127.0.0.1:8080/my_project/?a=1&b=2`);
- `RouteCollector::currentRouteUri()` - получить текущий URL без аргументов и без URL хоста (пример: `/test`);
- `RouteCollector::currentRouteMethod()` - получить название HTTP метода для текущего запроса (GET/POST/PUT/PATCH/DELETE);
- `RouteCollector::isCurrentRouteMethodGET()` - метод для проверки является ли текущий метод запроса методом HTTP GET;
- `RouteCollector::currentRouteArgs()` - получить массив входных аргументов для текущего запроса (ПРИМЕЧАНИЕ: данный метод так же разбирает массив загружаемых на сервер файлов, и если он не пуст, то добавляет их описание в возвращаемый результат с соответствующими ключами);
- `RouteCollector::currentRouteArg(string $key, $def = null)` - получить значение входного аргумента для текущего запроса (ПРИМЕЧАНИЕ: если требуемый аргумент не задан, то возвращается базовое значение «\$def»);
- `RouteCollector::currentRouteFiles()` - получить массив входных (загруженных на сервер) файлов для текущего запроса;
- `RouteCollector::currentRouteHeaders(bool $namesToLower = false)` - получить массив HTTP заголовков текущего запроса
 - `namesToLower [bool]` - преобразовывать названия HTTP заголовков к нижнему регистру;
- `RouteCollector::currentRouteHeader(string $name)` - получить значение HTTP заголовка текущего запроса (ПРИМЕЧАНИЕ: поиск заголовка производится по его названию «\$name»; если требуемый заголовок не найден, возвращается «null»);
- `RouteCollector::makeValidUrl(string $uri)` - сформировать корректную строку адреса с добавлением URL префикса приложения, если он задан:

```
#> echo makeValidUrl("/api/test_api");
#> if url-prefix not set => "/api/test_api"
#> if url-prefix set ("/my_project") => "/my_project/api/test_api"
```

ПРИМЕЧАНИЕ: чтобы избежать ошибок маршрутизации, рекомендуется использовать данный метод для формирования адресов внутри приложения.

- `RouteCollector::spliceUrlFirst(string $uri)` - обрезать URL строку вначале (исключить начальные символы «/»)

```
#> echo spliceUrlFirst("/app1/");
#> => "/app1/"
```

- `RouteCollector::spliceUrlLast(string $uri)` - обрезать URL строку в конце (исключить конечные символы «/»)

```
#> echo spliceUrlLast("/app1/");
#> => "/app1"
```

Вспомогательные методы ядра маршрутизации, упрощающие формирование списка доступных маршрутов приложения:

- `root(string $controller, string $action)` - задать корневой маршрут приложения, где:
 - `controller [string]` - название контроллера обработчика;
 - `action [string]` - название метода контроллера, который будет обрабатывать входящий запрос;
- `get(string $uri, array $args = [])` - добавить HTTP GET маршрут, где:
 - `uri [string]` - URL доступа;
 - `args [array]` - массив аргументов для создания маршрута;
- `post(string $uri, array $args = [])` - добавить HTTP POST маршрут, где:
 - `uri [string]` - URL доступа;
 - `args [array]` - массив аргументов для создания маршрута;
- `put(string $uri, array $args = [])` - добавить HTTP PUT маршрут, где:
 - `uri [string]` - URL доступа;
 - `args [array]` - массив аргументов для создания маршрута;
- `patch(string $uri, array $args = [])` - добавить HTTP PATCH маршрут, где:
 - `uri [string]` - URL доступа;
 - `args [array]` - массив аргументов для создания маршрута;
- `delete(string $uri, array $args = [])` - добавить HTTP DELETE маршрут, где:
 - `uri [string]` - URL доступа;
 - `args [array]` - массив аргументов для создания маршрута.

Список ключей массива аргументов, необходимого для создания маршрута:

- `to [string]` - название контроллера и его метода, разделённых символом «#»;
- `controller [string]` - название контроллера обработчика;
- `action [string]` - название метода контроллера, который будет обрабатывать входящий запрос;
- `as [string]` - псевдоним маршрута для быстрого доступа к его URL (на этапе запуска автоматически формируется `define` формата «AS_url»);
- все прочие аргументы будут переданы в маршрут как статические параметры.

ПРИМЕЧАНИЕ: ключи «to» и «controller» + «action» взаимоисключающие!

Начиная с версии 1.3.0 стало возможным задавать маршруты с указанием аргументов:

- `/test?id=123` - указание статического аргумента в теле URL (все статические аргументы автоматически добавляются в массив входных аргументов текущего маршрута);
- `/test/:id` - указание динамического аргумента, который формируется при разборе входящего URL (все динамические аргументы автоматически добавляются в массив входных аргументов текущего маршрута и имеют ключи с учетом заданных в маршруте).

Пример содержимого файла маршрутов приложения «[MyProject]/config/routes.php» (где «MyProject» - название нового проекта):

```

namespace FlyCubePHP;

// NOTE: Set routes here...
root("Test", "root");
get("/test", [ "to" => "Test#test" ] );
get("/test_2", [ "controller" => "Test", "action" => "test_2" ] );
get("/test_3", [ "controller" => "Test", "action" => "test_2", "as" => "test_3" ] );

```

Так же для получения списка всех маршрутов приложения можно воспользоваться консольной командой «./fly_cube_php --routes»:

```

MyProject/bin> ./fly_cube_php --routes

==== FlyCubePHP: Application routes ====
GET      /           TestController#root    as: root
GET      /test       TestController#test   as: test_test
GET      /test_2     TestController#test_2  as: test_test_2
GET      /test_3     TestController#test_3  as: test_3
==== FlyCubePHP =====

```

Пример формирования маршрутов с указанием аргументов:

```

namespace FlyCubePHP;

// NOTE: Set routes here...
get("/test/:id", [ "to" => "Test#test" ]);
get("/error_1?code=404", [ "controller" => "Test", "action" => "error_1" ] );
get("/error_2", [ "controller" => "Test", "action" => "error_2", "code" => 404 ] );

```

Пример части входных параметров, полученных при обработке маршрутов с указанием аргументов:

```

// --- 1 ---
// get("/test/:id", [ "to" => "Test#test" ]);

#> curl http://localhost:8080/test/123

--> Test::_params['controller'] = 'Test'
    Test::_params['action'] = 'test'
    Test::_params['id'] = '123'
    ...

// --- 2 ---
// get("/error_1?code=404", [ "controller" => "Test", "action" => "error_1" ]);

#> curl http://localhost:8080/error_1

--> Test::_params['controller'] = 'Test'
    Test::_params['action'] = 'error_1'
    Test::_params['code'] = '404'
    ...

// --- 3 ---
// get("/error_2", [ "controller" => "Test", "action" => "error_2", "code" => 404 ] );

#> curl http://localhost:8080/error_2

--> Test::_params['controller'] = 'Test'
    Test::_params['action'] = 'error_2'
    Test::_params['code'] = 404
    ...

```

Пример формирования маршрутов с указанием псевдонима:

```
namespace FlyCubePHP;

// NOTE: Set routes here...
get("/test/:id", [ "to" => "Test#test", "as" => "test" ]);
get("/error_1?code=404", [ "controller" => "Test", "action" => "error_1", "as" => "error_1" ]);
get("/error_2", [ "to" => "Test#error_2", "code" => 404, "as" => "error_2" ]);
```

Пример быстрого доступа к URL маршрута по его псевдониму:

```
// --- 1 ---
// get("/test/:id", [ "to"=>"Test#test", "as"=>"test" ]);

#> var_dump(test_url); // '/test/:id'

// --- 2 ---
// get("/error_1?code=404", [ "controller"=>"Test", "action"=>"error_1", "as"=>"error_1" ]);

#> var_dump(error_1_url); // '/error_1'

// --- 3 ---
// get("/error_2", [ "to"=>"Test#error_2", "code"=>404, "as"=>"error_2" ]);

#> var_dump(error_2_url); // '/error_2'
```

где «test_url», «error_1_url», «error_2_url» - это константы, созданные с помощью метода define().

ПРИМЕЧАНИЕ: в случае возникновения ошибок рекомендуется очистить каталог временных файлов приложения «[MyProject]/tmp/*» (где «MyProject» - название нового проекта).

6. Asset Pipeline и работа с ресурсами

Asset Pipeline - класс, предоставляющий доступ к ресурсам приложения и выполняющим обработку этих ресурсов. В asset pipeline входит несколько подклассов-обработчиков, реализующих различные задачи:

- JSBuilder - производит поиск, загрузку и компиляцию *.js и *.js.php файлов с учетом заданных каталогов ресурсов; так же данный класс производит формирование дерева ресурсов с учетом указанных в файлах директив « = require ... » и « = require_tree ... »;
- CSSBuilder - производит поиск, загрузку и компиляцию *.css и *.scss файлов с учетом заданных каталогов ресурсов; так же данный класс производит формирование дерева ресурсов с учетом указанных в файлах директив « = require ... » и « = require_tree ... »;
- ImageBuilder - производит поиск и загрузку изображений; поддерживаемые форматы:
 - *.svg
 - *.png
 - *.jpg
 - *.jpeg
 - *.gif

Основные каталоги ресурсов приложения (где «MyProject» - название нового проекта):

- [MyProject]/app/assets/ - каталог ресурсов приложения;
- [MyProject]/lib/assets/ - каталог ресурсов и библиотек, разработанных в рамках проекта;
- [MyProject]/vendor/assets/ - каталог ресурсов и библиотек сторонних разработчиков;
- [MyProject]/plugins/*/app/assets/ - каталог ресурсов плагина;
- [MyProject]/plugins/*/lib/assets/ - каталог ресурсов и библиотек, разработанных в рамках плагина.

Основные методы Asset Pipeline, доступные для использования:

- ->viewDirs(bool \$fullPath) - получить список добавленных каталогов «views»; если «\$fullPath = true» - вернуть полные пути к каталогам;
- ->appendViewDir(string \$dir) - добавить путь к каталогу «views» в Asset Pipeline (ПРИМЕЧАНИЕ: при добавлении нового каталога «views» проверяется его наличие);
- ->jsDirs(bool \$fullPath) - получить список добавленных каталогов «javascripts»; если «\$fullPath = true» - вернуть полные пути к каталогам;
- ->appendJSDir(string \$dir) - добавить путь к каталогу «javascripts» в Asset Pipeline (ПРИМЕЧАНИЕ: при добавлении нового каталога «javascripts» проверяется его наличие);

- `->javascriptFilePath(string $name)` - получить путь (или массив путей) для требуемого javascript-файла (ПРИМЕЧАНИЕ: поиск осуществляется по имени файла без его расширения и указания пути, то есть для поиска файла «MyProject/app/assets/javascripts/test.js» требуется указать только «test»);
- `->cssDirs(bool $fullPath)` - получить список добавленных каталогов «stylesheets»; если «\$fullPath = true» - вернуть полные пути к каталогам;
- `->appendCSSDir(string $dir)` - добавить путь к каталогу «stylesheets» в Asset Pipeline (ПРИМЕЧАНИЕ: при добавлении нового каталога «stylesheets» проверяется его наличие);
- `->stylesheetFilePath(string $name)` - получить путь (или массив путей) для требуемого stylesheet-файла (ПРИМЕЧАНИЕ: поиск осуществляется по имени файла без его расширения и указания пути, то есть для поиска файла «MyProject/app/assets/stylesheets/test.css» требуется указать только «test»);
- `->imageDirs(bool $fullPath)` - получить список добавленных каталогов «images»; если «\$fullPath = true» - вернуть полные пути к каталогам;
- `->imageList(bool $fullPath)` - получить список загруженных изображений и пути к ним; если «\$fullPath = true» - вернуть полные пути к файлам;
- `->imageFilePath(string $name)` - получить путь для требуемого image-файла (ПРИМЕЧАНИЕ: поиск осуществляется по имени файла без указания пути, то есть для поиска файла «MyProject/app/assets/images/test.png» требуется указать только «test.png»);
- `->appendImageDir(string $dir)` - добавить путь к каталогу «images» в Asset Pipeline (ПРИМЕЧАНИЕ: при добавлении нового каталога «images» проверяется его наличие).

6.1. Работа с *.js и *.js.php файлами

Всей работой по загрузке, обработке и компиляции javascript файлов занимается подкласс Asset Pipeline-а «\FlyCubePHP\Core\AssetPipeline\JSBuilder». На текущий момент JSBuilder позволяет обрабатывать следующие типы файлов:

- `*.js` - стандартный JavaScript файл;
- `*.js.php` - стандартный JavaScript файл, содержащий вставки PHP кода, который предварительно требуется обработать и результат выполнения подставить в исходный файл.

Пример использования PHP вставок кода в скриптах с расширением *.js.php:

```
/***
 * = require application_page
 */

// append namespace in CoreApplication
(function() {
    this.CoreApplication || (this.CoreApplication = {});

    /**
     * @brief Namespace по работе с базовыми методами.
     */
    CoreApplication.Default_2 = {
        /**
         * @brief Массив базовых настроек приложения
         */
        SETTINGS: {
            "application-url-prefix": "<?php echo \FlyCubePHP\Core\Config\Config::TAG_APP_URL_PREFIX; ?>",
            "application-current-act-name": "",
            "application-current-act-help-anchor": "",
            <?php
            if (\FlyCubePHP\Core\Config\Config::TAG_APP_URL_PREFIX === "APP_URL_PREFIX") {
                echo "\\"--11--\\": \"true\",\\n";
                echo "\\"--22--\\": \"123\",\\n";
            }
            ?>
            "application-support-help": "false",
            "application-support-auth": "true",
            "application-auth-plugin-name": "qwerty",
            <?php if (\FlyCubePHP\Core\Config\Config::TAG_APP_URL_PREFIX === "APP_URL_PREFIX") {
                echo "\\"--123--\\": \"value\\n";
            }
            ?>
        }
    }
}).call(this);
```

Результат компиляции *.js.php файла в *.js файл:

```
/***
 * = require application_page
 */

// append namespace in CoreApplication
(function() {
    this.CoreApplication || (this.CoreApplication = {});

    /**
     * @brief Namespace по работе с базовыми методами.
     */
    CoreApplication.Default_2 = {
        /**
         * @brief Массив базовых настроек приложения
         */
        SETTINGS: {
            "application-url-prefix": "APP_URL_PREFIX",
            "application-current-act-name": "",
            "application-current-act-help-anchor": "",
            "--11--": "true",
            "--22--": "123",
            "application-support-help": "false",
            "application-support-auth": "true",
            "application-auth-plugin-name": "qwerty",
            "--123--": "value"
        }
    }
}).call(this);
```

ПРИМЕЧАНИЕ: для того чтобы JSBuilder добавил в *.js файл результаты выполнения вставок PHP кода, рекомендуется использовать стандартную функцию «echo». Все вставки PHP кода будут заменены на результаты их выполнения. В случае ошибок компиляции будет сгенерирована ошибка.

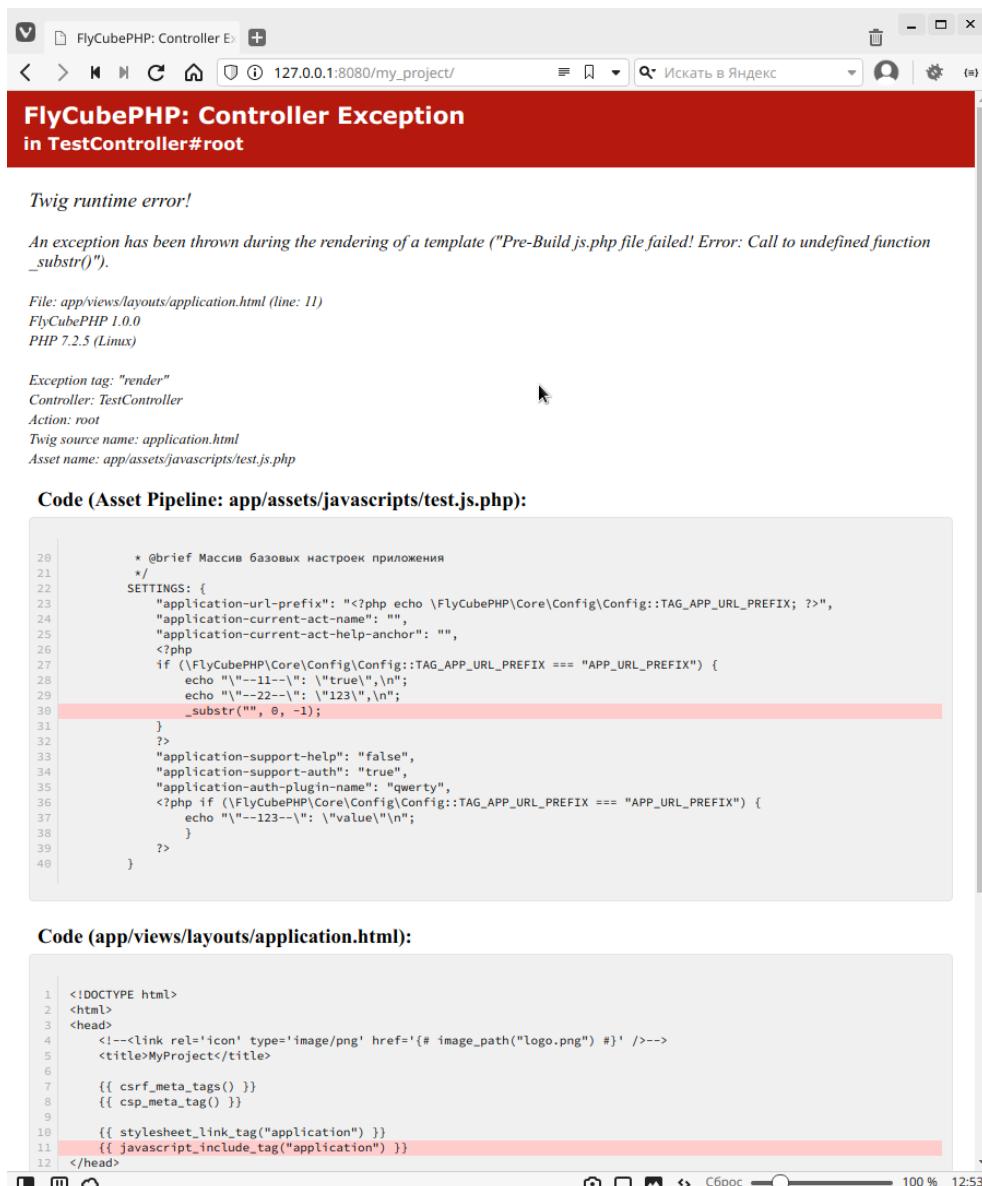


Рис. 8. Страница вывода ошибки при компиляции *.js.php файла.

Так же JSBuilder поддерживает специальные директивы зависимостей, позволяющие указать зависимые JavaScript файлы или дерево каталогов со всеми JavaScript файлами.

Поддерживаемые директивы:

- = require [FILE NAME] - указание зависимого JavaScript файла;
- = require_tree [DIR PATH] - указание зависимого каталога со всеми JavaScript файлами.

ПРИМЕЧАНИЕ: директивы зависимостей должны указываться в блоках комментариев вначале файла.

При загрузке файлов анализируются все скрипты на наличие данных директив и формируется дерево зависимостей. Если на этапе разбора файлов JSBuilder не находит требуемый зависимый файл, то генерируется исключение и выполнение приложения завершается.

Пример использования директив в файле «[MyProject]/app/assets/javascripts/application.js»:

```
//  
// Created by FlyCubePHP generator.  
// User: test  
// Date: 29.09.21  
// Time: 14:39  
//  
//  
// This is a manifest file that'll be compiled into application.js, which will include all the files  
// listed below.  
//  
// Any JavaScript/JavaScript.PHP file within this directory, lib/assets/javascripts, or any plugin's  
// vendor/assets/javascripts directory can be referenced here using a relative path.  
//  
// Supported require_* commands:  
// require [name]      - load file (search by name without extension)  
// require_tree [path] - load all files from path  
//  
//= require fly-cube-php-ujs  
//= require_tree .
```

где:

- `//= require fly-cube-php-ujs` - указание JSBuilder-у загрузить файл «fly-cube-php-ujs» перед текущим файлом «application.js»;
- `//= require_tree .` - указание JSBuilder-у загрузить все содержимое каталога «[MyProject]/app/assets/javascripts/» перед текущим файлом «application.js».

С версии FlyCubePHP 1.4.0 стало доступным указывать имена зависимостей как без расширения файла так и с ним:

```
//  
//= require fly-cube-php-ujs  
//= require jquery-3.6.0.min  
//= require bootstrap-5.1.3.min  
//
```

или

```
//  
//= require fly-cube-php-ujs.js  
//= require jquery-3.6.0.min.js  
//= require bootstrap-5.1.3.min.js  
//
```

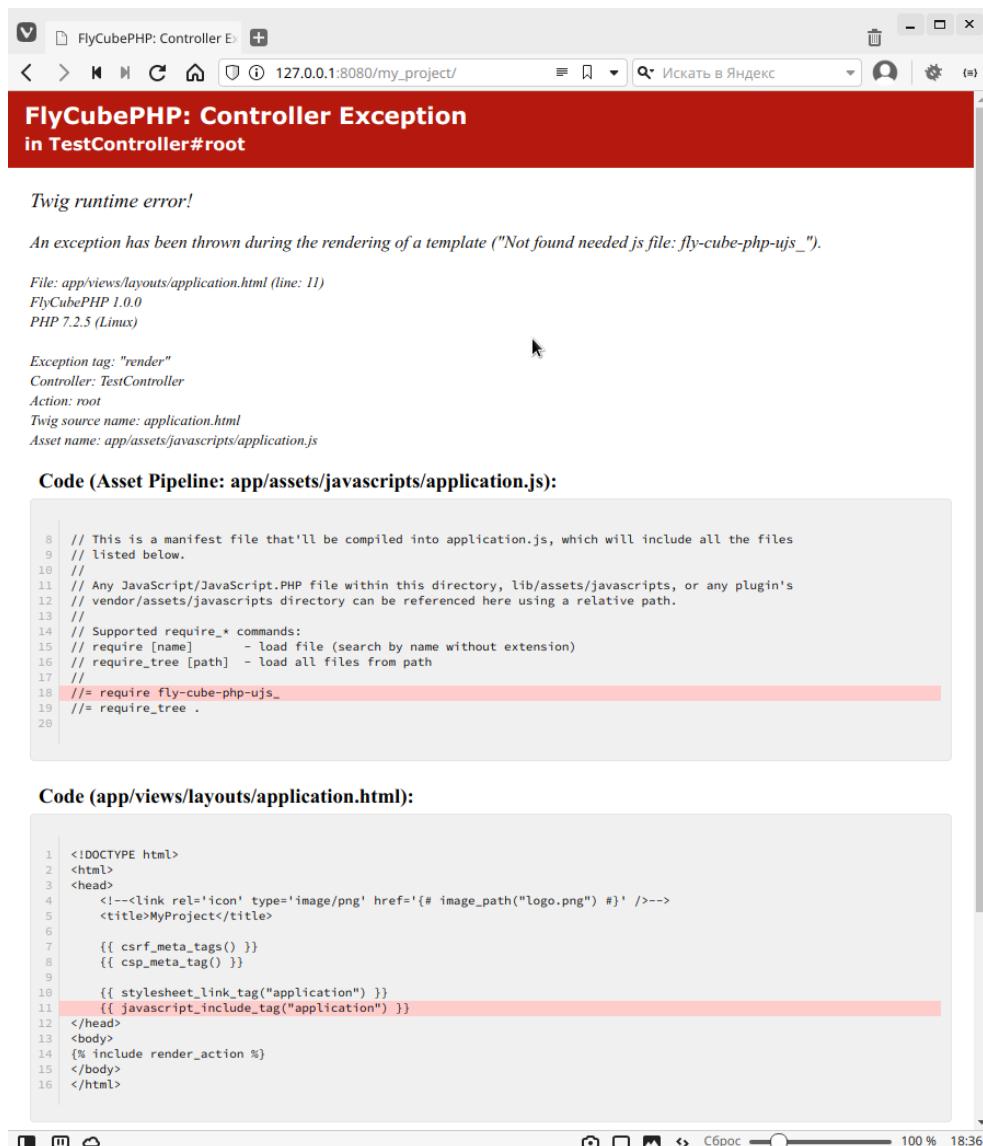


Рис. 9. Страница вывода ошибки о некорректном зависимом JavaScript файле.

Можно еще выделить различия в поведении JSBuilder при различных режимах работы приложения:

- `development mode` - JSBuilder производит только разбор зависимостей скриптов с учетом указанных директив `<?= require ... ?>` и `<?= require_tree ... ?>`; результат работы - массив скриптов, отсортированный по очереди их загрузке на клиенте;
- `production mode` - JSBuilder производит разбор зависимостей скриптов с учетом указанных директив `<?= require ... ?>` и `<?= require_tree ... ?>` и сборку всех файлов в единый скрипт с последующей компиляцией его в `*.min.js` файл; результат работы - скомпилированный `*.min.js` файл.

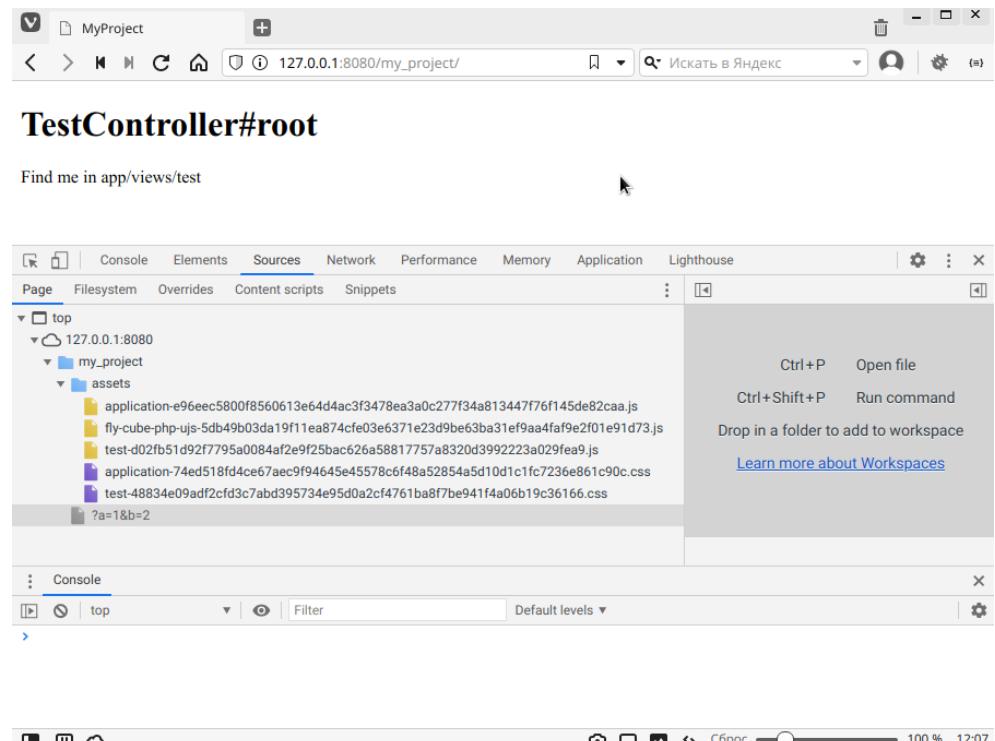


Рис. 10. Дерево ресурсов приложения в «development mode».

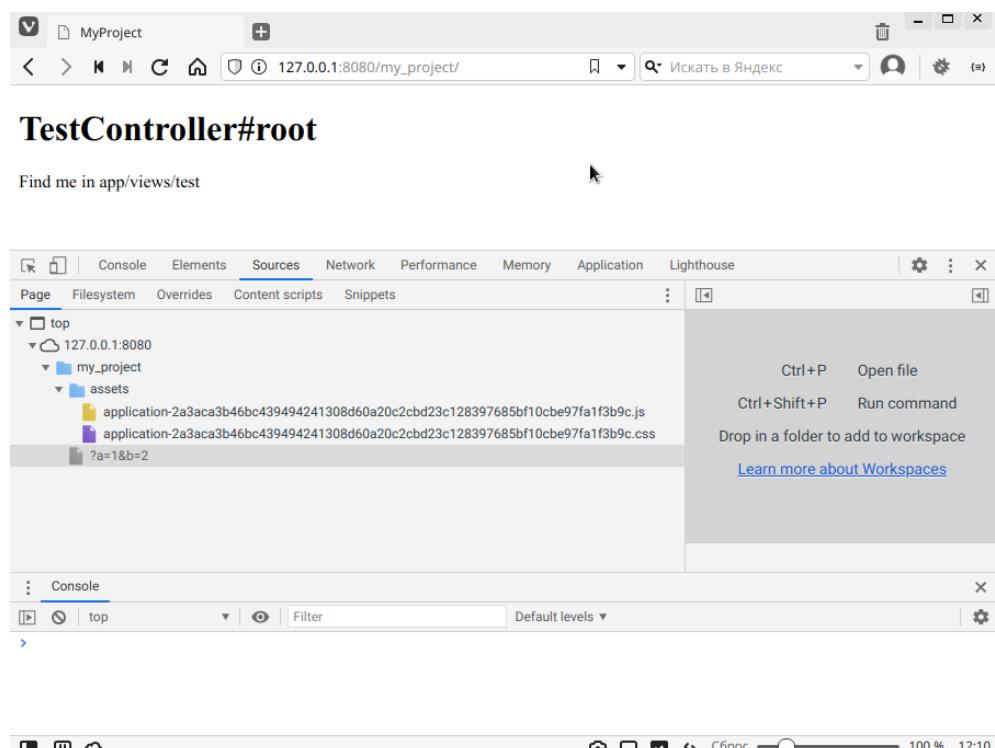


Рис. 11. Дерево ресурсов приложения в «production mode».

6.2. Работа с *.css и *.scss файлами

Всей работой по загрузке, обработке и компиляции stylesheets файлов занимается подкласс Asset Pipeline-а «\FlyCubePHP\Core\AssetPipeline\CSSBuilder». На текущий момент CSSBuilder позволяет обрабатывать следующие типы файлов:

- *.css - стандартный CSS Stylesheets файл;
- *.scss - стандартный Sass Stylesheets файл.

Так же CSSBuilder поддерживает специальные директивы зависимостей, позволяющие указать зависимые Stylesheets файлы или дерево каталогов со всеми Stylesheets файлами.

Поддерживаемые директивы:

- = require [FILE NAME] - указание зависимого Stylesheets файла;
- = require_tree [DIR PATH] - указание зависимого каталога со всеми Stylesheets файлами.

ПРИМЕЧАНИЕ: директивы зависимостей должны указываться в блоках комментариев вначале файла.

При загрузке файлов анализируются все скрипты на наличие данных директив и формируется дерево зависимостей. Если на этапе разбора файлов CSSBuilder не находит требуемый зависимый файл, то генерируется исключение и выполнение приложения завершается.

Пример использования директив в файле «[MyProject]/app/assets/stylesheets/application.css»:

```
/*
 * Created by FlyCubePHP generator.
 * User: test
 * Date: 29.09.21
 * Time: 14:39
 *
 *
 * This is a manifest file that'll be compiled into application.css, which will include all the files
 * listed below.
 *
 * Any CSS and SCSS file within this directory, lib/assets/stylesheets, or any plugin's
 * vendor/assets/stylesheets directory can be referenced here using a relative path.
 *
 * Supported require_* commands:
 *
 * require [name]      - load file (search by name without extension)
 * require_tree [path] - load all files from path
 *
 *= require_tree .
 */
```

где:

- = require_tree . - указание CSSBuilder-у загрузить все содержимое каталога «[MyProject]/app/assets/stylesheets/» перед текущим файлом «application.css».

С версии FlyCubePHP 1.4.0 стало доступным указывать имена зависимостей как без расширения файла так и с ним:

```
/*
*= require bootstrap-5.1.3.min
*/
```

ИЛИ

```
/*
*= require bootstrap-5.1.3.min.css
*/
```

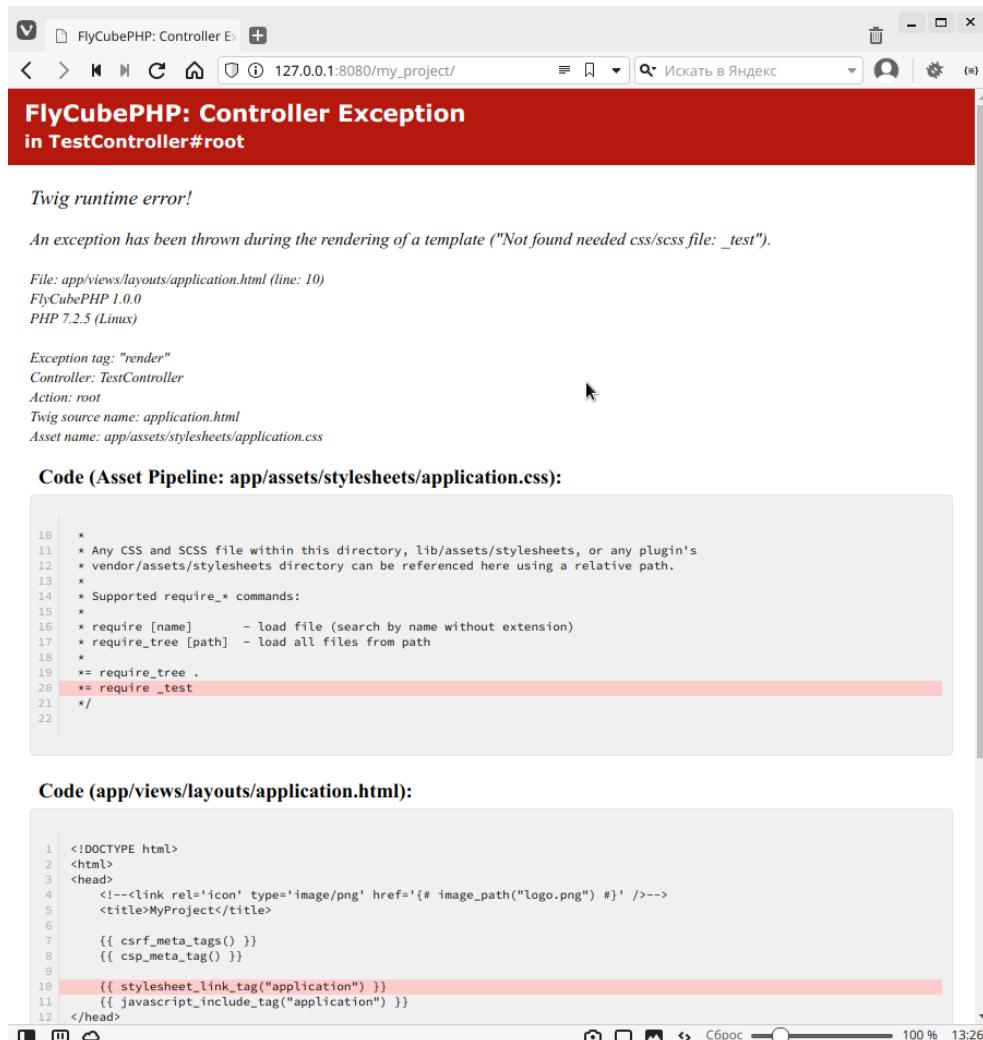


Рис. 12. Страница вывода ошибки о некорректном зависимом Stylesheets файле.

Для Sass Stylecheets файлов можно использовать свою директиву указания зависимых файлов «@import [FILE NAME]». Если на этапе разбора и копиляции Sass Stylesheets файлов не найдены требуемые зависимые файлы, то генерируется исключение и выполнение приложения завершается.

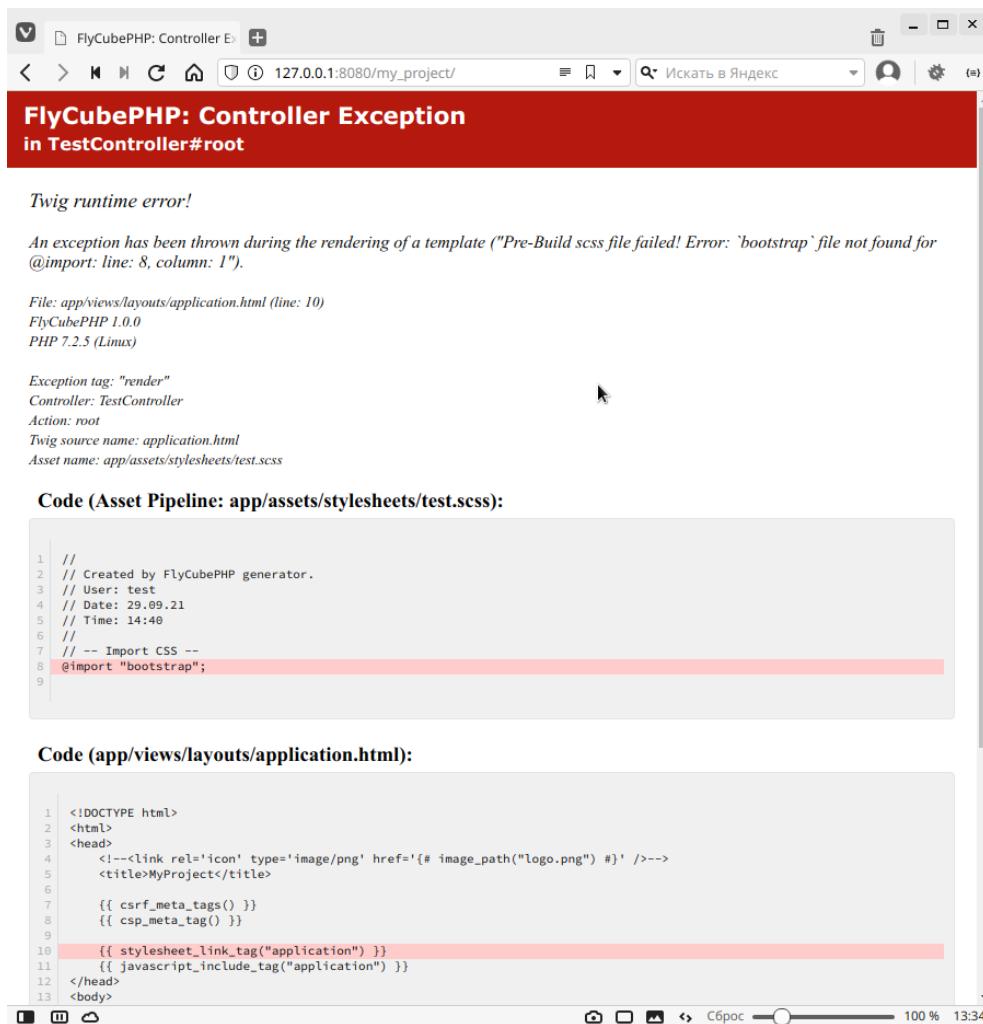


Рис. 13. Страница вывода ошибки о некорректном зависимом Sass Stylesheets файле.

Можно еще выделить различия в поведении CSSBuilder при различных режимах работы приложения:

- `development mode` - CSSBuilder производит только разбор зависимостей скриптов с учетом указанных директив `<?= require ... ?>` и `<?= require_tree ... ?>`, а также сборку Sass Stylesheets файлов; результат работы - массив скриптов, отсортированный по очереди их загрузке на клиенте;
- `production mode` - CSSBuilder производит разбор зависимостей скриптов с учетом указанных директив `<?= require ... ?>` и `<?= require_tree ... ?>` и сборку всех файлов в единый скрипт с последующей компиляцией его в `*.css` файл; результат работы - скомпилированный `*.css` файл.

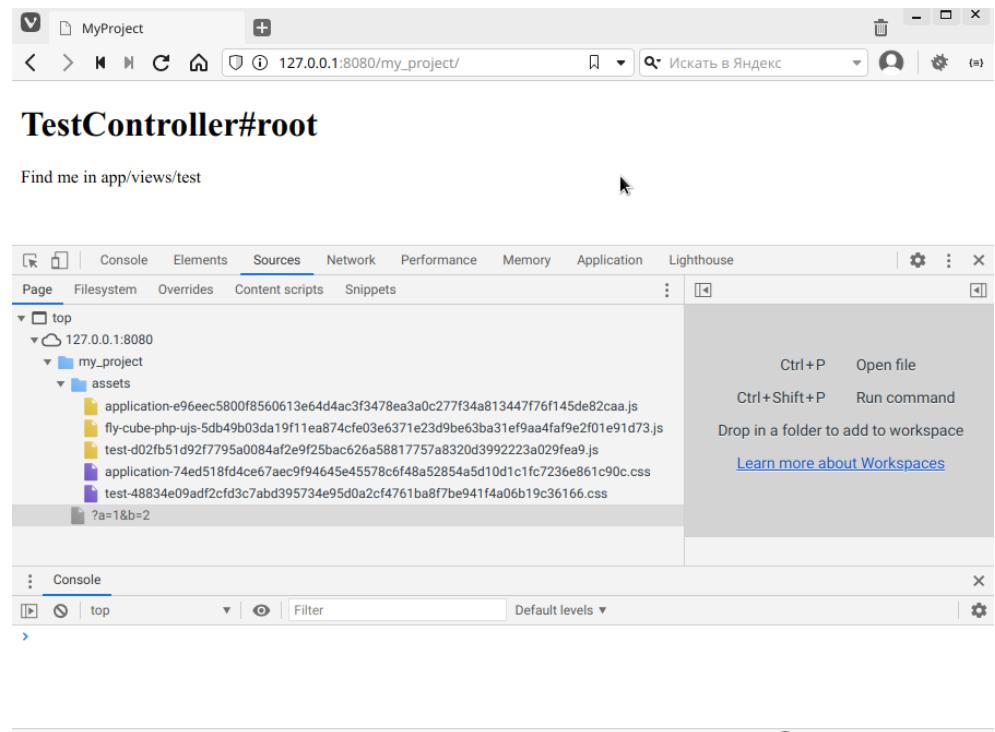


Рис. 14. Дерево ресурсов приложения в «development mode».

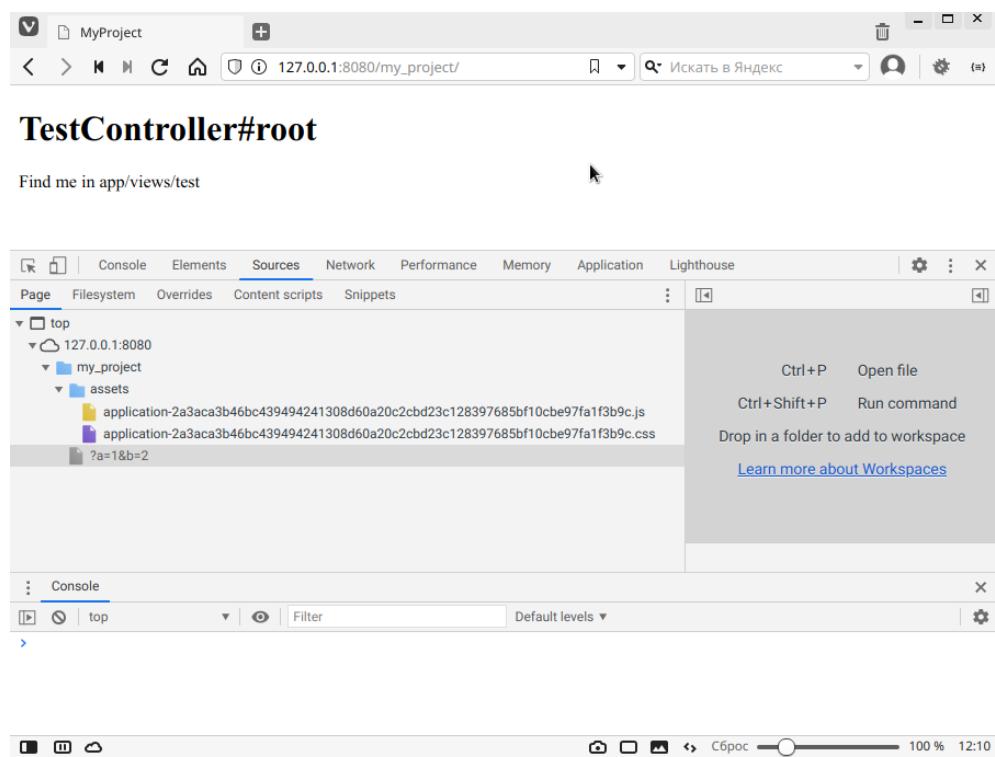


Рис. 15. Дерево ресурсов приложения в «production mode».

6.2.1. Вспомогательные функции системы компиляции Sass Stylesheets

Для более удобного написания Sass Stylesheets скриптов и их компиляции в CSS Stylesheets скрипты CSSBuilder предоставляет следующий перечень вспомогательных функций:

- `asset_path(string $path)` - получить путь до необходимого ресурса;
ПРИМЕЧАНИЕ: если необходимый ресурс не был найден, то генерируется исключение и работа приложения завершается.
- `asset_url(string $path)` - получить путь до необходимого ресурса в формате «`url(PATH)`»;
ПРИМЕЧАНИЕ: если необходимый ресурс не был найден, то генерируется исключение и работа приложения завершается.

ПРИМЕЧАНИЕ: на момент релиза FlyCubePHP 1.4.0 методы «`asset_path`» и «`asset_url`» производят поиск ресурса только среди изображений.

Пример использования данных вспомогательных функций в `*.scss` файле:

```
i.api-deprecated {  
background: url(asset_path('deprecated.svg')) no-repeat;  
width: 16px;  
height: 16px;  
padding-left: 16px;  
padding-right: 2px;  
padding-top: 1px;  
}  
i.api-required {  
background: asset_url('required.svg') no-repeat;  
width: 16px;  
height: 16px;  
padding-left: 16px;  
padding-right: 2px;  
margin-bottom: -4px;  
display: inline-block;  
}
```

6.3. Работа с изображениями

Всей работой по загрузке и поиску изображений занимается подкласс Asset Pipeline-а «\FlyCubePHP\Core\AssetPipeline\ImageBuilder». На текущий момент ImageBuilder позволяет обрабатывать следующие типы файлов:

- *.svg
- *.png
- *.jpg
- *.jpeg
- *.gif

Доступ к ресурсам изображений и их поиск осуществляется средствами базовых методов Asset Pipeline-а, которые были описаны выше, или средствами методов вспомогательных классов представлений (views) и CSSBuilder-а.

Краткий перечень методов Asset Pipeline-а:

- ->imageDirs(bool \$fullPath)
- ->imageList(bool \$fullPath)
- ->imageFilePath(string \$name)
- ->appendImageDir(string \$dir)

7. FlashMessages - передача данных между запросами

FlashMessages - это специальная часть сессии, которая очищается после доступа к ней во время нового запроса, в отличии от объекта Flash в Ruby on Rails, где очистка происходит при каждом запросе. Объект FlashMessages реализован классом «\FlyCubePHP\Core\Controllers\FlashMessages». Внутри объекта, все его данные представлены в форме массива ключ-значение, а доступ к ним осуществляется посредством следующих методов:

- FlashMessages::alert() - статический метод получения сообщения типа «alert»;
- FlashMessages::setAlert(string \$val) - статический метод установки сообщения типа «alert»;
- FlashMessages::notice() - статический метод получения сообщения типа «notice»;
- FlashMessages::setNotice(string \$val) - статический метод установки сообщения типа «notice»;
- ->containsKey(string \$key) - содержится ли в массиве искомый ключ;
- ->keys() - получить список всех ключей, хранимых во FlashMessages;
- ->allValues() - получить массив данных ключ-значение, хранимых во FlashMessages;
- ->value(string \$key, \$def) - получить значение ключа, где «\$def» - базовое значение, возвращаемое при отсутствии искомого;
- ->setValue(string \$key, \$value) - задать (заменить) значение массива FlashMessages;
- ->removeValue(string \$key) - удалить ключ и его значение из массива FlashMessages;
- ->clearAll() - удалить все объекты массива FlashMessages.

Пример использования FlashMessages в методах контроллера:

```
class AppCoreController extends ApplicationController
{
    public function test() {
        // Set flash message 'notice'.
        \FlyCubePHP\Core\Controllers\FlashMessages::setNotice("hi, this is flash notice");

        $this->render();
    }

    public function test_2() {
        // Get flash message 'notice'.
        var_dump(\FlyCubePHP\Core\Controllers\FlashMessages::notice());

        $this->render();
    }
}
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/FlashMessages.php».

8. Работа с cookie

Как и многие web-фреймворки или языки, разработанные (или адаптированные) для создания web-приложений, FlyCubePHP предоставляет разработчику инструменты для работы с Cookie. Как и Ruby on Rails, FlyCubePHP позволяет работать с тремя различными вариантами Cookie:

- стандартные cookie;
- подписанные cookie (signed cookie);
- шифрованные cookie (encrypted cookie).

Объект, реализующий работу с Cookie, представлен классом «\FlyCubePHP\Core\Cookie\Cookie». Данный класс предоставляет следующий набор методов:

- `->containsCookie(string $key)` - содержится ли в массиве необходимый объект cookie;
- `->cookie(string $key, $def)` - получить необходимый объект стандартного cookie по ключу, где «\$def» - базовое значение, возвращаемое при отсутствии искомого;
- `->setCookie(string $key, array $options)` - задать стандартный cookie, где «\$key» - ключ, «\$options» - массив дополнительных параметров, значения которых могут быть:
 - `value` - значение cookie;
 - `expires [int]` - время жизни cookie (`default: 1 day`);
 - `path [string]` - задать путь хранения cookie (`default: «»`);
 - `domain [string]` - задать домен cookie (`default: «»`);
 - `secure [bool]` - задать использование только HTTPS для cookie (`default: false`);
 - `httponly [bool]` - задать использование HTTP-Only флаг для cookie (`default: false`);
- `->signedCookie(string $key, $def)` - получить необходимый объект подписанного cookie по ключу, где «\$def» - базовое значение, возвращаемое при отсутствии искомого;
- `->setSignedCookie(string $key, array $options)` - задать подписанный cookie, где «\$key» - ключ, «\$options» - массив дополнительных параметров, значения которых могут быть:
 - `value` - значение cookie;
 - `expires [int]` - время жизни cookie (`default: 1 day`);
 - `path [string]` - задать путь хранения cookie (`default: «»`);
 - `domain [string]` - задать домен cookie (`default: «»`);
 - `secure [bool]` - задать использование только HTTPS для cookie (`default: false`);
 - `httponly [bool]` - задать использование HTTP-Only флаг для cookie (`default: false`);
- `->encryptedCookie(string $key, $def)` - получить необходимый объект шифрованного cookie по ключу, где «\$def» - базовое значение, возвращаемое при отсутствии искомого;

- `->setEncryptedCookie(string $key, array $options)` - задать шифрованный cookie, где «\$key» - ключ, «\$options» - массив дополнительных параметров, значения которого могут быть:
 - `value` - значение cookie;
 - `expires [int]` - время жизни cookie (`default: 1 day`);
 - `path [string]` - задать путь хранения cookie (`default: «»`);
 - `domain [string]` - задать домен cookie (`default: «»`);
 - `secure [bool]` - задать использование только HTTPS для cookie (`default: false`);
 - `httponly [bool]` - задать использование HTTP-Only флаг для cookie (`default: false`);
- `->removeCookie(string $key)` - удалить объект cookie по ключу; если объект cookie не найден - действие игнорируется;
- `->clearAll()` - удалить все объекты cookie.

ПРИМЕЧАНИЕ: методы «`removeCookie(...)`» и «`clearAll()`» применимы ко всем видам cookie.

ПРИМЕЧАНИЕ: методы по работе с cookie, такие как «`setCookie(...)`», «`setSignedCookie(...)`», «`setEncryptedCookie(...)`» и «`removeCookie(...)`» позволяют выполнить выбранное действие, если название cookie не равно «`session_name()`».

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «`FlyCubePHP/src/Core/Cookie/Cookie.php`».

8.1. Стандартные cookie

Значения стандартных cookie не изменяются ядром FlyCubePHP, и при передаче клиенту представляются в классическом виде. Пример работы со стандартными cookie приводится ниже:

```
class AppCoreController extends ApplicationController
{
    public function test() {
        // Set cookie data.
        \FlyCubePHP\Core\Cookie\Cookie::instance()->setCookie('my_cookie', [ "value" => 123,
        "httponly" => true ]);

        $this->render();
    }

    public function test_2() {
        // Get cookie data.
        var_dump(\FlyCubePHP\Core\Cookie\Cookie::instance()->cookie('my_cookie'));

        $this->render();
    }
}
```

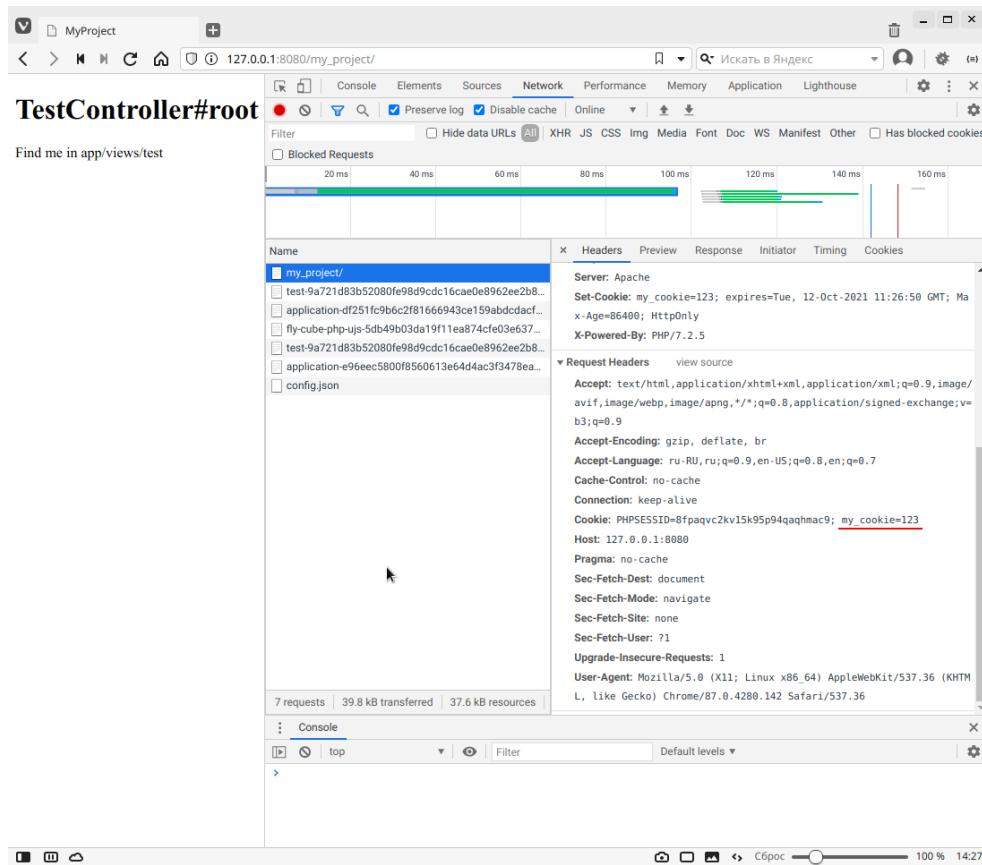


Рис. 16. Значение cookie в разделе «Request Headers».

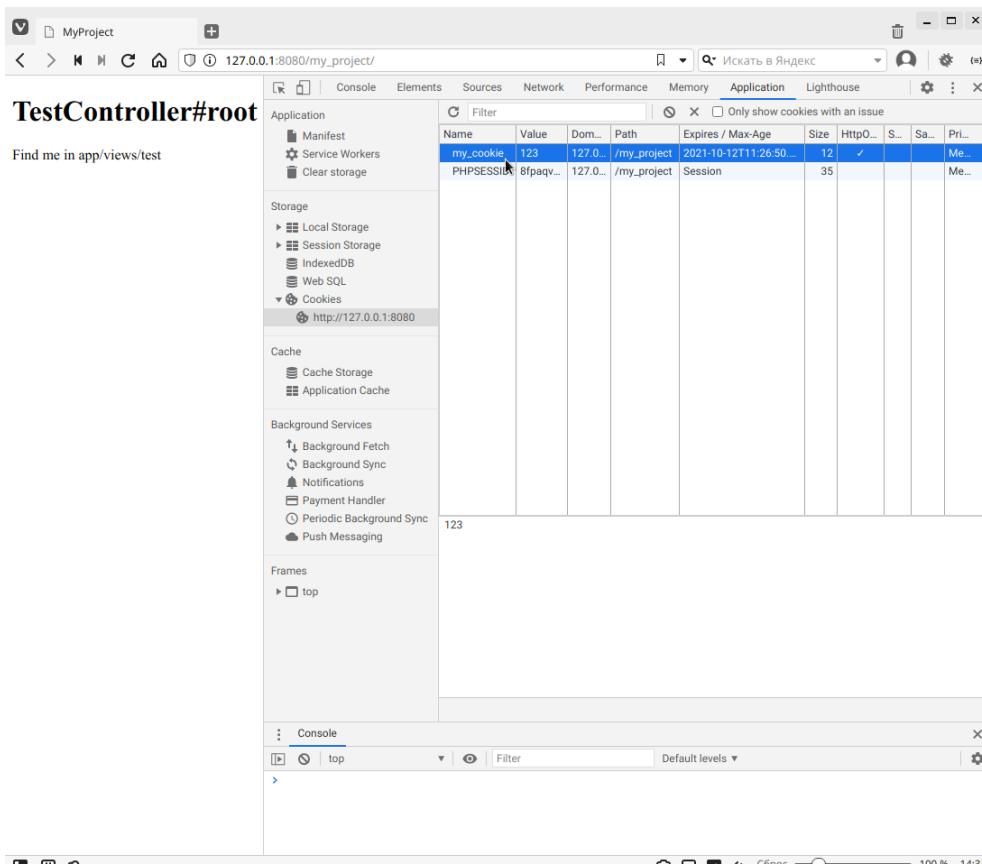


Рис. 17. Значение cookie в разделе «Application -> Storage -> Cookies».

8.2. Подписанные cookie (signed cookie)

Значения подписанных cookie представляют из себя данные, упакованные в JSON формат с указанием контрольной суммы. При получении таких cookie от клиента, ядро FlyCubePHP производит анализ контрольной суммы, и если полученное значение отличается от рассчитанного, данные cookie отбрасываются.

Пример подписанных cookie с данными «signed data 123»:

```
eyJfZmx5X2N1YmVfcGhwIjp7Im1lc3NhZ2UiOiJjMmxuYm1Wa0lHUmhkR0VnTVRJeisImV4cCI6MTYzNDAz0Dg5MywicHVyI  
joiY29va2llLm15X2Nvb2tpZV9zIn19--8ee86aae1d03f29fc1455884b04b0f39371110c6
```

где:

- `eyJfZmx...9zIn19` – тело cookie с данными;
- `8ee86aa...1110c6` – контрольная сумма.

Пример работы с подписанными cookie приводится ниже:

```
class AppCoreController extends ApplicationController  
{  
    public function test() {  
        // Set cookie data.  
        \FlyCubePHP\Core\Cookie\Cookie::instance()->setSignedCookie('my_cookie_s', [ "value" =>  
"signed data 123", "httponly" => true ]);  
  
        $this->render();  
    }  
  
    public function test_2() {  
        // Get cookie data.  
        var_dump(\FlyCubePHP\Core\Cookie\Cookie::instance()->signedCookie('my_cookie_s'));  
  
        $this->render();  
    }  
}
```

The screenshot shows the Network tab in the Chrome DevTools Network panel. A request for 'my_project/' is selected. In the Response Headers section, a cookie named 'my_cookie_s' is shown with its value highlighted in red. The cookie's value is a long string of characters starting with 'eyJfZmx5X2N1YmVfcGhwIjp7Im1lc3NhZ2UiOiJjMmxuYm1Wa0lHUmhkR0VnTVRJeisImV4cCl6MTYzNDAzODg5...'. Below the cookie, the 'Request Headers' section is visible.

Рис. 18. Значение подписанных cookie в разделе «Request Headers».

The screenshot shows the Application > Storage > Cookies tab in the Chrome DevTools Application panel. A table lists cookies for the domain 'http://127.0.0.1:8080/'. One cookie, 'my_cookie_s', is selected and its value is displayed in the bottom pane: 'eyJfZmx5X2N1YmVfcGhwIjp7Im1lc3NhZ2UiOiJjMmxuYm1Wa0lHUmhkR0VnTVRJeisImV4cCl6MTYzNDAzODg5...'. Other columns in the table include Name, Value, D_, Path, Expires / Max-Age, Size, HttpOnly, SameSite, and Priority.

Рис. 19. Значение подписанных cookie в разделе «Application -> Storage -> Cookies».

8.3. Шифрованные cookie (encrypted cookie)

Значения шифрованных cookie представляют из себя данные, упакованные в JSON формат и зашифрованные секретным ключом приложения. При получении таких cookie от клиента, ядро FlyCubePHP производит попытку расшифровки, и если процесс разшифровки проваливается, данные cookie отбрасываются.

Пример шифрованных cookie с данными «encrypted data 123»:

```
T0RPVVhJdDY0dFNQL2llMnlaN05Ec0NWc0xrZzd6RjVWMHB5ZEdvS2ZybGRST3Fmekh1NU54ZmFCbDFIdlZRenMxejM5U2ZtR
ThLZ2VLRG1kTnJ5Um1uWERzcnIrL1VNVG5ZZzRSbDliV1pCblRBVnhJ0EQzaWVCZmRkaGQzY296djFUaE9BRkxIaThlQnR0Vl
BheDRiMTZQQyt1aHU0WFpxM05aK2R0dktGMnpKSE9wQnN1Y0E9PQ%253D%253D--US1TcUh1PEiV9ltK--
6tkSFbxmup%252FxNZX2nMtB7g%253D%253D
```

где:

- `T0RPVVh...D%253D` - тело cookie с данными;
- `US1TcUh1PEiV9ltK` - вектор инициализации со случайной последовательностью данных;
- `6tkSFbx...D%253D` - тег аутентификации в режиме шифрования AEAD.

Пример работы с шифрованными cookie приводится ниже:

```
class AppCoreController extends ApplicationController
{
    public function test() {
        // Set cookie data.
        \FlyCubePHP\Core\Cookie\Cookie::instance()->setEncryptedCookie('my_cookie_e', [ "value"
=> "encrypted data 123", "httponly" => true ]);

        $this->render();
    }

    public function test_2() {
        // Get cookie data.
        var_dump(\FlyCubePHP\Core\Cookie\Cookie::instance()->encryptedCookie('my_cookie_e'));

        $this->render();
    }
}
```

The screenshot shows the Network tab of the Chrome DevTools. A request to 'my_project/' is selected. In the Headers section, there is a cookie named 'my_cookie_e' with a very long, encoded value. This value is highlighted with a red box.

Name	Value
my_cookie_e	T0RPVVhJdDYo... (long encoded value)

Рис. 20. Значение шифрованных cookie в разделе «Request Headers».

The screenshot shows the Application > Storage > Cookies tab in the Chrome DevTools. It lists the cookie 'my_cookie_e' with its details: Name, Value, Domain, Path, Expires / Max-Age, Size, HttpOnly, SameSite, and Priority. The Value column shows the same long, encoded string as in the Network tab.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	SameSite	Priority
my_cookie_e	T0RPVVhJdDYo... (long encoded value)	127.0.0.1:8080	/my_project	2021-10-12T11:56:18.3...	323	✓		Medium

Рис. 21. Значение шифрованных cookie в разделе «Application -> Storage -> Cookies».

9. Работа с сессиями

Сессии предназначены для хранения различных данных во время активного сеанса клиента. PHP предоставляет широкий набор методов по работе с сессиями и глобальный хэш-массив `$_SESSION` для доступа к данным. Для работы с сессиями FlyCubePHP предлагает использовать класс-обертку «\FlyCubePHP\Core\Session\Session». Данный класс предоставляет следующие методы:

- `->isInit()` - получить результат инициализации объекта сессии;
- `->containsKey(string $key)` - содержит ли необходимый ключ в массиве объектов сессии;
- `->keys()` - получить массив всех ключей массива сессии;
- `->value(string $key, $def)` - получить значение ключа, где «\$key» - требуемый ключ объекта, «\$def» - базовое значение, возвращаемое при отсутствии искомого;
- `->setValue(string $key, $value)` - задать (или заменить) новое значение массива сессии;
- `->removeValue(string $key)` - удалить ключ и его значение из массива сессии;
ПРИМЕЧАНИЕ: если требуемый ключ не содержится в массиве, то операция игнорируется.
- `->clearAll()` - удалить все значения и их ключи из массива сессии.

Пример работы с сессиями приводится ниже:

```
class AppCoreController extends ApplicationController
{
    public function test() {
        // Set session data.
        \FlyCubePHP\Core\Session\Session::instance()->setValue('my_session_key', 123);

        $this->render();
    }

    public function test_2() {
        // Get session data.
        var_dump(\FlyCubePHP\Core\Session\Session::instance()->value('my_session_key'));

        $this->render();
    }
}
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Session/Session.php».

10. Работа с базами данных

Вся работа с базами данных во FlyCubePHP построена на базе PHP PDO. Ядро системы по работе с базами данных реализовано в классе «\FlyCubePHP\Core\Database\DatabaseFactory» и занимается разбором конфигураций и созданием соединений. FlyCubePHP поддерживает следующие СУБД:

- SQLite 3;
- PostgreSQL 9.6 или старше;
- MySQL/MariaDB (tested on MariaDB 10.2.36).

Для расширения списка поддерживаемых СУБД в ядро включены механизмы системы расширений, загружаемых при старте приложения. Данное поведение можно изменить в файле конфигурации приложения. Более подробно о расширении системы по работе с базами данных будет описано позже.

Основной список методов, предоставляемый ядром, для работы с базами данных:

- `->registerDatabaseAdapter(string $name, string $className)` - метод регистрации нового адаптера по работе с базой данных, где «\$name» - название адаптера в конфигурационном файле, «\$className» - название PHP класса адаптера;
ПРИМЕЧАНИЕ: данный метод актуален для регистрации нового адаптера из системы расширений.
- `->createDatabaseAdapter(bool $autoConnect = true)` - создать новый адаптер и подключиться к базе данных, если значение «\$autoConnect» задано «true»;
ПРИМЕЧАНИЕ: если по какой-то причине создать новый адаптер или подключиться к базе данных не удается, генерируется исключение или возвращается «null».
- `->currentAdapterName()` - получить текущее название адаптера, заданное в конфигурационном файле.

Все создаваемые ядром адаптеры должны быть унаследованы от базового класса «\FlyCubePHP\Core\Database\BaseDatabaseAdapter». Данный класс предоставляет следующий перечень методов по созданию запросов к базе данных:

- `->recreatePDO(array $settings)` - пересоздать соединение с базой данных; если новый массив «\$settings» не задан, то используются настройки, заданные при создании объекта;
- `->isValid()` - проверка на валидность, т.е. был ли создан объект PDO по работе с базой данных;
- `->setShowOutput(bool $value)` - задать флаг вывода результатов работы в консоль (`default: false`);
- `->setOutputDelimiter(string $value)` - задать разделитель строк для выводимых данных (`default: '
'`);
- `->settings()` - получить массив с текущими настройками адаптера;
- `->database()` - получить имя текущей базы данных из настроек;

- `->query(string $sql, array $params, string $className)` - выполнить запрос к базе данных, где:
 - `$sql` - SQL запрос;
 - `$params` - параметры запроса;
 - `$className` - имя класса, для создания объектов результата (default: `'stdClass'`);
- `->queryTransaction(string $sql, array $params, string $className)` - выполнить запрос к базе данных в рамках одной транзакции, где:
 - `$sql` - SQL запрос;
 - `$params` - параметры запроса;
 - `$className` - имя класса, для создания объектов результата (default: `'stdClass'`);
- `->beginTransaction()` - открыть транзакцию;
- `->commitTransaction()` - применить транзакцию;
- `->rollBackTransaction()` - отменить транзакцию;
- `->inTransaction()` - проверить, открыта ли транзакция;
- `->serverVersion()` - получить версию сервера базы данных;
- `->tables()` - получить список таблиц базы данных;
- `->name()` - получить имя адаптера;
- `->quoteTableName(string $name)` - получить корректное экранированное имя таблицы.

ПРИМЕЧАНИЕ: во избежании SQL инъекций рекомендуется выполнять запросы к базе данных без подстановки значений в сам запрос. Данные параметры должны передаваться аргументом `«$params»` методов `«->query(...»` и `«->queryTransaction(...»`. Пример корректного формирования запроса к базе данных:

```
$res = $db->query("SELECT * FROM \"myTable\" LIMIT :f_value;", [ ":f_value" => 50 ]);
```

где:

- `:f_value` - ключ из массива `«$params»` подставляемых в запрос значений.

Пример работы с базой данных приводится ниже:

```
class AppCoreController extends ApplicationController
{
    public function test() {
        $db = \FlyCubePHP\Core\Database\DatabaseFactory::instance()->createDatabaseAdapter();
        if (is_null($db))
            return;

        try {
            $res = $db->query("SELECT * FROM \"myTable\" LIMIT :f_value;", [ ":f_value" => 50 ]);
        } catch (ErrorDatabase $ex) {
            // Show error message.
            var_dump($ex->getMessage());
            return;
        }
        // Show result data.
        var_dump($res);
    }
}
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файлах «FlyCubePHP/src/Core/Database/DatabaseFactory.php» и «FlyCubePHP/src/Core/Database/BaseDatabaseAdapter.php».

10.1. Настройка доступа к базам данных

Весь доступ приложения к базе данных задается в конфигурационном файле «[MyProject]/config/database.json» (где «MyProject» - название нового проекта). Данный файл может содержать неограниченное число настроек для подключения к различным СУБД, но поля «production» и «development» могут быть заданы только один раз. Данные поля указывают ядру системы, какую секцию настроек требуется использовать для подключения и какой адаптер требуется создать в том или ином режиме работы приложения.

Пример файла конфигурации доступа к СУБД приводится ниже:

```
{
  "default_sqlite_dev": {
    "adapter": "sqlite",
    "database": "db/development.sqlite3"
  },
  "default_sqlite_prod": {
    "adapter": "sqlite",
    "database": "db/production.sqlite3"
  },
  "default_postgresql_dev": {
    "adapter": "postgresql",
    "database": "php_test_dev",
    "host": "127.0.0.1",
    "port": 5432,
    "username": "postgres",
    "password": "postgres"
  },
  "default_postgresql_prod": {
    "adapter": "postgresql",
    "database": "php_test_prod",
    "host": "127.0.0.1",
    "port": 5432,
    "username": "postgres",
    "password": "postgres"
  },
  "default_postgresql_unix_dev": {
    "adapter": "postgresql",
    "database": "fly_cube_php_dev",
    "unix_socket_dir": "/var/run/postgresql",
    "username": "postgres",
    "password": "postgres"
  },
  "default_postgresql_unix_prod": {
    "adapter": "postgresql",
    "database": "fly_cube_php_prod",
    "unix_socket_dir": "/var/run/postgresql",
    "username": "postgres",
    "password": "postgres"
  },
  "default_mysql_dev": {
    "adapter": "mysql",
    "database": "fly_cube_php_dev",
    "host": "127.0.0.1",
    "port": 3306,
    "username": "root",
    "password": "root"
  },
  "default_mysql_prod": {
    "adapter": "mysql",
    "database": "fly_cube_php_prod",
    "host": "127.0.0.1",
    "port": 3306,
    "username": "root",
    "password": "root"
  }
},
```

```

"default_mysql_prod": {
    "adapter": "mysql",
    "database": "fly_cube_php_prod",
    "host": "127.0.0.1",
    "port": 3306,
    "username": "root",
    "password": "root"
},
"default_mysql_unix_dev": {
    "adapter": "mysql",
    "database": "fly_cube_php_dev",
    "unix_socket": "/run/mysql/mysql.sock",
    "username": "root",
    "password": "root"
},
"default_mysql_unix_prod": {
    "adapter": "mysql",
    "database": "fly_cube_php_prod",
    "unix_socket": "/run/mysql/mysql.sock",
    "username": "root",
    "password": "root"
},
"production": "default_sqlite_prod",
"development": "default_postgresql_dev"
}

```

где:

- "default_sqlite_dev" - секция с настройками доступа к SQLite базе данных в «development mode»;
ПРИМЕЧАНИЕ: полный путь файла базы данных будет «[MyProject]/db /development.sqlite3».
- "default_sqlite_prod" - секция с настройками доступа к SQLite базе данных в «production mode»;
ПРИМЕЧАНИЕ: полный путь файла базы данных будет «[MyProject]/db /production.sqlite3».
- "default_postgresql_dev" - секция с настройками доступа к PostgreSQL базе данных в «development mode»;
- "default_postgresql_prod" - секция с настройками доступа к PostgreSQL базе данных в «production mode»;
- "default_postgresql_unix_dev" - секция с настройками доступа к PostgreSQL базе данных в «development mode» используя файловый unix сокет;
- "default_postgresql_unix_prod" - секция с настройками доступа к PostgreSQL базе данных в «production mode» используя файловый unix сокет;
- "default_mysql_dev" - секция с настройками доступа к MySQL/MariaDB базе данных в «development mode»;
- "default_mysql_prod" - секция с настройками доступа к MySQL/MariaDB базе данных в «production mode»;
- "default_mysql_unix_dev" - секция с настройками доступа к MySQL/MariaDB базе данных в «development mode» используя файловый unix сокет;
- "default_mysql_unix_prod" - секция с настройками доступа к MySQL/MariaDB базе данных в «production mode» используя файловый unix сокет;
- "production" - содержит название секции доступа к базе данных для «production mode»;
- "development" - содержит название секции доступа к базе данных для «development mode».

11. Модели данных

Модели данных представляют собой объекты, предоставляющие удобный для разработчика интерфейс по взаимодействию с таблицами в базе данных. Все модели данных во FlyCubePHP наследуются от базового класса «\FlyCubePHP\Core\ActiveRecord\ActiveRecord». Данный класс поддерживает широкий набор методов, позволяющих быстро и эффективно производить манипуляции с данными, а конечный SQL запрос формируется только из тех полей, которые были изменены или добавлены новые. Доступный перечень методов приводится ниже в отдельных главах, а описание базового класса можно найти в файле «FlyCubePHP/src/Core/ActiveRecord/ActiveRecord.php».

11.1. Создание новой модели данных

FlyCubePHP позволяет создавать два типа моделей данных:

- модель данных приложения;
- модель данных плагина.

Оба типа моделей отличаются только расположением файла в дереве каталогов проекта, а на момент запуска будут доступны всем классам приложения, как и в Ruby on Rails.

Каталоги с расположением моделей данных:

- «[MyProject]/app/models/» - каталог моделей данных приложения;
- «[MyProject]/plugins/*/app/models/» - каталог моделей данных плагина.

Для создания новой модели данных приложения (например: модель по работе с таблицей пользователей) требуется выполнить следующую команду генератора «./fly_cube_php --new --model --name=User»:

```
MyProject/bin> ./fly_cube_php --new --model --name=User  
== FlyCubePHP: Create new model ==  
[Created] app/models/User.php  
[Created] db/migrate/20211012104949_CreateUsers.php  
== FlyCubePHP =====
```

Для создания новой модели данных плагина требуется выполнить команду генератора с частично измененными аргументами «./fly_cube_php --new --plugin-model --name=User --plugin-name=ExamplePlugin»:

```
MyProject/bin> ./fly_cube_php --new --plugin-model --name=User --plugin-name=ExamplePlugin  
== FlyCubePHP: Create new plugin model ==  
[Created] plugins/ExamplePlugin/app/models/User.php  
[Created] plugins/ExamplePlugin/db/migrate/20211012105405_CreateUsers.php  
== FlyCubePHP =====
```

При создании новой модели автоматически создается файл миграции, который позволит внести все необходимые изменения в структуру базы данных. Более подробно о миграциях будет рассказано в следующих главах.

ПРИМЕЧАНИЕ: рекомендуется не использовать множественное число в названии моделей данных, так как объект модели сопоставляет себя объекту в базе данных. Например, таблица по работе с пользователями в базе данных называется «users», тогда класс модели данных должен называться «User», так как описывает одну строку данных в таблице, а при запросе всей таблицы будет сформирован массив из объектов «User».

11.2. Возможности моделей данных

В данном разделе описываются базовые методы класса «\FlyCubePHP\Core\ActiveRecord\ActiveRecord», позволяющие производить различные манипуляции с данными.

11.2.1. tableName / setTableName

Данная пара методов позволяет получить или задать имя таблицы, с которой работает класс модели. Поумолчанию имя таблицы формируется из имени класса модели, приведенного в «underscore» формат. Например, если имя класса модели данных «UserInfo», то имя таблицы в базе данных должно быть «user_info».

Описание методов:

```
/**  
 * Имя текущей таблицы  
 * @return string  
 */  
final protected function tableName(): string {...}  
  
/**  
 * Задать имя текущей таблицы  
 * @param string $name  
 */  
final protected function setTableName(string $name) {...}
```

Пример использования методов:

```
class User extends \FlyCubePHP\Core\ActiveRecord\ActiveRecord  
{  
    public function __construct() {  
        //  
        // Before parent construct: $this->tableName() == "" (empty string)  
        //  
        parent::__construct(); // NOTE: always use to initialize the parent class!  
        //  
        // After parent construct: $this->tableName() == "user"  
        //  
        // Set new table name:  
        $this->setTableName("test_schema.users");  
    }  
}
```

11.2.2. primaryKey / setPrimaryKey

Данная пара методов позволяет получить или задать имя колонки первичного ключа таблицы, с которой работает класс модели. Поумолчанию имя первичного ключа таблицы - «`id`». Имя первичного ключа используется для формирования корректных SQL запросов к базе данных, таких как обновление или удаление записи, поиск записи и т.д.

Описание методов:

```
/**  
 * Имя колонки первичного ключа  
 * @return string  
 */  
final protected function primaryKey(): string {...}  
  
/**  
 * Задать имя колонки первичного ключа  
 * @param string $name  
 */  
final protected function setPrimaryKey(string $name) {...}
```

Пример использования методов:

```
class User extends \FlyCubePHP\Core\ActiveRecord\ ActiveRecord  
{  
    public function __construct() {  
        //  
        // Before parent construct: $this->primaryKey() == "id"  
        //  
        parent::__construct(); // NOTE: always use to initialize the parent class!  
        //  
        // After parent construct: $this->primaryKey() == "id"  
        //  
        // Set new primary key name:  
        $this->setPrimaryKey("uuid");  
    }  
}
```

11.2.3. hasColumnMapping / columnMapping / setColumnMapping

Данный набор методов предназначен для запроса или формирования сопоставлений параметров класса названию колонки в таблице базы данных. Данный набор методов позволяет избежать ошибки при выполнении запросов, когда имена колонок таблиц имеют нестандартный вид.

Пример некорректного преобразования без сопоставлений:

```
//  
// Преобразование при запросе из БД (где name_ - имя колонки в таблице):  
//  
// name_ ==> ActiveRecord->name  
//  
// Обратное преобразование при отправке в БД:  
//  
// ActiveRecord->name ==> name <-- FAIL!  
//
```

Описание методов:

```
/**  
 * Задано ли сопоставление параметра класса реальному названию колонки в таблице  
 * @param string $classParam  
 * @return bool  
 */  
final protected function hasColumnMapping(string $classParam): bool {...}  
  
/**  
 * Сопоставление параметра класса реальному названию колонки в таблице  
 * @param string $classParam  
 * @return string  
 */  
final protected function columnMapping(string $classParam): string {...}  
  
/**  
 * Массив сопоставлений параметров класса реальным названиям колонок в таблице  
 * @return array  
 */  
final protected function columnMappings():array {...}  
  
/**  
 * Задать сопоставление параметра класса реальному названию колонки в таблице  
 * @param string $classParam  
 * @param string $columnName  
 */  
final protected function setColumnMapping(string $classParam, string $columnName) {...}  
  
/**  
 * Задать сопоставление параметров класса реальным названиям колонок в таблице  
 * @param array $columnMappings  
 *  
 * ПРИМЕЧАНИЕ:  
 * $columnMappings должен иметь следующий вид:  
 *   * key    - название параметра класса  
 *   * value  - название колонки в таблице  
 *  
 * Example:  
 * setColumnMappings([ 'name' => 'name_', 'password' => 'password_' ])  
 */  
final protected function setColumnMappings(array $columnMappings) {...}
```

Пример использования методов:

```
class User extends \FlyCubePHP\Core\ActiveRecord\ ActiveRecord  
{  
    public function __construct()  
    {  
        parent::__construct(); // NOTE: always use to initialize the parent class!  
  
        // Set column mapping:  
        $this->setColumnMapping("name", "name_");  
  
        //  
        // After setColumnMapping:  
        //  
        // User->name ==> name_ <-- OK  
    }  
}
```

11.2.4. passwordColumn / setPasswordColumn

Данная пара методов позволяет получить или задать имя колонки с паролем, которая автоматически будет преобразовывать данные в безопасный хэш. Поумолчанию имя колонки с паролем - «password».

Описание методов:

```
/**  
 * Имя колонки с паролем  
 * @return string  
 */  
final protected function passwordColumn(): string {...}  
  
/**  
 * Задать имя колонки с паролем  
 * @param string $name  
 */  
final protected function setPasswordColumn(string $name) {...}
```

Пример использования методов:

```
class User extends \FlyCubePHP\Core\ActiveRecord\ActiveRecord  
{  
    public function __construct()  
    {  
        //  
        // Before parent construct: $this->passwordColumn() == "password"  
        //  
        parent::__construct(); // NOTE: always use to initialize the parent class!  
        //  
        // After parent construct: $this->passwordColumn() == "password"  
        //  
        // Set new password column name:  
        $this->setPasswordColumn("my_password");  
    }  
}
```

11.2.5. isAuthenticated

Данный метод предназначен для проверки корректности введенного пароля пользователя. Он производит сравнение введенных и сохраненных данных. Для сравнения используются данные параметра колонки с хэшем пароля. Если такой параметр класса отсутствует, то метод возвращает `false`.

Описание метода:

```
/**  
 * Проверка авторизации  
 * @param string $plainPassword - нешифрованный пароль  
 * @return bool  
 *  
 * ПРИМЕЧАНИЕ:  
 * Проверка выполняется, если в классе присутствует параметр с паролем.  
 * Иначе - false.  
 */  
final public function isAuthenticated(string $plainPassword): bool {...}
```

Пример использования метода проверки пароля:

```
class AppCoreController extends ApplicationController  
{  
    public function test() {  
        // Select user object.  
        $userObject = User::find($this->_params['id']);  
        // Check is found  
        if (is_null($userObject))  
            return; // Invalid user id! Show error!  
        if (!$userObject->isAuthenticated($this->_params['password']))  
            return; // Invalid user password! Show error!  
    }  
}
```

11.2.6. isReadOnly / setReadOnly

Данная пара методов позволяет получить или установить флаг модели «только для чтения». Поумолчанию данный флаг - «`false`».

Описание методов:

```
/**  
 * Находится ли объект класса в режиме "Только чтение"  
 * @return bool  
 *  
 * NOTE: Call to the save/destroy functions with the specified "read-only" flag  
 * will trigger an error!  
 */  
final protected function isReadOnly(): bool {...}  
  
/**  
 * Задать режим "Только чтение" для объекта класса  
 * @param bool $value  
 *  
 * NOTE: Call to the save/destroy functions with the specified "read-only" flag  
 * will trigger an error!  
 */  
final protected function setReadOnly(bool $value) {...}
```

Пример использования методов:

```
class User extends \FlyCubePHP\Core\ActiveRecord\ActiveRecord
{
    public function __construct() {
        parent::__construct(); // NOTE: always use to initialize the parent class!
        // Set read-only flag:
        $this->setReadOnly(true);
    }
}
```

11.2.7. save

Данный метод предназначен для сохранения изменений объекта в базе данных. Данные могут быть как новыми, так и измененными.

Описание метода:

```
/**
 * Сохранить/Обновить объект в БД
 * @throws
 */
final public function save() {...}
```

Пример использования метода сохранения нового объекта:

```
class AppCoreController extends ApplicationController
{
    public function test() {
        // Create new user object.
        $userObject = new User();
        // Set name.
        $userObject->name = "Test name";
        // Append new object in database.
        $userObject->save();
    }
}
```

SQL запрос при сохранении нового объекта:

```
INSERT INTO "users" ("name") VALUES ("Test name");
```

Пример использования метода сохранения изменений:

```
class AppCoreController extends ApplicationController
{
    public function test() {
        // Select first user object from database.
        $userObject = User::first();
        if (is_null($userObject))
            return;
        // Show user object.
        var_dump($userObject);
        // Set new name.
        $userObject->name = "Test new name";
        // Save changes.
        $userObject->save();
    }
}
```

SQL запрос при сохранении изменений объекта:

```
UPDATE "users" SET "name" = "Test name" WHERE "users"."id" = 1;
```

11.2.8. destroy

Данный метод предназначен для удаления объекта и его данных из таблицы базы данных. Поиск в таблице производится по первичному ключу.

Описание метода:

```
/**  
 * Удалить объект из БД  
 * @throws  
 */  
final public function destroy() {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController  
{  
    public function test() {  
        // Select first user object from database.  
        $userObject = User::first();  
        if (is_null($userObject))  
            return;  
        // Show user object.  
        var_dump($userObject);  
        // Delete object from database.  
        $userObject->destroy();  
    }  
}
```

SQL запрос:

```
DELETE FROM "users" WHERE "users"."id" = 1;
```

11.2.9. all

Данный метод предназначен для запроса всех данных из таблицы базы данных. Результатом выполнения метода будет массив объектов класса модели.

Описание метода:

```
/**  
 * Запрос всех объектов из таблицы  
 * @return array  
 * @throws  
 */  
final public static function all(): array {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController  
{  
    public function test() {  
        // Select all user objects from database.  
        $userObjects = User::all();  
        if (is_null($userObjects))  
            return;  
        // Show user object.  
        var_dump($userObjects); // array[User, User, ..., User]  
    }  
}
```

SQL запрос:

```
SELECT * FROM "users";
```

11.2.10. `destroyAll`

Данный метод предназначен для удаления всех данных из таблицы базы данных.

Описание метода:

```
/**  
 * Удалить все объекты из таблицы  
 * @throws  
 */  
final public static function destroyAll() {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController  
{  
    public function test() {  
        //  
        // Select all user objects from database.  
        var_dump(User::all()); // array[User, User, ..., User]  
        //  
        // Delete all user objects from database.  
        User::destroyAll();  
        //  
        // Select all user objects from database.  
        var_dump(User::all()); // empty array  
    }  
}
```

SQL запрос:

```
DELETE FROM "users";
```

11.2.11. `first`

Данный метод предназначен для запроса первого объекта из таблицы базы данных. Результатом выполнения метода будет объект класса модели или «null», если таблица данных пуста.

Описание метода:

```
/**  
 * Запрос первого объекта из таблицы  
 * @return Object|null  
 * @throws  
 */  
final public static function first() {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController  
{  
    public function test() {  
        //  
        // Select first user objects from database.  
        var_dump(User::first()); // Class User - if table not empty.  
                                // null - if table empty.  
    }  
}
```

SQL запрос:

```
SELECT * FROM "users" LIMIT 1;
```

11.2.12. limit

Данный метод предназначен для запроса определенного числа объектов из таблицы базы данных. Результатом выполнения метода будет объект массив объектов класса модели или «null», если таблица данных пуста.

Описание метода:

```
/**  
 * Запросить определенное количество объектов из БД  
 * @param int $num  
 * @return array|null  
 * @throws  
 */  
final public static function limit(int $num) {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController  
{  
    public function test() {  
        //  
        // Select first five user objects from database.  
        var_dump(User::limit(5)); // array[User, User, ..., User] - if table not empty.  
                                // null - if table empty.  
    }  
}
```

SQL запрос:

```
SELECT * FROM "users" LIMIT 5;
```

11.2.13. count

Данный метод предназначен для запроса количества объектов в таблице базы данных.

Описание метода:

```
/**  
 * Запросить количество объектов в БД  
 * @return int  
 * @throws  
 */  
final public static function count() {}
```

Пример использования:

```
class AppCoreController extends ApplicationController  
{  
    public function test() {  
        //  
        // Select count user objects from database.  
        var_dump(User::count()); // N - if table not empty (where N - table rows count).  
                                // 0 - if table empty.  
    }  
}
```

SQL запрос:

```
SELECT COUNT(*) AS count FROM "users";
```

11.2.14. find

Данный метод предназначен для поиска объекта в таблице базы данных по значению первичного ключа. Результатом выполнения метода будет объект класса модели или «null», если таблица данных пуста или объект не найден.

Описание метода:

```
/**  
 * Поиск объекта в БД по значению первичного ключа  
 * @param $pkVal  
 * @return Object|null  
 * @throws  
 */  
final public static function find($pkVal) {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController  
{  
    public function test()  
    {  
        //  
        // Find user objects by primary key, where primary key value == 123.  
        var_dump(User::find(123)); // Class User - if object found.  
                                // null - if table empty or object not found.  
    }  
}
```

SQL запрос:

```
SELECT * FROM "users" WHERE "users"."id" = 123;
```

11.2.15. findBy

Данный метод предназначен для поиска объекта в таблице базы данных по значению произвольной колонки. Результатом выполнения метода будет объект класса модели или «null», если таблица данных пуста или объект не найден.

Описание метода:

```
/**  
 * Поиск объекта в БД по значению произвольной колонки  
 * @param string $column  
 * @param $val  
 * @param bool $prepareNames - автоматический перевод имен колонок в underscore  
 * @return array|null  
 * @throws  
 */  
final public static function findBy(string $column, $val, bool $prepareNames = true) {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController
{
    public function test() {
        //
        // Find user objects by column "name", where value == "Test Name".
        var_dump(User::findBy("name","Test Name")); // Class User - if object found.
                                                // null - if table empty or object not found.

    }
}
```

SQL запрос:

```
SELECT * FROM "users" WHERE "users"."name" = "Test Name";
```

11.2.16. findBySql

Данный метод предназначен для поиска объекта в таблице базы данных используя произвольный SQL запрос. Результатом выполнения метода будет массив объектов класса модели или «null», если таблица данных пуста или объект не найден.

Описание метода:

```
/**
 * Поиск объекта в БД используя произвольный SQL
 * @param string $sql - SQL запрос
 * @param array $params - массив параметров запроса с их значениями
 * @return array|null
 * @throws
 */
final public static function findBySql(string $sql, array $params = []) {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController
{
    public function test() {
        //
        // Find user objects by SQL query.
        var_dump(User::findBySql("SELECT * FROM users WHERE type = :type_v", [":type_v" => 100]));
    }
}
```

SQL запрос:

```
SELECT * FROM users WHERE type = 100;
```

11.2.17. where

Данный метод предназначен для запроса объектов из таблицы базы данных, отвечающие определенным условиям. Результатом выполнения метода будет массив объектов класса модели или «null», если таблица данных пуста или объект не найден.

Описание метода:

```
/**  
 * Запрос объектов с фильтрацией  
 * @param array $args  
 * @param bool $prepareNames - автоматический перевод имен в underscore  
 * @return array|null  
 * @throws  
 */  
final public static function where(array $args, bool $prepareNames = true) {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController  
{  
    public function test() {  
        //  
        // Select user objects with filter.  
        var_dump(User::where([ "name" => "test", "type" => 100 ]));  
    }  
}
```

SQL запрос:

```
SELECT * FROM "users" WHERE "users"."name" = "test" AND "users"."type" = 100;
```

11.2.18. exists

Данный метод предназначен для проверки наличия объекта в таблице базы данных по значению первичного ключа.

Описание метода:

```
/**  
 * Проверка наличия объекта в БД по первичному ключу  
 * @param $pkVal  
 * @return bool  
 * @throws  
 */  
final public static function exists($pkVal) {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController  
{  
    public function test() {  
        //  
        // Check user objects where primary key == 123.  
        var_dump(User::exists(123));  
    }  
}
```

SQL запрос:

```
SELECT 1 AS one FROM "users" WHERE "users"."id" = 123 LIMIT 1;
```

11.2.19. existsSome

Данный метод предназначен для проверки наличия объекта в таблице базы данных по значению произвольной колонки.

ПРИМЕЧАНИЕ: данный метод вернет «true» если из набора значений в таблице базы данных есть хотя бы одно.

Описание метода:

```
/**  
 * Проверка наличия объекта в БД по произвольной колонке и ее значений  
 * @param string $column  
 * @param array $values  
 * @param bool $prepareNames - автоматический перевод имен в underscore  
 * @return bool  
 * @throws  
 */  
final public static function existsSome(string $column, array $values, bool $prepareNames = true)  
{...}
```

Пример использования:

```
class AppCoreController extends ApplicationController  
{  
    public function test()  
    {  
        //  
        // Check user objects where column "type" value == 100 or 200 or 300.  
        var_dump(User::existsSome("type", [ 100, 200, 300 ]));  
    }  
}
```

SQL запрос:

```
SELECT 1 AS one FROM "users" WHERE "users"."type" IN (100 200 300) LIMIT 1;
```

11.2.20. selectSome

Данный метод предназначен для запроса произвольного набора колонок объекта из таблицы базы данных.

Описание метода:

```
/**  
 * Запрос произвольного набора колонок для объекта  
 * @param array $columns - имена необходимых колонок  
 * @param bool $prepareNames - автоматический перевод имен в underscore  
 * @return array|null  
 * @throws  
 */  
final public static function selectSome(array $columns, bool $prepareNames = true) {...}
```

Пример использования:

```
class AppCoreController extends ApplicationController
{
    public function test() {
        //
        // Select user objects with only columns: "name" and "type".
        var_dump(User::selectSome([ "name", "type" ]));
    }
}
```

SQL запрос:

```
SELECT "name", "type" FROM "users";
```

11.2.21. Функции обратного вызова (callbacks)

Функции обратного вызова нужны для выполнения дополнительных действий перед (после) выполнением определенных действий с объектом модели данных. На данный момент FlyCubePHP ActiveRecord предоставляет возможность для регистрации следующий callback-ов:

- `->beforeSave(string $method)` - добавить callback, который будет вызван перед сохранением объекта модели;
- `->afterSave(string $method)` - добавить callback, который будет вызван после сохранения объекта модели;
- `->beforeInsert(string $method)` - добавить callback, который будет вызван перед вставкой нового объекта модели в базу данных;
- `->afterInsert(string $method)` - добавить callback, который будет вызван после вставки нового объекта модели в базу данных;
- `->beforeUpdate(string $method)` - добавить callback, который будет вызван перед обновлением объекта модели в базе данных;
- `->afterUpdate(string $method)` - добавить callback, который будет вызван после обновления объекта модели в базе данных;
- `->beforeDestroy(string $method)` - добавить callback, который будет вызван перед удалением объекта модели из базы данных;
- `->afterDestroy(string $method)` - добавить callback, который будет вызван после удаления объекта модели из базы данных.

ПРИМЕЧАНИЕ: функции для callback-ов должны иметь область видимости «public» или «protected».

Пример использования callback-a:

```
class MyModel extends \FlyCubePHP\Core\ActiveRecord\ActiveRecord
{
    public function __construct() {
        $this->beforeSave('preparePassword');
    }

    /**
     * Метод подготовки пароля к сохранению в БД
     */
    final protected function preparePassword() {
        $tmpPassName = CoreHelper::camelize($this->_passwordColumn, false);
        if (!isset($this->$tmpPassName))
            return;
        $tmpValue = $this->$tmpPassName;
        if ($this->_newRecord === false
            && array_key_exists($tmpPassName, $this->_dataHash)
            && $this->_dataHash[$tmpPassName] === $tmpValue)
            return; // skip not changed value
        $this->$tmpPassName = $this->encryptPassword($tmpValue);
    }
}
```

11.2.22. Специальные функции обработки

Специальные функции обработки необходимы для внесение изменений в работу класса FlyCubePHP ActiveRecord, без его модификации. Данные функции имеют четко объявленный набор входных аргументов и формат возвращаемых данных. На данный момент доступны следующие специальные функции обработки:

- `->beforeSet(string $name, mixed $value)`: `mixed` - метод предобработки добавляемых данных (аналог метода `__set` с возвращаемыми данными).

Пример использования:

```
class MyModel extends \FlyCubePHP\Core\ActiveRecord\ActiveRecord
{
    /**
     * Метод предобработки добавляемых данных (aka __set with return)
     * @param string $name
     * @param mixed $value
     * @return mixed
     */
    final protected function beforeSet(string $name, $value) {
        if (strcmp($name, 'classifMetadata') !== 0)
            return $value;
        if (empty($value) || !is_string($value))
            return $value;
        $tmpXml = simplexml_load_string(base64_decode($value));
        $tmpValue = [];
        foreach ($tmpXml->data as $data) {
            $guid = "";
            foreach ($data->attributes() as $attr => $attrVal) {
                if (strcmp($attr, 'guid') === 0) {
                    $guid = strval($attrVal);
                    break;
                }
            }
            $tmpValue[$guid] = strval($data);
        }
        return $tmpValue;
    }
}
```

12. Миграции Active Record

Миграции - это удобный способ управления структурой вашей базы данных. Как и в Ruby on Rails, система миграций FlyCubePHP предоставляет широкий набор методов для работы со схемой базы данных. Данный подход позволяет не использовать SQL для описания или изменения схемы базы данных, что в свою очередь дает возможность не зависеть от конкретной СУБД и её свойств.

Каждая новая миграция вносит изменения в структуру базы данных, добавляя, изменяя или удаляя таблицы, их колонки или свойства, и может рассматриваться как «новая» версия базы данных.

Во избежании проблем и конфликтов при установке миграций, каждый новый файл получает уникальный префикс, который так же является его версией. Префикс представляет собой текущую дату-время в формате YYYYMMDDHHmmSS, где:

- YYYY - год;
- MM - месяц с ведущим нулём (01-12);
- DD - день с ведущим нулём (01-31);
- HH - часы с ведущим нулём в 24 часовом формате (00-23);
- mm - минуты с ведущим нулём (00-59);
- SS - секунды с ведущим нулём (00-59).

Данный подход позволяет устанавливать миграции только в том порядке, в котором они были созданы. Все версии установленных миграций хранятся в базе данных, что позволяет в любой момент времени получить полную информацию о текущей версии базы данных или состояние каждой миграции из списка доступных.

Установка или удаление (откат) миграций производится последовательно в режиме транзакции, если его поддерживает СУБД. В случае успешной операции в базу данных сохраняется текущая версия миграций и создается (или обновляется) файл «слепка» текущей схемы базы данных «[MyProject]/db/Schema.php», который так же можно использовать для быстрого развертывания.

Ядро системы по работе с миграциями реализовано в классе «\FlyCubePHP\Core\Migration\MigrationsCore» и занимается поиском файлов миграций в рамках текущего проекта и обработкой заданий по их установке или удалению (откату). FlyCubePHP поддерживает следующие СУБД:

- SQLite 3;
- PostgreSQL 9.6 или старше;
- MySQL/MariaDB (tested on MariaDB 10.2.36).

Для расширения списка поддерживаемых СУБД в ядро миграций включены механизмы системы расширений, загружаемых при начале работы с миграциями. Данное поведение можно изменить в файле конфигурации приложения. Более подробно о расширении системы миграций будет описано позже.

12.1. Создание новой миграции

FlyCubePHP позволяет создавать два типа миграций:

- миграции приложения;
- миграции плагина.

Оба типа миграций отличаются только расположением файла в дереве каталогов проекта. При работе с миграциями ядро производит поиск миграций в следующих каталогах проекта:

- «[MyProject]/db/migrate/» - каталог миграций приложения;
- «[MyProject]/plugins/*/db/migrate/» - каталог миграций плагина.

ПРИМЕЧАНИЕ: если поддержка системы плагинов отключена флагом «FLY_CUBE_PHP_ENABLE_PLUGINS_CORE: false», то поиск миграций плагинов производиться не будет.

Для создания новой миграции приложения требуется выполнить следующую команду генератора «./fly_cube_php --new --migration --name=TestMigration»:

```
MyProject/bin> ./fly_cube_php --new --migration --name=TestMigration
```

```
==== FlyCubePHP: Create new migration ===
[Created] db/migrate/20211013094233_TestMigration.php
==== FlyCubePHP =====
```

Для создания новой миграции плагина требуется выполнить команду генератора с частично измененными аргументами «./fly_cube_php --new --plugin-migration --name=TestMigration --plugin-name=ExamplePlugin»:

```
MyProject/bin> ./fly_cube_php --new --plugin-migration --name=TestMigration --plugin-name=ExamplePlugin
==== FlyCubePHP: Create new plugin migration ===
[Created] plugins/ExamplePlugin/db/migrate/20211013094439_TestMigration.php
==== FlyCubePHP =====
```

Содержимое файла миграции будет следующим:

```
<?php

class TestMigration extends \FlyCubePHP\Core\Migration
{
    final public function up() {
        // Set here your code for install migration
    }

    final public function down() {
        // Set here your code for uninstall migration
    }
}
```

12.2. Возможности миграций

Как видно из примера выше, показывающего содержимое файла миграции, все миграции наследуются от базового класса «\FlyCubePHP\Core\Migration\Migration». Данный класс обязует разработчика реализовать два обязательных метода:

- up - метод, содержащий инструкции по внесению изменений в схему базы данных;
- down - метод, содержащий инструкции по удалению изменений из схемы базы данных и её откат к предыдущему состоянию.

Доступный перечень методов приводится ниже в отдельных главах, а описание базового класса можно найти в файле «FlyCubePHP/src/Core/Migration/Migration.php».

12.2.1. up

Данный метод обязателен к реализации в файле миграции и содержит инструкции по внесению изменений в схему базы данных.

Описание метода:

```
/**  
 * Внесение изменений миграции  
 * @return mixed  
 */  
abstract protected function up();
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function up() {  
        // Create new table  
        $this->createTable('test',  
            [  
                'id' => false,  
                'if_not_exists' => true,  
                'my_id' => [ 'type' => 'integer', 'null' => false, 'primary_key' => true ],  
                'my_data' => [ 'type' => 'integer', 'null' => false ]  
            ]);  
    }  
}
```

12.2.2. down

Данный метод обязателен к реализации в файле миграции и содержит инструкции по удалению изменений из схемы базы данных и её откат к предыдущему состоянию.

Описание метода:

```
/**  
 * Удаление изменений миграции  
 * @return mixed  
 */  
abstract protected function down();
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function down() {  
        // Delete table  
        $this->dropTable('test');  
    }  
}
```

12.2.3. adapterName

Данный метод позволяет получить название текущего адаптера, используемого для доступа к базе данных. Если объект адаптера не был создан, возвращается пустая строка.

Описание метода:

```
/**  
 * Название адаптера для работы с базой данных  
 * @return string  
 */  
final protected function adapterName(): string {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function up() {  
        var_dump($this->adapterName());  
    }  
  
    final public function down() {  
        ...  
    }  
}
```

12.2.4. createSchema

Данный метод позволяет создать новую схему в базе данных. Если база данных не поддерживает данный функционал, операция игнорируется.

Описание метода:

```
/**  
 * Создать новую схему данных  
 * @param string $name - имя  
 * @param array $props - свойства  
 *  
 * Supported Props:  
 *  
 * [bool] if_not_exists - добавить флаг 'IF NOT EXISTS'  
 */  
final protected function createSchema(string $name, array $props = []) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function up() {  
        //  
        // Create schema without check 'IF NOT EXISTS'  
        //  
        // If the schema "test" already exists, then an SQL error will be generated  
        // and the installation of the migration will fail!  
        //  
        $this->createSchema('test');  
  
        //  
        // Create schema with check 'IF NOT EXISTS'  
        //  
        // If the schema "gui" already exists, this step is ignored.  
        //  
        $this->createSchema('gui', [ 'if_not_exists' => true ]);  
    }  
  
    final public function down() {  
        ...  
    }  
}
```

12.2.5. dropSchema

Данный метод позволяет удалить созданную ранее схему в базе данных. Если база данных не поддерживает данный функционал, операция игнорируется.

Описание метода:

```
/**  
 * Удалить схему данных  
 * @param string $name - имя  
 * @param array $props - свойства  
 *  
 * Supported Props:  
 *  
 * [bool] if_exists - добавить флаг 'IF EXISTS'  
 */  
final protected function dropSchema(string $name, array $props = []) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        $this->createSchema('test');
        $this->createSchema('gui');
    }

    final public function down() {
        //
        // Delete schema without check 'IF EXISTS'
        //
        // If the "test" schema does not exist, then an SQL error will be generated
        // and the migration rollback will fail!
        //
        $this->dropSchema('test');

        //
        // Delete schema with check 'IF EXISTS'
        //
        // If the schema "gui" does not exist, this step is ignored.
        //
        $this->dropSchema('gui', [ 'if_exists' => true ]);
    }
}
```

12.2.6. createTable

Данный метод позволяет создать новую таблицу в базе данных.

Описание метода:

```
/**
 * Создать таблицу
 * @param string $name - название
 * @param array $args - массив колонок и их спецификация
 *
 * Supported Keys:
 *
 * [bool] if_not_exists - добавить флаг 'IF NOT EXISTS' (только для таблицы)
 * [bool] id - использовать колонку ID или нет
 * [string] type - тип данных колонки (обязательный)
 * [integer] limit - размер данных колонки
 * [bool] null - может ли быть NULL
 * [string] default - базовое значение
 * [bool] primary_key - использовать как первичный ключ
 * [bool] unique - является уникальным
 * [string] unique_group - является уникальной группой (значение: имя группы)
 */
final protected function createTable(string $name, array $args) {...}
```

ПРИМЕЧАНИЕ: различия ключей аргументов «unique» И «unique_group»:

- «unique» - указывает, что значения в колонке данных должны быть уникальными;
- «unique_group» - указывает, что колонка входит в массив уникальных колонок, имя которого задается в аргументах (SQL: «UNIQUE(column_1, column_2, column_3)»).

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        $this->createTable('test',
            [
                'id' => false,
                'if_not_exists' => true,
                'my_id' => [ 'type' => 'integer', 'null' => false, 'primary_key' => true ],
                'my_data' => [ 'type' => 'integer', 'null' => false, 'unique_group' => 'a' ],
                'my_data_2' => [ 'type' => 'string', 'limit' => 128, 'default' => '', 'unique_group' => 'a' ],
                'my_data_3' => [ 'type' => 'string', 'default' => '', 'unique' => true ]
            ]);
    }
}
```

SQL запрос примера выше для PostgreSQL:

```
CREATE TABLE public.test
(
    my_id integer NOT NULL,
    my_data integer NOT NULL,
    my_data_2 character varying(128) DEFAULT ''::character varying,
    my_data_3 text DEFAULT ''::text,
    CONSTRAINT test_pkey PRIMARY KEY (my_id),
    CONSTRAINT test_my_data_3_key UNIQUE (my_data_3),
    CONSTRAINT test_my_data_my_data_2_key UNIQUE (my_data, my_data_2)
)
```

12.2.7. renameTable

Данный метод позволяет переименовать созданную ранее таблицу в базе данных.

Описание метода:

```
/**
 * Переименовать таблицу
 * @param string $name - имя
 * @param string $newName - новое имя
 */
final protected function renameTable(string $name, string $newName) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        $this->renameTable('test', 'old_test');
    }
}
```

12.2.8. dropTable

Данный метод позволяет удалить созданную ранее таблицу из базы данных.

Описание метода:

```
/**  
 * Удалить таблицу  
 * @param string $name - название  
 * @param array $props - свойства  
 *  
 * Supported Props:  
 *  
 * [bool] if_exists - добавить флаг 'IF EXISTS'  
 * [bool] cascade - добавить флаг 'CASCADE'  
 */  
final protected function dropTable(string $name, array $props = []) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function up() {  
        //  
        // Delete table without check 'IF EXISTS' and with 'CASCADE'  
        //  
        // If the "old_test" table does not exist, then an SQL error will be generated  
        // and the migration install will fail!  
        //  
        $this->dropTable('old_test', [ 'cascade' => true ]);  
  
        //  
        // Delete table with check 'IF EXISTS' and 'CASCADE'  
        //  
        // If the table "test" does not exist, this step is ignored.  
        //  
        $this->dropTable('test', [ 'if_exists' => true, 'cascade' => true ]);  
    }  
}
```

12.2.9. addColumn

Данный метод позволяет добавить новую колонку данных в созданную ранее таблицу базы данных.

Описание метода:

```
/**  
 * Добавить колонку в таблицу  
 * @param string $table - название таблицы  
 * @param string $column - название колонки  
 * @param array $props - свойства  
 *  
 * Supported Props:  
 *  
 * [bool] if_not_exists - добавить флаг 'IF NOT EXISTS'  
 * [string] type - тип данных колонки (обязательный)  
 * [integer] limit - размер данных колонки  
 * [bool] null - может ли быть NULL  
 * [string] default - базовое значение  
 */  
final protected function addColumn(string $table, string $column, array $props = []) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        //
        // Append column "second_column" in table "old_test".
        //
        $this->addColumn('old_test', 'second_column', [
            'if_not_exists' => true,
            'type' => 'integer',
            'default' => 0,
            'null' => false
        ]);
    }
}
```

12.2.10. renameColumn

Данный метод позволяет переименовать существующую колонку данных в таблице базы данных. Все связанные с данной колонкой объекты, вторичные ключи и прочие элементы базы данных автоматически обновляются с учетом нового имени.

Описание метода:

```
/***
 * Переименовать колонку в таблице
 * @param string $table - название таблицы
 * @param string $column - название колонки
 * @param string $newName - новое название колонки
 */
final protected function renameColumn(string $table, string $column, string $newName) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        //
        // Rename column "second_column" in table "old_test".
        //
        $this->renameColumn('old_test', 'second_column', 'old_second_column');
    }
}
```

12.2.11. changeColumn

Данный метод позволяет изменить свойства существующей колонки данных в таблице базы данных.

Описание метода:

```
/**  
 * Изменить тип колонки и ее дополнительные параметры, если они заданы  
 * @param string $table - название таблицы  
 * @param string $column - название колонки  
 * @param string $type - новый тип данных  
 * @param array $props - свойства  
 *  
 * Supported Props:  
 *  
 * [integer] limit - размер данных колонки  
 * [bool] null - может ли быть NULL  
 * [string] default - базовое значение  
 */  
final protected function changeColumn(string $table, string $column, string $type, array $props = []) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function up() {  
        //  
        // Change column "old_second_column" in table "old_test".  
        // Change:  
        // - column type: int -> string  
        // - column default value  
        // - add column check 'not null'  
        //  
        $this->changeColumn('old_test', 'old_second_column', 'string', [  
            'default' => 'no data',  
            'null' => false  
        ]);  
    }  
}
```

12.2.12. changeColumnDefault

Данный метод позволяет изменить или удалить свойство «DEFAULT» существующей колонки данных в таблице базы данных.

Описание метода:

```
/**  
 * Изменить/Удалить секцию DEFAULT у колонки  
 * @param string $table - название таблицы  
 * @param string $column - название колонки  
 * @param $default - значение секции DEFAULT (если null -> секция DEFAULT удаляется)  
 */  
final protected function changeColumnDefault(string $table, string $column, $default = null)  
{...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        //
        // Change column default value.
        //
        $this->changeColumnDefault('old_test', 'old_second_column', 'new default data');

        //
        // Delete column default flag.
        //
        $this->changeColumnDefault('old_test', 'old_second_column');
    }
}
```

12.2.13. changeColumnNull

Данный метод позволяет изменить или удалить свойство «NOT NULL» существующей колонки данных в таблице базы данных.

Описание метода:

```
/**
 * Добавить/Удалить секцию NOT NULL у колонки
 * @param string $table - название таблицы
 * @param string $column - название колонки
 * @param bool $notNull - значение секции (если false - секция NOT NULL удаляется)
 */
final protected function changeColumnNull(string $table, string $column, $notNull = false) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        //
        // Add column check 'NOT NULL'.
        //
        $this->changeColumnNull('old_test', 'old_second_column', true);

        //
        // Delete column check 'NOT NULL'.
        //
        $this->changeColumnNull('old_test', 'old_second_column');
    }
}
```

12.2.14. dropColumn

Данный метод позволяет удалить существующую колонку данных из таблицы базы данных.

Описание метода:

```
/**  
 * Удалить колонку из таблицы.  
 * @param string $table - название таблицы  
 * @param string $column - название колонки  
 */  
final protected function dropColumn(string $table, string $column) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function up() {  
        //  
        // Delete column "old_second_column" from table "old_test".  
        //  
        $this->dropColumn('old_test', 'old_second_column');  
    }  
}
```

12.2.15. addIndex

Данный метод позволяет добавить новый индекс для существующей таблицы базы данных.

Описание метода:

```
/**  
 * Добавить индекс для таблицы  
 * @param string $table - название таблицы  
 * @param array $column - названия колонок  
 * @param array $props - свойства  
 *  
 * Supported Props:  
 *  
 * [string] name - имя индекса (необязательное)  
 * [bool] unique - является ли уникальным  
 */  
final protected function addIndex(string $table, array $column, array $props = []) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function up() {  
        //  
        // Add a new index of the table "old_test" on the column "old_second_column".  
        //  
        $this->addIndex('old_test', [ 'old_second_column' ]);  
        //  
        // Add a new index of the table "old_test" on the column "old_second_column"  
        // with index name "my_index".  
        //  
        $this->addIndex('old_test', [ 'old_second_column' ], [ 'name' => 'my_index' ]);  
    }  
}
```

12.2.16. renameIndex

Данный метод позволяет переименовать существующий индекс таблицы базы данных. Все связанные с данным индексом объекты, вторичные ключи и прочие элементы базы данных автоматически обновляются с учетом нового имени.

Описание метода:

```
/**  
 * Переименовать индекс для таблицы  
 * @param string $table - название таблицы  
 * @param string $oldName - старое название  
 * @param string $newName - новое название  
 */  
final protected function renameIndex(string $table, string $oldName, string $newName) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration  
{  
    final public function up() {  
        //  
        // Rename table index: "my_index" -> "my_new_index".  
        //  
        $this->renameIndex('old_test', 'my_index', 'my_new_index');  
    }  
}
```

12.2.17. dropIndex

Данный метод позволяет удалить существующий индекс из таблицы базы данных.

Описание метода:

```
/**  
 * Удалить индекс таблицы  
 * @param string $table - название таблицы  
 * @param array $props - свойства  
 *  
 * Supported Props:  
 *  
 * [string] column - имя колонки  
 * [string] name - имя индекса  
 * [bool] if_exists - добавить флаг 'IF EXISTS' (может не поддерживаться)  
 * [bool] cascade - добавить флаг 'CASCADE' (может не поддерживаться)  
 *  
 * NOTE: Должен быть задан хотябы один!  
 * NOTE: Приоритет отдается name!  
 */  
final protected function dropIndex(string $table, array $props = []) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        //
        // Delete table index by column name.
        //
        $this->dropIndex('old_test', [ 'column' => 'old_second_column' ]);

        //
        // Delete table index by index name.
        //
        $this->dropIndex('old_test', [ 'name' => 'my_new_index' ]);

        //
        // Delete table index by index name.
        // Column name is ignored!
        //
        $this->dropIndex('old_test', [
            'column' => 'old_second_column',
            'name' => 'my_new_index',
            'if_exists' => true
        ]);
    }
}
```

12.2.18. setPrimaryKey

Данный метод позволяет задать новый первичный ключ таблицы базы данных. Колонка для нового первичного ключа уже должна присутствовать в таблице.

Описание метода:

```
/**
 * Установить новый первичный ключ таблицы
 * @param string $table - название таблицы
 * @param string $column - название колонки
 */
final protected function setPrimaryKey(string $table, string $column) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        //
        // Set new primary key in table: "my_id" -> "old_second_column".
        //
        $this->setPrimaryKey('old_test', 'old_second_column');
    }
}
```

12.2.19. dropPrimaryKey

Данный метод позволяет удалить первичный ключ таблицы базы данных.

Описание метода:

```
/**  
 * Удалить первичный ключ таблицы  
 * @param string $table - название таблицы  
 * @param string $column - название колонки  
 */  
final protected function dropPrimaryKey(string $table, string $column) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function up() {  
        //  
        // Delete primary key from table "old_test".  
        //  
        $this->dropPrimaryKey('old_test', 'old_second_column');  
    }  
}
```

12.2.20. addForeignKey

Данный метод позволяет добавить вторичный ключ для таблицы базы данных. Таблица и её колонки, на которые будет ссылаться ключ, уже должны присутствовать в базе данных, а типы колонок должны совпадать. Для MySQL/MariaDB типы данных должны иметь размерность (пример: varchar(128), bigint и т.д.).

Описание метода:

```
/**  
 * Добавить вторичный ключ для таблицы  
 * @param string $table - название таблицы  
 * @param array $columns - названия колонок  
 * @param string $refTable - название таблицы на которую ссылаемся  
 * @param array $refColumns - названия колонок на которые ссылаемся  
 * @param array $props - свойства  
 *  
 * Supported Props:  
 *  
 * [bool] on_update - добавить флаг 'ON UPDATE' (может не поддерживаться)  
 * [bool] on_delete - добавить флаг 'ON DELETE' (может не поддерживаться)  
 * [string] action - добавить флаг поведения 'NO ACTION / CASCADE / RESTRICT /  
 *                   SET DEFAULT / SET NULL' (может не поддерживаться)  
 * [string] name - задать имя вторичного ключа  
 */  
final protected function addForeignKey(string $table, array $columns,  
                                      string $refTable, array $refColumns,  
                                      array $props = [])  
{...}
```

ПРИМЕЧАНИЕ: дополнительные флаги вторичного ключа «ON UPDATE», «ON DELETE», «CASCADE» и «RESTRICT» и прочие могут не поддерживаться СУБД. В этом случае они будут проигнорированы.

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        //
        // Create new table "test_table_2".
        //
        $this->createTable('test_table_2',
            [
                'id' => true,
                'if_not_exists' => true,
                'my_data' => [ 'type' => 'string', 'default' => '', 'unique' => true ]
            ]);
        //
        // Append foreign key from table "old_test" to table "test_table_2".
        //
        $this->addForeignKey(
            'old_test', [ 'my_data' ],
            'test_table_2', [ 'id' ],
            [ 'on_delete' => true, 'action' => 'cascade' ]
        );
    }
}
```

12.2.21. addForeignKeyPKey

Данный метод позволяет добавить вторичный ключ для таблицы базы данных, который будет ссылаться на первичной ключ требуемой таблицы. Таблица, на которую будет ссылаться ключ, уже должна присутствовать в базе данных, а типы колонок должны совпадать.

Описание метода:

```
/**
 * Добавить вторичный ключ для таблицы, ссылающийся на первичный ключ другой таблицы
 * @param string $table - название таблицы
 * @param string $column - название колонки
 * @param string $refTable - название таблицы на которую ссылаемся
 * @param array $props - свойства
 *
 * NOTE: данный метод создает вторичный ключ, который будет ссылаться на первичный ключ таблицы $refTable.
 *
 * Supported Props:
 *
 * [bool] on_update - добавить флаг 'ON UPDATE' (может не поддерживаться)
 * [bool] on_delete - добавить флаг 'ON DELETE' (может не поддерживаться)
 * [string] action - добавить флаг поведения 'NO ACTION / CASCADE / RESTRICT /
 *                   SET DEFAULT / SET NULL' (может не поддерживаться)
 * [string] name - задать имя вторичного ключа
 */
final protected function addForeignKeyPKey(string $table, string $column,
                                         string $refTable,
                                         array $props = [])
{...}
```

ПРИМЕЧАНИЕ: дополнительные флаги вторичного ключа «ON UPDATE», «ON DELETE», «CASCADE» и «RESTRICT» и прочие могут не поддерживаться СУБД. В этом случае они будут проигнорированы.

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        //
        // Append foreign key from table "old_test" to table "test_table_2".
        //
        $this->addForeignKeyPKey('old_test', 'my_data',
            'test_table_2',
            [ 'on_delete' => true, 'action' => 'cascade' ]
        );
    }
}
```

12.2.22. dropForeignKey

Данный метод позволяет удалить вторичный ключ таблицы базы данных.

Описание метода:

```
/**
 * Удалить вторичный ключ таблицы
 * @param string $table - название таблицы
 * @param array $columns - названия колонок
 */
final protected function dropForeignKey(string $table, array $columns) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration
{
    final public function up() {
        //
        // Delete foreign key from table "old_test" to table "test_table_2".
        //
        $this->dropForeignKey('old_test', [ 'my_data' ]);
    }
}
```

12.2.23. dropForeignKeyPKey

Данный метод позволяет удалить вторичный ключ таблицы базы данных, ссылающийся на первичный ключ другой таблицы.

Описание метода:

```
/**  
 * Удалить вторичный ключ таблицы, ссылающийся на первичный ключ другой таблицы  
 * @param string $table - название таблицы  
 * @param string $column - название колонки  
 */  
final protected function dropForeignKeyPKey(string $table, string $column) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function up() {  
        //  
        // Delete foreign key from table "old_test" to table "test_table_2".  
        //  
        $this->dropForeignKeyPKey('old_test', 'my_data');  
    }  
}
```

12.2.24. execute

Данный метод позволяет выполнить произвольный SQL запрос.

Описание метода:

```
/**  
 * Выполнить SQL запрос  
 * @param string $sql  
 */  
final protected function execute(string $sql) {...}
```

Пример использования:

```
class TestMigration extends \FlyCubePHP\Core\Migration\Migration  
{  
    final public function up() {  
        //  
        // Execute some SQL query.  
        //  
        $this->execute('CREATE EXTENSION IF NOT EXISTS \"uuid-ossp\";');  
    }  
}
```

12.3. Управление базами данных

Исполняемый файл `bin/fly_cube_php`, расположенный в каталоге проекта предоставляет широкий перечень команд для управления текущей схемой базы данных. Перечень доступных команд приводится ниже:

```
MyProject/bin> ./fly_cube_php -h
Usage: ./fly_cube_php [options]
Options include:
--help                                Show this message [-h, -?]
--version                             Print the version [-v]
--output=[true/false]                  Show output (optional)
--env=[VALUE]                           Set current environment (production/development;
                                         default: development)

...
--db-create                            Create database for current environment
--db-create-all                        Create databases for all environments (development and production)
--db-drop                             Drop database for current environment
--db-drop-all                         Drop databases for all environments (development and production)
--db-setup                            Init database for current environment
--db-reset                            Re-Init database for current environment
--db-seed                             Load database Seed.php file
--db-schema-dump                      Create database schema dump
--db-schema-load                      Re-Create database and load schema dump
```

Более подробно эти команды рассматриваются ниже в отдельных главах.

12.3.1. --db-create

Данная команда позволяет создать пустую базу данных для текущего режима работы приложения (environment). Режим работы задается в конфигурационном файле приложения «`[MyProject]/config/application_env.conf`», но его так же можно изменить, используя флаг `--env=[VALUE]`.

Пример работы команды при использовании SQLite 3:

1. вывод содержимого каталога `«[MyProject]/db/»` до выполнения команды:

```
MyProject> ls -l db/
all 4
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate
```

2. выполнение команды по созданию базы данных:

```
MyProject/bin> ./fly_cube_php --db-create
== FlyCubePHP: Create database ==
MigrationsCore: Start create database (name: db/development.sqlite3)
MigrationsCore: Finish create database
== FlyCubePHP =====
```

3. вывод содержимого каталога «[MyProject]/db/» после выполнения команды:

```
MyProject> ls -l db/  
all 4  
-rw-r--r-- 1 test users    0 oct 13 17:48 development.sqlite3  
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate
```

Пример работы команды при использовании SQLite 3 для «production mode», заданный флагом «--env»:

1. вывод содержимого каталога «[MyProject]/db/» до выполнения команды:

```
MyProject> ls -l db/  
all 4  
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate
```

2. выполнение команды по созданию базы данных для «production mode»:

```
MyProject/bin> ./fly_cube_php --db-create --env=production  
== FlyCubePHP: Create database ==  
MigrationsCore: Start create database (name: db/production.sqlite3)  
MigrationsCore: Finish create database  
== FlyCubePHP =====
```

3. вывод содержимого каталога «[MyProject]/db/» после выполнения команды:

```
MyProject> ls -l db/  
all 4  
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate  
-rw-r--r-- 1 test users    0 oct 13 17:56 production.sqlite3
```

12.3.2. --db-create-all

Данная команда позволяет создать пустые базы данных для всех режимов работы приложения (environment). Флаг «--env=[VALUE]» игнорируется.

Пример работы команды при использовании SQLite 3:

1. вывод содержимого каталога «[MyProject]/db/» до выполнения команды:

```
MyProject> ls -l db/  
all 4  
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate
```

2. выполнение команды по созданию баз данных:

```
MyProject/bin> ./fly_cube_php --db-create-all  
==== FlyCubePHP: Create all databases ====  
MigrationsCore: Start create database (name: db/development.sqlite3)  
MigrationsCore: Finish create database  
MigrationsCore: Start create database (name: db/production.sqlite3)  
MigrationsCore: Finish create database  
==== FlyCubePHP =====
```

3. вывод содержимого каталога «[MyProject]/db/» после выполнения команды:

```
MyProject> ls -l db/  
  
all 4  
-rw-r--r-- 1 test users 0 oct 13 18:01 development.sqlite3  
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate  
-rw-r--r-- 1 test users 0 oct 13 18:01 production.sqlite3
```

12.3.3. --db-drop

Данная команда позволяет удалить базу данных для текущего режима работы приложения (environment). Режим работы задается в конфигурационном файле приложения «[MyProject]/config/application_env.conf», но его так же можно изменить, используя флаг «--env=[VALUE]».

Пример работы команды при использовании SQLite 3:

1. вывод содержимого каталога «[MyProject]/db/» до выполнения команды:

```
MyProject> ls -l db/  
  
all 4  
-rw-r--r-- 1 test users 0 oct 13 18:01 development.sqlite3  
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate  
-rw-r--r-- 1 test users 0 oct 13 18:01 production.sqlite3
```

2. выполнение команды по удалению базы данных:

```
MyProject/bin> ./fly_cube_php --db-drop  
==== FlyCubePHP: Drop database ====  
MigrationsCore: Start drop database (name: db/development.sqlite3)  
MigrationsCore: Finish drop database  
==== FlyCubePHP =====
```

3. вывод содержимого каталога «[MyProject]/db/» после выполнения команды:

```
MyProject> ls -l db/  
  
all 4  
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate  
-rw-r--r-- 1 test users 0 oct 13 18:01 production.sqlite3
```

Пример работы команды при использовании SQLite 3 для «production mode», заданный флагом «--env»:

1. вывод содержимого каталога «[MyProject]/db/» до выполнения команды:

```
MyProject> ls -l db/  
  
all 4  
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate  
-rw-r--r-- 1 test users    0 oct 13 18:01 production.sqlite3
```

2. выполнение команды по удалению базы данных для «production mode»:

```
MyProject/bin> ./fly_cube_php --db-drop --env=production  
  
== FlyCubePHP: Drop database ==  
MigrationsCore: Start drop database (name: db/production.sqlite3)  
MigrationsCore: Finish drop database  
== FlyCubePHP =====
```

3. вывод содержимого каталога «[MyProject]/db/» после выполнения команды:

```
MyProject> ls -l db/  
  
all 4  
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate
```

12.3.4. --db-drop-all

Данная команда позволяет удалить базы данных для всех режимов работы приложения (environment). Флаг «--env=[VALUE]» игнорируется.

Пример работы команды при использовании SQLite 3:

1. вывод содержимого каталога «[MyProject]/db/» до выполнения команды:

```
MyProject> ls -l db/  
  
all 4  
-rw-r--r-- 1 test users    0 oct 13 18:01 development.sqlite3  
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate  
-rw-r--r-- 1 test users    0 oct 13 18:01 production.sqlite3
```

2. выполнение команды по удалению баз данных:

```
MyProject/bin> ./fly_cube_php --db-drop-all  
  
== FlyCubePHP: Drop all databases ==  
MigrationsCore: Start drop database (name: db/development.sqlite3)  
MigrationsCore: Finish drop database  
MigrationsCore: Start drop database (name: db/production.sqlite3)  
MigrationsCore: Finish drop database  
== FlyCubePHP =====
```

3. вывод содержимого каталога «[MyProject]/db/» после выполнения команды:

```
MyProject> ls -l db/
all 4
drwxr-xr-x 2 test users 4096 oct 13 12:42 migrate
```

12.3.5. --db-setup

Данная команда позволяет создать базу данных для текущего режима работы приложения (environment) и установить все миграции, используя «слепок» базы данных из файла «[MyProject]/db/Schema.php». Если данный этап успешен, то ядро миграций производит поиск файла «[MyProject]/db/Seed.php» и его загрузку в систему.

Режим работы задается в конфигурационном файле приложения «[MyProject]/config/application_env.conf», но его так же можно изменить, используя флаг «--env=[VALUE]».

Пример работы команды при использовании SQLite:

```
MyProject/bin> ./fly_cube_php --db-setup
== FlyCubePHP: Database setup ==
MigrationsCore: Start load schema dump
MigrationsCore: Finish load schema dump
MigrationsCore: Current migration version: 20211013094233
== FlyCubePHP =====
```

Пример работы команды при использовании SQLite и отсутствии файла «[MyProject]/db/Seed.php»:

```
MyProject/bin> ./fly_cube_php --db-setup
== FlyCubePHP: Database setup ==
MigrationsCore: Start load schema dump
MigrationsCore: Finish load schema dump
MigrationsCore: Current migration version: 20211013094233
MigrationsCore: Not found Seed.php file!
== FlyCubePHP =====
```

Пример работы команды при использовании SQLite 3 и отсутствии необходимых файлов:

```
MyProject/bin> ./fly_cube_php --db-setup
== FlyCubePHP: Database setup ==
MigrationsCore: Not found schema dump file!
MigrationsCore: Not found Seed.php file!
== FlyCubePHP =====
```

12.3.6. --db-reset

Данная команда позволяет пересоздать базу данных для текущего режима работы приложения (environment) и установить все миграции, используя «слепок» базы данных из файла «[MyProject]/db/Schema.php». Если данный этап успешен, то ядро миграций производит поиск файла «[MyProject]/db/Seed.php» и его загрузку в систему.

Режим работы задается в конфигурационном файле приложения «[MyProject]/config/application_env.conf», но его так же можно изменить, используя флаг «--env=[VALUE]».

Пример работы команды при использовании SQLite:

```
MyProject/bin> ./fly_cube_php --db-reset

==== FlyCubePHP: Database reset ====
MigrationsCore: Start drop database (name: db/development.sqlite3)
MigrationsCore: Finish drop database
MigrationsCore: Start load schema dump
MigrationsCore: Finish load schema dump
MigrationsCore: Current migration version: 20211013094233
==== FlyCubePHP =====
```

Пример работы команды при использовании SQLite и отсутствии файла «[MyProject]/db/Seed.php»:

```
MyProject/bin> ./fly_cube_php --db-reset

==== FlyCubePHP: Database reset ====
MigrationsCore: Start drop database (name: db/development.sqlite3)
MigrationsCore: Finish drop database
MigrationsCore: Start load schema dump
MigrationsCore: Finish load schema dump
MigrationsCore: Current migration version: 20211013094233
MigrationsCore: Not found Seed.php file!
==== FlyCubePHP =====
```

Пример работы команды при использовании SQLite 3 и отсутствии необходимых файлов:

```
MyProject/bin> ./fly_cube_php --db-reset

==== FlyCubePHP: Database reset ====
MigrationsCore: Start drop database (name: db/development.sqlite3)
MigrationsCore: Finish drop database
MigrationsCore: Not found schema dump file!
MigrationsCore: Not found Seed.php file!
==== FlyCubePHP =====
```

12.3.7. --db-schema-dump

Данная команда позволяет создать «слепок» базы данных для текущего режима работы приложения (environment) и сохранить его как файла «[MyProject]/db/Schema.php».

Режим работы задается в конфигурационном файле приложения «[MyProject]/config/application_env.conf», но его так же можно изменить, используя флаг «--env=[VALUE]».

Пример работы команды при использовании SQLite:

```
MyProject/bin> ./fly_cube_php --db-schema-dump
== FlyCubePHP: Create database schema dump ==
[Created] db/Schema.php
== FlyCubePHP =====
```

Если база данных пуста, или не создавалась, то содержимое «слепка» будет следующим:

```
<?php
// This file is auto-generated from the current state of the database. Instead
// of editing this file, please use the migrations feature of Active Record to
// incrementally modify your database, and then regenerate this schema definition.
//
// Note that this Schema.php definition is the authoritative source for your
// database schema. If you need to create the application database on another
// system, you should be using 'fly_cube_php --db-schema-load', not running all
// the migrations from scratch. The latter is a flawed and unsustainable approach
// (the more migrations you'll amass, the slower it'll run and the greater likelihood
// for issues).
//
// It's strongly recommended that you check this file into your version control system.
//
class Schema extends \FlyCubePHP\Core\Migration\BaseSchema
{
    public function __construct() {
        parent::__construct();
    }

    final public function up() {

    }
}
```

Если же база данных содержит установленные миграции, то содержимое «слепка» может быть таким:

```
<?php
// This file is auto-generated from the current state of the database. Instead
// of editing this file, please use the migrations feature of Active Record to
// incrementally modify your database, and then regenerate this schema definition.
//
// Note that this Schema.php definition is the authoritative source for your
// database schema. If you need to create the application database on another
// system, you should be using 'fly_cube_php --db-schema-load', not running all
// the migrations from scratch. The latter is a flawed and unsustainable approach
// (the more migrations you'll amass, the slower it'll run and the greater likelihood
// for issues).
//
// It's strongly recommended that you check this file into your version control system.
//
class Schema extends \FlyCubePHP\Core\Migration\BaseSchema
{
    public function __construct() {
        parent::__construct(20210309092620);
    }

    final public function up() {

        $this->createSchema('public', [ 'if_not_exists' => true ]);

        $this->createTable('test', [
            'id' => false,
            'my_id' => [ 'type' => 'integer', 'null' => false, 'primary_key' => true ],
            'my_data' => [ 'type' => 'integer', 'null' => false, 'primary_key' => false ],
            'my_data_2' => [ 'type' => 'character varying (128)', 'null' => true, 'primary_key' =>
false, 'default' => '' ],
            'my_data_3' => [ 'type' => 'text', 'null' => true, 'primary_key' => false, 'default'
=> '' ]
        ]);
        $this->addIndex('test', [ 'my_data_3' ], [ 'name' => 'test_my_data_3_key', 'unique' =>
true ]);
        $this->addIndex('test', [ 'my_data', 'my_data_2' ], [ 'name' =>
'test_my_data_my_data_2_key', 'unique' => true ]);
    }
}
```

12.3.8. --db-schema-load

Данная команда позволяет пересоздать базу данных для текущего режима работы приложения (environment) и установить её «слепок» из файла «[MyProject]/db/Schema.php».

Режим работы задается в конфигурационном файле приложения «[MyProject]/config/application_env.conf», но его так же можно изменить, используя флаг «--env=[VALUE]».

Пример работы команды при использовании SQLite:

```
MyProject/bin> ./fly_cube_php --db-schema-load
== FlyCubePHP: Load database schema dump ==
MigrationsCore: Start load schema dump
MigrationsCore: Finish load schema dump
MigrationsCore: Current migration version: 20210309092620
== FlyCubePHP =====
```

Пример работы команды при использовании SQLite с пустым или некорректным файлом «слепка»:

```
MyProject/bin> ./fly_cube_php --db-schema-load  
==== FlyCubePHP: Load database schema dump ====  
MigrationsCore: Start load schema dump  
MigrationsCore: Invalid schema dump version (version: 0)!  
==== FlyCubePHP =====
```

12.3.9. --db-seed

Данная команда позволяет загрузить в систему файл «[MyProject]/db/Seed.php».

Пример работы команды при использовании SQLite:

```
MyProject/bin> ./fly_cube_php --db-seed  
==== FlyCubePHP: Load database Seed.php ====  
==== FlyCubePHP =====
```

Пример работы команды при использовании SQLite отсутствующим файлом «[MyProject]/db/Seed.php»:

```
MyProject/bin> ./fly_cube_php --db-seed  
==== FlyCubePHP: Load database Seed.php ====  
MigrationsCore: Not found Seed.php file!  
==== FlyCubePHP =====
```

12.4. Управление миграциями

Исполняемый файл `bin/fly_cube_php`, расположенный в каталоге проекта предоставляет широкий перечень команд для управления миграциями. Перечень доступных команд приводится ниже:

```
MyProject/bin> ./fly_cube_php -h
Usage: ./fly_cube_php [options]
Options include:
  --help                                Show this message [-h, -?]
  --version                             Print the version [-v]
  --output=[true/false]                  Show output (optional)
  --env=[VALUE]                          Set current environment (production/development;
                                         default: development)

  ...
  --db-migrate                           Start database migrations
  --db-migrate-redo                     Start re-install last database migration
  --db-migrate-status                   Select migrations status
  --db-rollback                          Start uninstall last database migration
  --db-rollback-all                     Start uninstall all database migrations
  --db-version                          Select current database migration version
  --to-version=[VALUE]                  Set needed migration version (optional;
                                         if 0 - uninstall all migrations)

  --step=[VALUE]                         Set needed number of steps for uninstall (re-install) migrations
                                         (optional; default: 1)
```

Более подробно эти команды рассматриваются ниже в отдельных главах.

12.4.1. --db-migrate

Данная команда позволяет установить все новые миграции, и актуализировать версию базы данных. Если ядро миграций определит, что еще не устанавливалась ни одна миграция, то для установки будут запущены все миграции. Так же с помощью флага `--to-version=YYYYMMDDHHmmSS` можно указать необходимую версию миграции, до которой будет установлена или откочена (rollback) база данных. В случае успешной операции в базу данных сохраняется текущая версия миграций и создается (или обновляется) файл «слепка» текущей схемы базы данных `«[MyProject]/db/Schema.php»`, который так же можно использовать для быстрого развертывания.

ПРИМЕЧАНИЕ: если версия миграции, до которой требуется установка, будет равна «0», то будет выполнен откат (rollback) всех миграций базы данных.

Пример работы команды (установка всех миграций в пустую базу данных):

```
MyProject/bin> ./fly_cube_php --db-migrate

==== FlyCubePHP: Migrate database ====
MigrationsCore: Start migrate from 0
[Up] Migrate to (20211012104949 - 'CreateUsers')
[Up] Migrate to (20211013094233 - 'TestMigration')
MigrationsCore: Finish migrate
MigrationsCore: Current migration version: 20211013094233
[Created] db/Schema.php
==== FlyCubePHP =====
```

Пример работы команды (установка новых миграций и актуализация базы данных):

1. создание новой миграции:

```
MyProject/bin> ./fly_cube_php --new --migration --name=TestMigrationNew

==== FlyCubePHP: Create new migration ===
[Created] db/migrate/20211014092355_TestMigrationNew.php
==== FlyCubePHP =====
```

2. установка новых миграций и актуализация базы данных:

```
MyProject/bin> ./fly_cube_php --db-migrate

==== FlyCubePHP: Migrate database ====
MigrationsCore: Start migrate from 20211013094233
[Skip] Migration (20211012104949 - 'CreateUsers')
[Skip] Migration (20211013094233 - 'TestMigration')
[Up] Migrate to (20211014092355 - 'TestMigrationNew')
MigrationsCore: Finish migrate
MigrationsCore: Current migration version: 20211014092355
[Created] db/Schema.php
==== FlyCubePHP =====
```

Пример работы команды (установка миграций до необходимой версии):

- если требуемая версия больше текущей версии миграций в базе данных:

```
MyProject/bin> ./fly_cube_php --db-migrate --to-version=20211013094233

==== FlyCubePHP: Migrate database ====
MigrationsCore: Start migrate from 20211012104949
[Skip] Migration (20211012104949 - 'CreateUsers')
[Up] Migrate to (20211013094233 - 'TestMigration')
MigrationsCore: Finish migrate
MigrationsCore: Current migration version: 20211013094233
[Created] db/Schema.php
==== FlyCubePHP =====
```

- если требуемая версия меньше текущей версии миграций в базе данных:

```
MyProject/bin> ./fly_cube_php --db-migrate --to-version=20211012104949

==== FlyCubePHP: Migrate database ===
MigrationsCore: Start migrate from 20211014092355
[Down] Migrate from (20211014092355 - 'TestMigrationNew')
[Down] Migrate from (20211013094233 - 'TestMigration')
MigrationsCore: Finish migrate
MigrationsCore: Current migration version: 20211012104949
[Created] db/Schema.php
==== FlyCubePHP =====
```

12.4.2. --db-migrate-redo

Данная команда позволяет переустановить необходимое число последних миграций. Число миграций, которые требуется переустановить, можно задать флагом «`--step=[INT VALUE]`». Если данный флаг не задан, то выполняется переустановка самой последней миграции. В случае успешной операции в базу данных сохраняется текущая версия миграций и создается (или обновляется) файл «слепка» текущей схемы базы данных «`[MyProject]/db/Schema.php`», который так же можно использовать для быстрого развертывания.

Пример работы команды:

```
MyProject/bin> ./fly_cube_php --db-migrate-redo

==== FlyCubePHP: Re-Install database migration ===
MigrationsCore: Start rollback from 20211013094233
[Skip] Migration (20211014092355 - 'TestMigrationNew')
[Down] Migrate from (20211013094233 - 'TestMigration')
MigrationsCore: Finish rollback
MigrationsCore: Current migration version: 20211012104949

MigrationsCore: Start migrate from 20211012104949
[Skip] Migration (20211012104949 - 'CreateUsers')
[Up] Migrate to (20211013094233 - 'TestMigration')
MigrationsCore: Finish migrate
MigrationsCore: Current migration version: 20211013094233
==== FlyCubePHP =====
```

Пример работы команды с указанием числа шагов:

```
MyProject/bin> ./fly_cube_php --db-migrate-redo --step=2

==== FlyCubePHP: Re-Install database migration ===
MigrationsCore: Start rollback from 20211013094233
[Skip] Migration (20211014092355 - 'TestMigrationNew')
[Down] Migrate from (20211013094233 - 'TestMigration')
[Down] Migrate from (20211012104949 - 'CreateUsers')
MigrationsCore: Finish rollback
MigrationsCore: Current migration version: 0

MigrationsCore: Start migrate from 0
[Up] Migrate to (20211012104949 - 'CreateUsers')
[Up] Migrate to (20211013094233 - 'TestMigration')
MigrationsCore: Finish migrate
MigrationsCore: Current migration version: 20211013094233
==== FlyCubePHP =====
```

12.4.3. --db-rollback

Данная команда позволяет откатить (rollback) необходимое число последних миграций. Число миграций, которые требуется откатить, можно задать флагом «`--step=[INT VALUE]`». Если данный флаг не задан, то выполняется откат самой последней миграции.

ПРИМЕЧАНИЕ: при откате миграций файл «слепка» «`[MyProject]/db/Schema.php`» не создается и не обновляется.

Пример работы команды:

```
MyProject/bin> ./fly_cube_php --db-rollback
==== FlyCubePHP: Rollback database ===
MigrationsCore: Start rollback from 20211013094233
[Skip] Migration (20211014092355 - 'TestMigrationNew')
[Down] Migrate from (20211013094233 - 'TestMigration')
MigrationsCore: Finish rollback
MigrationsCore: Current migration version: 20211012104949
==== FlyCubePHP =====
```

Пример работы команды с указанием числа шагов:

```
MyProject/bin> ./fly_cube_php --db-rollback --step=2
==== FlyCubePHP: Rollback database ===
MigrationsCore: Start rollback from 20211014092355
[Down] Migrate from (20211014092355 - 'TestMigrationNew')
[Down] Migrate from (20211013094233 - 'TestMigration')
MigrationsCore: Finish rollback
MigrationsCore: Current migration version: 20211012104949
==== FlyCubePHP =====
```

12.4.4. --db-rollback-all

Данная команда позволяет откатить (rollback) все миграции. Число миграций, заданных флагом «`--step=[INT VALUE]`» игнорируется.

ПРИМЕЧАНИЕ: при откате миграций файл «слепка» «`[MyProject]/db/Schema.php`» не создается и не обновляется.

Пример работы команды:

```
MyProject/bin> ./fly_cube_php --db-rollback-all
==== FlyCubePHP: Rollback all database ===
MigrationsCore: Start rollback from 20211012104949
[Skip] Migration (20211014092355 - 'TestMigrationNew')
[Skip] Migration (20211013094233 - 'TestMigration')
[Down] Migrate from (20211012104949 - 'CreateUsers')
MigrationsCore: Finish rollback
MigrationsCore: Current migration version: 0
==== FlyCubePHP =====
```

12.4.5. --db-migrate-status

Данная команда позволяет получить состояния всех миграций.

Пример работы команды:

```
MyProject/bin> ./fly_cube_php --db-migrate-status  
==== FlyCubePHP: Migrate status ====  
MigrationsCore: Current database version: 20211013094233  
MigrationsCore: Found migration files: 3  
MigrationsCore: Installed in database: 2  
[Installed] Migration (20211012104949 - 'CreateUsers')  
[Installed] Migration (20211013094233 - 'TestMigration')  
[Not Installed] Migration (20211014092355 - 'TestMigrationNew')  
==== FlyCubePHP =====
```

12.4.6. --db-version

Данная команда позволяет запросить текущую версию миграций базы данных для текущего режима работы приложения (environment).

Пример работы команды при использовании SQLite:

```
MyProject/bin> ./fly_cube_php --db-version  
[FlyCubePHP] Current database version: 20211013094233
```

Пример работы команды при использовании SQLite с указанием environment:

```
MyProject/bin> ./fly_cube_php --db-version --env=production  
[FlyCubePHP] Current database version: 0
```

13. Классы каталога Network

Данный каталог ядра FlyCubePHP предоставляет различные классы по работе с сетью, позволяющие упростить разработку. Все network классы располагаются в каталоге «FlyCubePHP/src/Network/», а их подробное описание приводится ниже.

13.1. HttpClient

Данный класс предоставляет методы для отправки HTTP запросов на указанный адрес. В основе класса используется библиотеки cURL. Все методы класса возвращают результат в виде объекта класса HttpResponse.

Перечень доступных методов:

- HttpClient::curlGet - метод отправки запроса HTTP GET;
- HttpClient::curlPost - метод отправки запроса HTTP POST;
- HttpClient::curlPut - метод отправки запроса HTTP PUT;
- HttpClient::curlPatch - метод отправки запроса HTTP PATCH;
- HttpClient::curlDelete - метод отправки запроса HTTP DELETE.

Подробное описание доступных методов:

```
/**
 * Send a GET request using cURL
 * @param string $url to request
 * @param array $data - Values to send
 * @param int $timeoutSec - Time out in seconds
 * @param array $httpHeaders - Additional HTTP headers
 * @param array $cookie - Additional Cookie
 * @param array $curlOptions - Additional options for cURL
 * @return HttpResponse
 *
 * ===== Example
 *
 * $headers = [
 *     'Cache-Control' => 'no-cache',
 *     'X-CSRF-Token' => '1nqfvB9JAriJmFb022EcBwdcyVn0DXHHA8XEnqtNBE/DFZG0/AiHNI1Rtm+1v6ecXQA=='
 * ];
 * $cookie = [
 *     'PHPSESSID' => '5bkCqrV8gt8lqmdiui4aghg47q'
 * ];
 * $data = [
 *     'id' => 123,
 *     'name' => 'test'
 * ];
 * $res = \FlyCubePHP\Network\HttpClient::curlGet('http://127.0.0.1:8080/test_get', $data, 5,
 *                                                 $headers, $cookie);
 * var_dump($res);
 */
```

```

static public function curlGet(string $url,
                             array $data = [],
                             int $timeoutSec = 5,
                             array $httpHeaders = [],
                             array $cookie = [],
                             array $curlOptions = []): HttpResponse {...}

/**
 * Send a POST request using cURL
 * @param string $url - URL to request
 * @param array $data - Values to send
 * @param int $timeoutSec - Time out in seconds
 * @param array $httpHeaders - Additional HTTP headers
 * @param array $cookie - Additional Cookie
 * @param array $curlOptions - Additional options for cURL
 * @return HttpResponse
 *
 * ===== Example
 *
 * $headers = [
 *     'Cache-Control' => 'no-cache',
 *     'X-CSRF-Token' => '1nqfvB9JAriJmFb022EcBwdcyVn0DXHHA8XEnqtNBE/DFZG0/AiHNI1Rtm+1v6ecXQA=='
 * ];
 * $cookie = [
 *     'PHPSESSID' => '5bkcqrV8gtqlqmdiu4aghg47q'
 * ];
 * $data = [
 *     'id' => 123,
 *     'name' => 'test'
 * ];
 * $res = \FlyCubePHP\Network\HttpClient::curlPost('http://127.0.0.1:8080/test_post', $data, 5,
 *                                                 $headers, $cookie);
 * var_dump($res);
 */
static public function curlPost(string $url,
                               array $data = [],
                               int $timeoutSec = 5,
                               array $httpHeaders = [],
                               array $cookie = [],
                               array $curlOptions = []): HttpResponse {...}

/**
 * Send a PUT request using cURL
 * @param string $url - URL to request
 * @param array $data - Values to send
 * @param int $timeoutSec - Time out in seconds
 * @param array $httpHeaders - Additional HTTP headers (key - value array)
 * @param array $cookie - Additional Cookie
 * @param array $curlOptions - Additional options for cURL
 * @return HttpResponse
 *
 * ===== Example
 *
 * $headers = [
 *     'Cache-Control' => 'no-cache',
 *     'X-CSRF-Token' => '1nqfvB9JAriJmFb022EcBwdcyVn0DXHHA8XEnqtNBE/DFZG0/AiHNI1Rtm+1v6ecXQA=='
 * ];
 * $cookie = [
 *     'PHPSESSID' => '5bkcqrV8gtqlqmdiu4aghg47q'
 * ];
 *

```

```

/*
* $data = [
*     'id' => 123,
*     'name' => 'test'
* ];
*$res = \FlyCubePHP\Network\HttpClient::curlPut('http://127.0.0.1:8080/test_put', $data, 5,
*                                                 $headers, $cookie);
* var_dump($res);
*/
static public function curlPut(string $url,
                                array $data = [],
                                int $timeoutSec = 5,
                                array $httpHeaders = [],
                                array $cookie = [],
                                array $curlOptions = []): HttpResponse {...}

/** 
* Send a PATCH request using cURL
* @param string $url - URL to request
* @param array $data - Values to send
* @param int $timeoutSec - Time out in seconds
* @param array $httpHeaders - Additional HTTP headers
* @param array $cookie - Additional Cookie
* @param array $curlOptions - Additional options for cURL
* @return HttpResponse
*
* ===== Example
*
* $headers = [
*     'Cache-Control' => 'no-cache',
*     'X-CSRF-Token' => '1nqfvB9JAriJmFb022ECbwdcyVn0DXHHA8XEnqtNBE/DFZG0/AiHNI1Rtm+1v6ecXQA=='
* ];
*$cookie = [
*     'PHPSESSID' => '5bkcqrV8gtslqmdiui4aghg47q'
* ];
*$data = [
*     'id' => 123,
*     'name' => 'test'
* ];
*$res = \FlyCubePHP\Network\HttpClient::curlPatch('http://127.0.0.1:8080/test_patch', $data, 5,
*                                                 $headers, $cookie);
* var_dump($res);
*/
static public function curlPatch(string $url,
                                 array $data = [],
                                 int $timeoutSec = 5,
                                 array $httpHeaders = [],
                                 array $cookie = [],
                                 array $curlOptions = []): HttpResponse {...}

```

```

/**
 * Send a PATCH request using cURL
 * @param string $url - URL to request
 * @param array $data - Values to send
 * @param int $timeoutSec - Time out in seconds
 * @param array $httpHeaders - Additional HTTP headers
 * @param array $cookie - Additional Cookie
 * @param array $curlOptions - Additional options for cURL
 * @return HttpResponse
 *
 * ===== Example
 *
 * $headers = [
 *     'Cache-Control' => 'no-cache',
 *     'X-CSRF-Token' => '1nqfvB9JAriJmFb022EcBwdcyVn0DXHHA8XEnqtNBE/DFZG0/AiHNI1Rtm+1v6ecXQA=='
 * ];
 * $cookie = [
 *     'PHPSESSID' => '5bkcqrV8gt8lqmdui4aghg47q'
 * ];
 * $data = [
 *     'id' => 123,
 * ];
 * $res = \FlyCubePHP\Network\HttpClient::curlDelete('http://127.0.0.1:8080/test_delete', $data,
 *                                                 5, $headers, $cookie);
 * var_dump($res);
 */
static public function curlDelete(string $url,
                                  array $data = [],
                                  int $timeoutSec = 5,
                                  array $httpHeaders = [],
                                  array $cookie = [],
                                  array $curlOptions = []): HttpResponse {...}

```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Network/HttpClient.php».

13.1.1. HttpResponse

Данный класс описывает результат запроса HttpClient-а и предоставляет следующий набор методов:

- `->url()` - получить URL запроса;
- `->code()` - получить код ответа http;
- `->isHttpInformation(): bool` - проверка, является ли ответ информацией (`code >= 100` и `code < 200`);
- `->isHttpSuccess(): bool` - проверка, является ли ответ успехом (`code >= 200` и `code < 300`);
- `->isHttpRedirection(): bool` - проверка, является ли ответ перенаправлением (`code >= 300` и `code < 400`);
- `->isHttpClientError(): bool` - проверка, является ли ответ ошибкой клиента (`code >= 400` и `code < 500`);

- `->isHttpServerError(): bool` - проверка, является ли ответ ошибкой сервера (`code >= 500` и `code < 600`);
- `->rawHeader(): string` - получить сырой заголовок http из ответа;
- `->hasHeaders(): bool` - проверка, заданы ли заголовки http в ответе;
- `->headers(): array` - получить заданные в ответе заголовки http;
- `->hasHeader(string $name): bool` - проверка, задан ли в ответе требуемый заголовок;
- `->header(string $name): mixed|null` - получить значение заданного в ответе требуемого заголовка;
- `->hasCookie(): bool` - проверка, заданы ли в ответе cookie;
- `->cookie(): array` - получить заданные в ответе cookie;
- `->hasCookieValue(string $key): bool` - проверка, задан ли в ответе требуемый cookie;
- `->cookieValue(string $key): mixed|null` - получить значение заданного в ответе требуемого cookie;
- `->hasBody(): bool` - проверка, задано ли в ответе тело сообщения;
- `->body(): string` - получить тело сообщения ответа;
- `->contentType(): string` - получить Content-Type;
- `->hasError(): bool` - проверка, задана ли ошибка cURL;
- `->error(): string` - получить ошибку cURL.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Network/HttpResponse.php».

13.2. HttpCodes.php

Данный вспомогательный класс позволяет получить информацию о необходимом HTTP коде. Ниже приводится перечень доступных методов:

- `->codeInfo(int $code)` - получить массив с информацией по HTTP коду:
- ```
$httpCodes = new \FlyCubePHP\HelperClasses\HttpCodes();
$httpCodes->codeInfo(404);
// ['title' => '404 Not Found', 'code_title' => 'Ресурс не найден', 'code_text' =>
'Запрашиваемый документ не существует.']
```
- `\HttpCodes::title(int $code)` - получить текст заголовка HTTP кода (статический метод):
- ```
\FlyCubePHP\HelperClasses\HttpCodes::title(404);
// '404 Not Found'
```
- `\HttpCodes::codeTitle(int $code)` - получить короткое описание HTTP кода (статический метод):
- ```
\FlyCubePHP\HelperClasses\HttpCodes::codeTitle(404);
// 'Ресурс не найден'
```

- `HttpCodes::codeText(int $code)` - получить полное описание HTTP кода (статический метод):

```
\FlyCubePHP\HelperClasses\HttpCodes::codeText(404);
// 'Запрашиваемый документ не существует.'
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «`FlyCubePHP/src/Network/HttpCodes.php`».

## 14. Вспомогательные классы

В ядре FlyCubePHP есть набор вспомогательных классов, позволяющих упростить разработку. Все вспомогательные классы располагаются в каталоге «`FlyCubePHP/src/HelperClasses/`», а их подробное описание приводится ниже.

### 14.1. CoreHelper.php

Данный вспомогательный класс предоставляет следующий набор статических методов:

- `rootDir()` - получить путь корневого каталога приложения:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
var_dump(CoreHelper::rootDir());
// "/home/test/FlyCubePHProjects/MyProject"
```

- `buildPath(...$segments)` - собрать путь до файла/каталога:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::buildPath(CoreHelper::rootDir(), "test", "directory");
// "/home/test/FlyCubePHProjects/MyProject/test/directory"
```

- `buildAppPath(...$segments)` - собрать путь до файла/каталога исключая путь корневого каталога приложения:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::buildAppPath(CoreHelper::rootDir(), "test", "directory");
// "test/directory"
```

- `getEnv(string $key, string $def = "")` - получить значение переменной окружения:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::getEnv("LANG");
// "ru_RU.UTF-8"
```

- `getEnvLocal(string $key, string $def = "")` - получить значение локальной переменной окружения, выставленной системой или методом «`putenv(...)`»:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::getEnvLocal("LANG");
// "ru_RU.UTF-8"
```

- `prettyName(string $str)` - преобразовать строку в формат «pretty name»:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::prettyName("TestStringValue");
// "Test String Value"
```

- `underscore(string $str)` - преобразовать строку в формат «underscore»:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::underscore("TestStringValue");
// "test_string_value"
```

- `camelcase(string $str, bool $incFirst = true)` - преобразовать строку в формат «camelcase», где «\$incFirst» указывает методу включать первое слово для преобразования или нет:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::camelcase("test-string-value");
// "TestStringValue"

CoreHelper::camelcase("test_string_value");
// "TestStringValue"

CoreHelper::camelcase("test-string-value", false);
// "testStringValue"

CoreHelper::camelcase("test_string_value", false);
// "testStringValue"

CoreHelper::camelcase("test string value", false);
// "testStringValue"
```

- `makeEvenLength(string $val, int $maxLength = -1, string $type = "after")` - создать строку заданного размера:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;

// Set the required string size to 10 characters.
CoreHelper::makeEvenLength("test", 10);
// "test "

// Set the required line size to 10 characters
// and add blank lines at the beginning of the word.
CoreHelper::makeEvenLength("test", 10, "before");
// " test"

// The string size is calculated automatically from the word size.
CoreHelper::makeEvenLength("test");
// "test"

// The line size is ignored because it is less than the word size.
CoreHelper::makeEvenLength("test", 1);
// "test"
```

- `scanDir(string $dir, bool $recursive = false, bool $appendDirs = false)` - сканировать каталог:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;

// Scanning without recursion and directories.
CoreHelper::scanDir(CoreHelper::rootDir());
// "/home/test/FlyCubePHProjects/MyProject/index.php"
// "/home/test/FlyCubePHProjects/MyProject/my_project.apache24.conf"

// Scanning without recursion, but with directories.
CoreHelper::scanDir(CoreHelper::rootDir(), false, true);
// "/home/test/FlyCubePHProjects/MyProject/app"
// "/home/test/FlyCubePHProjects/MyProject/bin"
// "/home/test/FlyCubePHProjects/MyProject/config"
// "/home/test/FlyCubePHProjects/MyProject/db"
// "/home/test/FlyCubePHProjects/MyProject/extensions"
// "/home/test/FlyCubePHProjects/MyProject/index.php"
// "/home/test/FlyCubePHProjects/MyProject/lib"
// "/home/test/FlyCubePHProjects/MyProject/log"
// "/home/test/FlyCubePHProjects/MyProject/my_project.apache24.conf"
// "/home/test/FlyCubePHProjects/MyProject/plugins"
// "/home/test/FlyCubePHProjects/MyProject/tmp"
// "/home/test/FlyCubePHProjects/MyProject/vendor"

// Scanning with recursion and directories.
CoreHelper::scanDir(CoreHelper::rootDir(), true, true);
// "/home/test/FlyCubePHProjects/MyProject/app"
// "/home/test/FlyCubePHProjects/MyProject/app/assets"
// "/home/test/FlyCubePHProjects/MyProject/app/assets/images"
// "/home/test/FlyCubePHProjects/MyProject/app/assets/javascripts"
// "/home/test/FlyCubePHProjects/MyProject/app/assets/javascripts/application.js"
// "/home/test/FlyCubePHProjects/MyProject/app/assets/javascripts/test.js.php"
// "/home/test/FlyCubePHProjects/MyProject/app/assets/stylesheets"
// "/home/test/FlyCubePHProjects/MyProject/app/assets/stylesheets/application.css"
// "/home/test/FlyCubePHProjects/MyProject/app/assets/stylesheets/test.scss"
// "/home/test/FlyCubePHProjects/MyProject/app/controllers"
// "/home/test/FlyCubePHProjects/MyProject/app/controllers/ApiController.php"
// "/home/test/FlyCubePHProjects/MyProject/app/controllers/ApplicationController.php"
// "/home/test/FlyCubePHProjects/MyProject/app/controllers/ApplicationControllerAPI.php"
// etc...
```

- `makeDir(string $path, $mode = 0777, $recursive = false)` - создать каталог:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;

// Make new directory without recursion.
CoreHelper::makeDir(CoreHelper::rootDir() . "/test");
// New dir: "/home/test/FlyCubePHProjects/MyProject/test"

// Make new directory with recursion.
CoreHelper::makeDir(CoreHelper::rootDir() . "/test_1/test_2", 0777, true);
// New dir: "/home/test/FlyCubePHProjects/MyProject/test_1"
// New dir: "/home/test/FlyCubePHProjects/MyProject/test_1/test_2"
```

- `dirName(string $path)` - получить путь до файла без его имени:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::dirName("assets/application_modules/test.js");
// "assets/application_modules"
```

- `fileName(string $path, bool $removeExt = false)` - получить имя файла:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;

// Get file name with extension.
CoreHelper::fileName("assets/application_modules/test.js");
// "test.js"

// Get file name without extension.
CoreHelper::fileName("assets/application_modules/test.js", true);
// "test"
CoreHelper::fileName("assets/application_modules/test.js.php", true);
// "test.js"
```

- `splicePathFirst(string $path)` - обрезать путь в начале строки (удалить первый символ разделения каталогов):

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::splicePathFirst("/assets/application_modules/");
// "assets/application_modules/"
```

- `splicePathLast(string $path)` - обрезать путь в конце строки (удалить последний символ разделения каталогов):

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::splicePathLast("/assets/application_modules/");
// "/assets/application_modules"
```

- `makeValidUrl(string $uri)` - сформировать корректную строку адреса с добавлением URL префикса приложения, если он задан:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::makeValidUrl("/api/test_api");
// if url-prefix not set => "/api/test_api"
// if url-prefix set ("my_project") => "/my_project/api/test_api"
```

- `spliceUrlFirst(string $uri)` - обрезать URL в начале строки (удалить первый символ «/»):

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::spliceUrlFirst("/api/v2");
// "api/v2/"
```

- `spliceUrlLast(string $uri)` - обрезать URL в конце строки (удалить последний символ «/»):

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::spliceUrlLast("/api/v2");
// "/api/v2"
```

- `toBool($val)` - преобразовать значение в bool:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;

// Convert from string to bool.
CoreHelper::toBool("true");
CoreHelper::toBool(" true "); // used auto-trim
CoreHelper::toBool("TRUE"); // used auto-lower
CoreHelper::toBool(" TRUE "); // used auto-trim and auto-lower
// true

// Convert from bool to bool.
CoreHelper::toBool(true);
// true

// Convert from int to bool.
CoreHelper::toBool(1);
// true

// NOTE: in all other cases, will be returned "false".
```

- `boolToStr($val)` - преобразовать значение в строковое представление bool:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;

// Convert from some value to string.
CoreHelper::boolToStr("true");
CoreHelper::boolToStr(" true "); // used auto-trim
CoreHelper::boolToStr("TRUE"); // used auto-lower
CoreHelper::boolToStr(" TRUE "); // used auto-trim and auto-lower
CoreHelper::boolToStr(true);
CoreHelper::boolToStr(1);
// "true"
```

- `uuid()` - сгенерировать UUID:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::uuid();
// "3f6372f0-c9cc-4c20-8fb5-ed5b2478f74c"
```

- `arrayIsList(array $arr): bool` - метод проверки является ли массив списком:

```
use \FlyCubePHP\HelperClasses\CoreHelper as CoreHelper;
CoreHelper::arrayIsList(['a', 'b', 'c']);
// true
CoreHelper::arrayIsList([1 => 'a', 2 => 'b', 3 => 'c']);
// true
CoreHelper::arrayIsList(['key_1' => 'a', 'key_2' => 'b', 'key_3' => 'c']);
// false
CoreHelper::arrayIsList(['key_1' => 'a', 123 => 'b', 'key_3' => 'c']);
// false
CoreHelper::arrayIsList([1 => 'a', 123 => 'b', 3 => 'c']);
// false
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «`FlyCubePHP/src/HelperClasses/CoreHelper.php`».

## 14.2. Enum.php

Данный вспомогательный класс является абстрактным и предоставляет набор методов по работе с перечислениями. Пример использования:

```
class BCState extends \FlyCubePHP\HelperClasses\Enum {
 const NO_STATE = 0;
 const INIT_SUCCESS = 1;
 const INIT_FAILED = 2;
}
```

Перечень доступных статических методов абстрактного класса «\FlyCubePHP\HelperClasses\Enum»:

- `toString(string $enumType, int $enumValue)` - преобразовать значение перечисления в строковое представление:

```
var_dump(BCState::toString("BCState", 0));
// "BCState::NO_STATE"

var_dump(BCState::toString("BCState", BCState::NO_STATE));
// "NO_STATE"
```

- `fromString(string $string)` - преобразовать строковое представление значения перечисления в число:

```
var_dump(BCState::fromString("BCState::NO_STATE"));
// 0
```

- `label(string $enumType, int $enumValue)` - получить короткое название строкового представления значения перечисления:

```
var_dump(BCState::label("BCState", 0));
// "NO_STATE"

var_dump(BCState::label("BCState", BCState::NO_STATE));
// "NO_STATE"
```

- `isValidValue(string $enumType, int $enumValue)` - проверить корректность числового значения перечисления:

```
var_dump(BCState::isValidValue("BCState", 0));
// true

var_dump(BCState::isValidValue("BCState", 5));
// false
```

- `isValidLabel(string $enumType, string $enumValue)` - проверить корректность короткого названия строкового представления значения перечисления:

```
var_dump(BCState::isValidLabel("BCState", "NO_STATE"));
// true

var_dump(BCState::isValidLabel("BCState", "no_state"));
// true

var_dump(BCState::isValidLabel("BCState", "no state"));
// false

var_dump(BCState::isValidLabel("BCState", "NO STATE"));
// false
```

Описание базового абстрактного класса можно найти в файле «FlyCubePHP/src/HelperClasses/Enum.php».

## 14.3. MimeType.php

Данный вспомогательный класс позволяет получить информацию о необходимом mime-типе, используя расширение файла. Поиск типов производится на основе анализа файла «/etc/mime.types». Ниже приводится перечень доступных методов:

- `->resolveMimeType(string $ext)` - поиск mime-типа по расширению файла:

```
$mTypes = new \FlyCubePHP\HelperClasses\MimeTypes();
$mTypes->resolveMimeType('xml');
// 'text/xml'
```

- `MimeType::mimeType(string $ext)` - поиск mime-типа по расширению файла (статический метод):

```
\FlyCubePHP\HelperClasses\MimeTypes::mimeType('xml');
// 'text/xml'
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «`FlyCubePHP/src/HelperClasses/MimeTypes.php`».

## 15. Вспомогательные методы шаблонов представлений (views helpers)

Вся система рендеринга страниц во FlyCubePHP построена на основе библиотеки Twig. Для упрощения разработки шаблонов страниц, FlyCubePHP предоставляет целый ряд дополнительных вспомогательных классов с их методами, регистрируемых в ядре Twig. Более подробно каждый вспомогательный метод рассматривается ниже.

Все шаблоны страниц регистрируются в Twig с учетом своих областей видимости (namespace), имя которого формируется из названия каталога, в котором расположен шаблон.

Пример подключения стороннего шаблона:

```
// Шаблон "test.html" расположжен по пути "[MyProject]/app/views/test/"
// Подключения данного шаблона в другом шаблоне:
{% include '@test/test.html' %}
```

## 15.1. Вставка данных в блок HTML Head

Для того чтобы выполнить вставку из произвольного шаблона в блок «`<head>...</head>`» результирующей страницы можно воспользоваться следующей конструкцией:

```
{% block head %}
{{ add_plugin_view_javascripts("test_2") }}
{{ add_plugin_view_stylesheets("test_2") }}
{% endblock %}
```

где:

- `add_plugin_view_javascripts("test_2")` - подключение файла «`test_2.js`»;
- `add_plugin_view_stylesheets("test_2")` - подключение файла «`test_2.scss`».

Пример кода страницы с добавленными файлами:

```
<!DOCTYPE html>
<html>
<head>
 <!--<link rel='icon' type='image/png' href=' ' /-->
 <title>MyProject</title>

 <meta name="csrf-param" content="authenticity_token" />
 <meta name="csrf-token"
content="A391QKMTMqwKrEAT8Qh4KYBo9LCA3kqx+c24rMFy08VCgT9vnd3mrlvInfwH55pE/5SN68akC8NL7ahlJhs6sg==" />

 <link rel="stylesheet" href="/my_project/assets/test-
c47a6de8e5e1aed33735dc6a57f0c5e36ffebc7ef50185aee75f57392fc346b7.css" />

 <link rel="stylesheet" href="/my_project/assets/application-
df251fc9b6c2f81666943ce159abddcacfa55b7573316bc0f17ed62b28a9086.css" />

 <script src="/my_project/assets/fly-cube-php-ujs-
91ea360e873b5d7f93df5d2d31e7129f07489908cca92e8ddaec6d6516736d15.js"></script>

 <script src="/my_project/assets/test-
c47a6de8e5e1aed33735dc6a57f0c5e36ffebc7ef50185aee75f57392fc346b7.js"></script>

 <script src="/my_project/assets/application-
e96eec5800f8560613e64d4ac3f3478ea3a0c277f34a813447f76f145de82caa.js"></script>

 <script src="/my_project/assets/test_2-
9085835610409af15db48f647d4d44f0c70cadcc66d09bd287236ccbfc55c03c.js"></script>

 <link rel="stylesheet" href="/my_project/assets/test_2-
ce7b6baf866deb1ce0d4966fa3d79b8e0aec6824db6048358e108a3944e58d91.css" />

</head>
<body>
<h1>TestController#root</h1>
<p>Find me in app/views/test</p>
</body>
</html>
```

## 15.2. params - доступ к параметрам

Для доступа к входным параметрам запроса из шаблона страницы можно воспользоваться переменной «`params`», представляющей из себя массив ключ-значение.

Пример использования в шаблоне страницы:

```
<p>Controller: {{ params['controller'] }}</p>
<p>Action: {{ params['action'] }}</p>
```

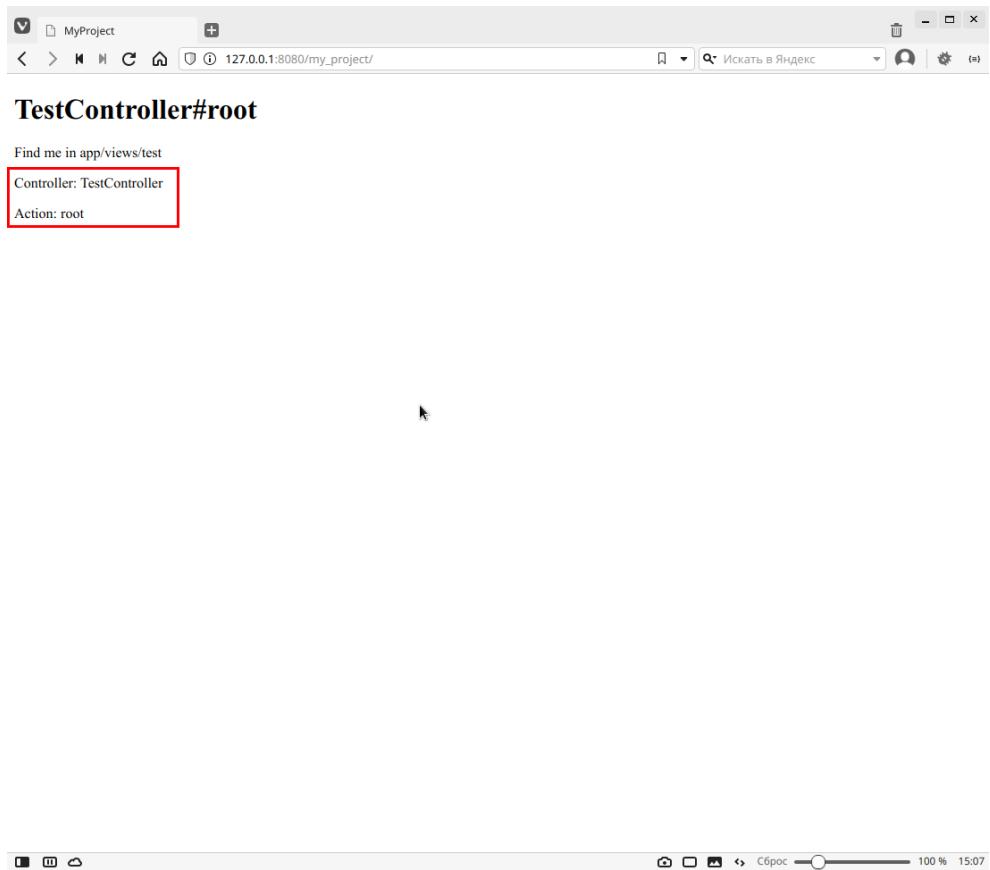


Рис. 22. Пример вывода значений массива параметров.

### 15.3. flash (FlashMessages) - передача данных между запросами

FlashMessages - это специальная часть сессии, которая очищается после доступа к ней во время нового запроса, в отличии от объекта Flash в Ruby on Rails, где очистка происходит при каждом запросе. Более подробно о доступных методах данного класса рассказывалось ранее в главе [FlashMessages - передача данных между запросами](#). Для доступа к объекту данного класса из шаблона страницы можно воспользоваться переменной «flash».

Пример использования в шаблоне страницы:

```
 {{ flash.setNotice('123') }}
<p>FLASH CONTAINS: {{ flash.containsKey('notice') }}</p>
<p>FLASH NOTICE: {{ flash.notice }}</p>
 {{ flash.setValue('my-key', 'my-value') }}
 {% for key, value in flash.allValues %}
 <p>FLASH: {{ key }} = {{ value }}</p>
 {% endfor %}
```

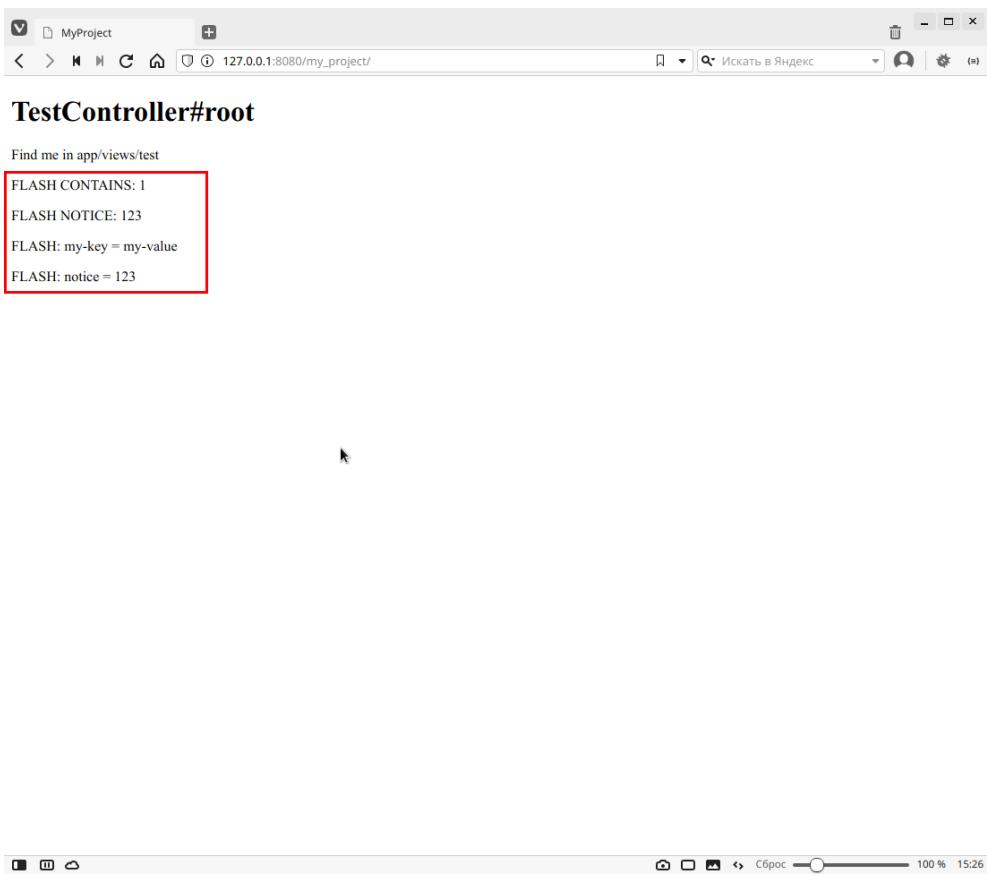


Рис. 23. Пример вывода значений объекта «FlashMessages».

## 15.4. router (RouteCollector) - доступ к методам системы маршрутизации

RouteCollector - основной класс ядра системы маршрутизации фрэймворка. Более подробно о доступных методах данного класса рассказывалось ранее в главе [Система маршрутизации приложения](#). Для доступа к объекту данного класса из шаблона страницы можно воспользоваться переменной «router».

Пример использования в шаблоне страницы:

```
<p>CURRENT URL: {{ router.currentUri }}</p>
<p>CLIENT IP: {{ router.currentClientIP }}</p>
```

## 15.5. AssetTagHelper.php

Данный класс предоставляет базовый набор вспомогательных методов по доступу к файлам ресурсов из шаблона страницы и генерации специальных HTML тегов. Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/Helpers/AssetTagHelper.php».

## 15.5.1. stylesheet\_link\_tag

Данный метод предназначен для добавления на страницу тэга с адресом подключаемого файла стилей.

Описание метода:

```
/**
 * Добавить тэг подключаемого файла стилей
 * @param string $name
 * @param array $options
 * @return string
 */
public function stylesheet_link_tag(string $name, array $options = []): string {...}
```

Пример использования в шаблоне страницы:

```
{{ stylesheet_link_tag("application") }}
```

Результат выполнения на примере кода страницы:

```
<link rel="stylesheet" href="/my_project/assets/test-c47a6de8e5e1aed33735dc6a57f0c5e36ffebc7ef50185aee75f57392fc346b7.css" />
<link rel="stylesheet" href="/my_project/assets/application-df251fc9b6c2f81666943ce159abdcacfa155b7573316bc0f17ed62b28a9086.css" />
```

## 15.5.2. javascript\_include\_tag

Данный метод предназначен для добавления на страницу тэга с адресом подключаемого файла скрипта.

Описание метода:

```
/**
 * Добавить тэг подключаемого файла скрипта
 * @param string $name
 * @param array $options
 * @return string
 */
public function javascript_include_tag(string $name, array $options = []): string {...}
```

Пример использования в шаблоне страницы:

```
{{ javascript_include_tag("application") }}
```

Результат выполнения на примере кода страницы:

```
<script src="/my_project/assets/test-c47a6de8e5e1aed33735dc6a57f0c5e36ffebc7ef50185aee75f57392fc346b7.js"></script>
<script src="/my_project/assets/application-e96eec5800f8560613e64d4ac3f3478ea3a0c277f34a813447f76f145de82caa.js"></script>
```

### 15.5.3. auto\_discovery\_link\_tag

Данный метод возвращает тег ссылки, который браузеры и программы чтения каналов могут использовать для автоматического определения канала RSS, Atom или JSON.

Описание метода:

```
/**
 * Returns a link tag that browsers and feed readers can use to auto-detect an RSS, Atom, or JSON feed.
 * @param string $type
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - rel - Specify the relation of this link, defaults to "alternate"
 * - title - Specify the title of the link, defaults to the +type+
 * - href - Specify the link URL
 * - controller - Specify the application controller class name
 * - action - Specify the application controller action
 *
 * NOTE: 'href' and 'controller + action' are mutually exclusive arguments!
 *
 * ===== Examples in Twig notations
 *
 * auto_discovery_link_tag()
 * * => <link rel="alternate" type="application/rss+xml" title="RSS"
 * href="http://www.currenthonst.com/controller/action" />
 *
 * auto_discovery_link_tag("atom")
 * * => <link rel="alternate" type="application/atom+xml" title="ATOM"
 * href="http://www.currenthonst.com/controller/action" />
 *
 * auto_discovery_link_tag("json")
 * * => <link rel="alternate" type="application/json" title="JSON"
 * href="http://www.currenthonst.com/controller/action" />
 *
 * auto_discovery_link_tag("rss", {"controller": "App", "action": "feed"})
 * * => <link rel="alternate" type="application/rss+xml" title="RSS"
 * href="http://www.currenthonst.com/app/feed" />
 *
 * auto_discovery_link_tag("rss", {"controller": "App", "action": "feed", "title": "My RSS"})
 * * => <link rel="alternate" type="application/rss+xml" title="My RSS"
 * href="http://www.currenthonst.com/app/feed" />
 *
 * auto_discovery_link_tag("rss", {"controller": "news", "action": "feed"})
 * * => <link rel="alternate" type="application/rss+xml" title="RSS"
 * href="http://www.currenthonst.com/news/feed" />
 *
 * auto_discovery_link_tag("rss", {"title": "Example RSS", "href": "http://www.example.com/feed.rss"})
 * * => <link rel="alternate" type="application/rss+xml" title="Example RSS"
 * href="http://www.example.com/feed.rss" />
 *
 */
public function auto_discovery_link_tag(string $type = "rss", array $options = []): string {...}
```

## 15.5.4. favicon\_link\_tag

Данный метод возвращает тег ссылки для favicon, используя для этого Asset Pipeline.

Описание метода:

```
/**
 * Returns a link tag for a favicon managed by the asset pipeline.
 * @param string $source
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - rel - Specify the relation of this link, defaults to "shortcut icon"
 * - type - Specify the type of this icon, defaults to "image/x-icon"
 *
 * ===== Examples in Twig notations
 *
 * favicon_link_tag()
 * * => <link href="/assets/favicon.ico" rel="shortcut icon" type="image/x-icon" />
 *
 * favicon_link_tag("myicon.ico")
 * * => <link href="/assets/myicon.ico" rel="shortcut icon" type="image/x-icon" />
 *
 * Mobile Safari looks for a different link tag, pointing to an image that
 * will be used if you add the page to the home screen of an iOS device.
 * The following call would generate such a tag:
 *
 * favicon_link_tag("mb-icon.png", {"rel": "apple-touch-icon", "type": "image/png"})
 * * => <link href="/assets/mb-icon.png" rel="apple-touch-icon" type="image/png" />
 *
 */
public function favicon_link_tag(string $source = "favicon.ico", array $options = []): string
{...}
```

## 15.5.5. preload\_link\_tag

Данный метод возвращает тег ссылки, который браузеры могут использовать для предварительной загрузки ресурса. Ресурсом может быть путь к ресурсу из Asset Pipeline, полный путь или URI.

Описание метода:

```
/**
 * Returns a link tag that browsers can use to preload the +source+.
 * The +source+ can be the path of a resource managed by asset pipeline, a full path, or an URI.
 * @param string $source
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - type - Override the auto-generated mime type, defaults to the mime type for
 * +source+ extension.
 * - as - Override the auto-generated value for as attribute, calculated using
 * +source+ extension and mime type.
 * - crossorigin - Specify the crossorigin attribute, required to load cross-origin resources.
 *
 * ===== Examples
 *
 * preload_link_tag("custom_theme.css")
 * * => <link rel="preload" href="/assets/custom_theme.css" as="style" type="text/css" />
 *
 * preload_link_tag("worker.js", ["as" => "worker"])
 * * => <link rel="preload" href="/assets/worker.js" as="worker" type="text/javascript" />
 *
 * preload_link_tag("//example.com/font.woff2")
 * * => <link rel="preload" href="//example.com/font.woff2" as="font" type="font/woff2"
 * crossorigin="anonymous"/>
 *
 * preload_link_tag("//example.com/font.woff2", ["crossorigin" => "use-credentials"])
 * * => <link rel="preload" href="//example.com/font.woff2" as="font" type="font/woff2"
 * crossorigin="use-credentials" />
 *
 */
public function preload_link_tag(string $source, array $options = []): string {...}
```

## 15.5.6. image\_tag

Данный метод возвращает тег изображения, поиск которого производится в Asset Pipeline.

Описание метода:

```
/**
 * Добавить тэг изображения
 * @param string $name
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - size - Supplied as "{Width}x{Height}" or "{Number}", so "30x45" becomes
 * width="30" and height="45", and "50" becomes width="50" and height="50".
 * - alt - Set alternative text for image.
 * - height - Set image height.
 * - width - Set image width.
 * - class - Set image class.
 *
 * ===== Examples
 *
 * image_tag("icon.png")
 * =>
 *
 * image_tag("icon.png", {"size": "16x10", "alt": "Edit Entry"})
 * =>
 *
 * image_tag("icons/icon.gif", {"size": "16"})
 * =>
 *
 * image_tag("icons/icon.gif", {"height": "32", "width": "32"})
 * =>
 *
 * image_tag("icons/icon.gif", {"class": "menu_icon"})
 * =>
 */
public function image_tag(string $name, array $options = array()): string {...}
```

## 15.5.7. link\_to

Данный метод возвращает тег ссылки.

Описание метода:

```
/**
 * Добавить тег ссылки
 * @param string $name
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - controller - Set controller name
 * - action - Set controller action name
 * - href - Set link URL
 * - method - Set link HTTP method
 * - class - Set link class
 * - id - Set link ID
 * - target - Set link target
 * - rel - Set link rel
 * - params - Set additional URL params
 *
 * NOTE: 'href' and 'controller + action' are mutually exclusive arguments!
 *
 * ===== Examples in Twig notations
 *
 * link_to("Test Link", {"controller": "AppCore", "action": "test"}), where AppCore::test -> GET
 * => Test Link
 *
 * link_to("Test Link", {"controller": "AppCore", "action": "test_with_id", "params": {"id":123}})
 * where AppCore::test_with_id -> GET (url with params: /test/:id)
 * => Test Link
 *
 * link_to("Test Link", {"controller": "AppCore", "action": "test"})
 * where AppCore::test -> POST/PUT/PATCH/DELETE
 * => Test Link
 *
 * link_to("Test Link", {"href": "/test"})
 * => Test Link
 *
 * link_to("Test Link", {"href": "/test", "method": "post"})
 * => Test Link
 *
 * link_to("Test Link", {"href": "/test", "class": "my-class"})
 * => Test Link
 *
 * link_to("Test Link", {"href": "/test", "class": "my-class", "id": "123"})
 * => Test Link
 *
 * link_to("Test Link", {"href": "/test", "class": "my-class", "id": "123", "target": "_blank"})
 * => Test Link
 *
 * link_to("Test Link", {"href": "/test", "class": "my-class", "id": "123", "rel": "nofollow"})
 * => Test Link
 */
public function link_to(string $name, array $options = array()): string {...}
```

## 15.5.8. add\_plugin\_view\_javascripts

Данный метод предназначен для добавления на страницу тэга с адресом подключаемого файла скрипта в секцию «html -> head». Проверяется наличие определенных флагов ядра, позволяющих выполнить данную операцию. Если флаги не заданы, операция игнорируется.

Описание метода:

```
/**
 * Добавить для загрузки необходимый js скрипт в раздел html->head
 * @param $context
 * @param string $name
 * @return string
 *
 * NOTE: Use only in twig block 'head'!
 *
 * ===== Examples in Twig notations
 *
 * {% block head %}
 * {{ add_plugin_view_javascripts("application") }}
 * {% endblock %}
 *
 */
public function add_plugin_view_javascripts($context, string $name): string {...}
```

## 15.5.9. add\_plugin\_view\_stylesheets

Данный метод предназначен для добавления на страницу тэга с адресом подключаемого файла стилей в секцию «html -> head». Проверяется наличие определенных флагов ядра, позволяющих выполнить данную операцию. Если флаги не заданы, операция игнорируется.

Описание метода:

```
/**
 * Добавить для загрузки необходимый css/scss скрипт в раздел html->head
 * @param $context
 * @param string $name
 * @return string
 *
 * NOTE: Use only in twig block 'head'!
 *
 * ===== Examples in Twig notations
 *
 * {% block head %}
 * {{ add_plugin_view_stylesheets("application") }}
 * {% endblock %}
 *
 */
public function add_plugin_view_stylesheets($context, string $name): string {...}
```

## 15.6. AssetUrlHelper.php

Данный класс предоставляет базовый набор вспомогательных методов по доступу к файлам ресурсов из шаблона страницы. Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/Helpers/AssetUrlHelper.php».

### 15.6.1. asset\_path

Данный метод предназначен для получения пути к файлу ресурса.

Описание метода:

```
/**
 * Computes the path to an some asset.
 * @param string $name
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - skip_asset_pipeline - Skip search in Asset Pipeline
 * - type - Set asset type (js/css/image)
 *
 * ===== Examples in Twig notations
 *
 * asset_path("test")
 * * => "/assets/test.js"
 *
 * asset_path("test", {"type": "css"})
 * * => "/assets/test.css"
 *
 * asset_path("test.js")
 * * => "/assets/test.js"
 *
 * asset_path("test.css")
 * * => "/assets/test.css"
 *
 * asset_path("test", {"skip_asset_pipeline": true})
 * * => "/assets/test"
 *
 * asset_path("test/test.js", {"skip_asset_pipeline": true})
 * * => "/assets/test/test.js"
 *
 * asset_path("/test/test.js")
 * * => "/test/test.js"
 *
 * asset_path("http://www.example.com/test/test.js")
 * * => "http://www.example.com/test/test.js"
 */
public function asset_path(string $name, array $options = []) {...}
```

## 15.6.2. asset\_url

Данный метод предназначен для получения полного пути к файлу ресурса в формате URL строки.

Описание метода:

```
/**
 * Computes the full URL to an some asset.
 * @param string $name
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - skip_asset_pipeline - Skip search in Asset Pipeline
 * - host - Set needed host in URL
 * - protocol - Set needed HTTP protocol in URL
 * - type - Set asset type (js/css/image)
 *
 * ===== Examples in Twig notations
 *
 * asset_url("test", {"host": "www.example.com"})
 * * => "http://www.example.com/assets/test.js"
 *
 * asset_url("test/test_2.js", {"host": "www.example.com", "skip_asset_pipeline": true})
 * * => "http://www.example.com/assets/test/test_2.js"
 *
 * asset_url("test")
 * * => "http://www.my_project.com/assets/test.js"
 *
 * asset_url("test", {"type": "css"})
 * * => "http://www.my_project.com/assets/test.css"
 *
 * asset_url("test.js")
 * * => "http://www.my_project.com/assets/test.js"
 *
 * asset_url("test/test_2.js", {"skip_asset_pipeline": true})
 * * => "http://www.my_project.com/assets/test/test_2.js"
 */
public function asset_url(string $name, array $options = []): string {...}
```

### 15.6.3. javascript\_path

Данный метод предназначен для получения пути к javascript файлу.

Описание метода:

```
/**
 * Computes the path to an javascript asset.
 * @param string $name
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - skip_asset_pipeline - Skip search in Asset Pipeline
 *
 * ===== Examples in Twig notations
 *
 * image_path("test")
 * * => "/assets/test.js"
 *
 * image_path("test/test.js")
 * * => "/assets/test.js"
 *
 * image_path("test", {"skip_asset_pipeline": true})
 * * => "/assets/test"
 *
 * image_path("test/test.js", {"skip_asset_pipeline": true})
 * * => "/assets/test/test.js"
 *
 * image_path("/test/test.js")
 * * => "/test/test.js"
 *
 * image_path("http://www.example.com/test/test.js")
 * * => "http://www.example.com/test/test.js"
 */
public function javascript_path(string $name, array $options = []): string {...}
```

## 15.6.4. javascript\_url

Данный метод предназначен для получения полного пути к javascript файлу ресурса в формате URL строки.

Описание метода:

```
/**
 * Computes the full URL to an javascript asset.
 * @param string $name
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - skip_asset_pipeline - Skip search in Asset Pipeline
 * - host - Set needed host in URL
 * - protocol - Set needed HTTP protocol in URL
 *
 * ===== Examples in Twig notations
 *
 * javascript_url("test", {"host": "www.example.com"})
 * => "http://www.example.com/assets/test.js"
 *
 * javascript_url("test/test_2.js", {"host": "www.example.com", "skip_asset_pipeline": true})
 * => "http://www.example.com/assets/test/test_2.js"
 *
 * javascript_url("test")
 * => "http://www.my_project.com/assets/test.js"
 *
 * javascript_url("test.js")
 * => "http://www.my_project.com/assets/test.js"
 *
 * javascript_url("test/test_2.js", {"skip_asset_pipeline": true})
 * => "http://www.my_project.com/assets/test/test_2.js"
 */
public function javascript_url(string $name, array $options = []): string {...}
```

## 15.6.5. stylesheet\_path

Данный метод предназначен для получения пути к файлу стиля.

Описание метода:

```
/**
 * Computes the path to an stylesheet asset.
 * @param string $name
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - skip_asset_pipeline - Skip search in Asset Pipeline
 *
 * ===== Examples in Twig notations
 *
 * stylesheet_path("test")
 * * => "/assets/test.css"
 *
 * stylesheet_path("test/test.css")
 * * => "/assets/test.css"
 *
 * stylesheet_path("test", {"skip_asset_pipeline": true})
 * * => "/assets/test"
 *
 * stylesheet_path("test/test.css", {"skip_asset_pipeline": true})
 * * => "/assets/test/test.css"
 *
 * stylesheet_path("/test/test.css")
 * * => "/test/test.css"
 *
 * stylesheet_path("http://www.example.com/test/test.css")
 * * => "http://www.example.com/test/test.css"
 *
 */
public function stylesheet_path(string $name, array $options = []): string {...}
```

## 15.6.6. stylesheet\_url

Данный метод предназначен для получения полного пути к файлу стиля в формате URL строки.

Описание метода:

```
/**
 * Computes the full URL to an stylesheet asset.
 * @param string $name
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - skip_asset_pipeline - Skip search in Asset Pipeline
 * - host - Set needed host in URL
 * - protocol - Set needed HTTP protocol in URL
 *
 * ===== Examples in Twig notations
 *
 * stylesheet_url("test", {"host": "www.example.com"})
 * * => "http://www.example.com/assets/test.css"
 *
 * stylesheet_url("test/test_2.css", {"host": "www.example.com", "skip_asset_pipeline": true})
 * * => "http://www.example.com/assets/test/test_2.css"
 *
 * stylesheet_url("test")
 * * => "http://www.my_project.com/assets/test.css"
 *
 * stylesheet_url("test.css")
 * * => "http://www.my_project.com/assets/test.css"
 *
 * stylesheet_url("test/test_2.css", {"skip_asset_pipeline": true})
 * * => "http://www.my_project.com/assets/test/test_2.css"
 */
public function stylesheet_url(string $name, array $options = []): string {...}
```

## 15.6.7. image\_path

Данный метод предназначен для получения пути к файлу изображения.

Описание метода:

```
/**
 * Computes the path to an image asset.
 * @param string $name
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - skip_asset_pipeline - Skip search in Asset Pipeline
 *
 * ===== Examples in Twig notations
 *
 * image_path("edit.png")
 * * => "/assets/edit.png"
 *
 * image_path("icons/edit_2.png")
 * * => "/assets/icons/edit_2.png"
 *
 * image_path("icons/edit_2.png", {"skip_asset_pipeline": true})
 * * => "/assets/icons/edit_2.png"
 *
 * image_path("/icons/edit.png")
 * * => "/icons/edit.png"
 *
 * image_path("http://www.example.com/img/edit.png")
 * * => "http://www.example.com/img/edit.png"
 *
 */
public function image_path(string $name, array $options = []): string {...}
```

## 15.6.8. image\_url

Данный метод предназначен для получения полного пути к файлу изображения в формате URL строки.

Описание метода:

```
/**
 * Computes the full URL to an image asset.
 * @param string $name
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - skip_asset_pipeline - Skip search in Asset Pipeline
 * - host - Set needed host in URL
 * - protocol - Set needed HTTP protocol in URL
 *
 * ===== Examples in Twig notations
 *
 * image_url("edit.png", {"host": "www.example.com"})
 * => "http://www.example.com/assets/edit.png"
 *
 * image_url("icons/edit_2.png", {"host": "www.example.com", "skip_asset_pipeline": true})
 * => "http://www.example.com/assets/icons/edit_2.png"
 *
 * image_url("edit.png")
 * => "http://www.my_project.com/assets/edit.png"
 *
 * image_url("icons/edit_2.png", {"skip_asset_pipeline": true})
 * => "http://www.my_project.com/assets/icons/edit_2.png"
 */
public function image_url(string $name, array $options = []): string {...}
```

## 15.6.9. image\_alt

Данный метод возвращает строку, подходящую для атрибута alt тега изображения HTML.

Описание метода:

```
/**
 * Returns a string suitable for an HTML image tag alt attribute.
 * @param string $name
 * @return string
 *
 * ===== Examples in Twig notations
 *
 * image_alt('rails.png')
 * => Rails
 *
 * image_alt('hyphenated-file-name.png')
 * => Hyphenated file name
 *
 * image_alt('underscored_file_name.png')
 * => Underscored file name
 *
 */
public function image_alt(string $name): string {...}
```

## 15.6.10. make\_valid\_url

Данный метод позволяет сформировать корректную строку адреса с учетом URL-префикса приложения, если он задан.

Описание метода:

```
/**
 * Получить валидную строку адреса (с App-Url-Prefix, если задан)
 * @param string $url - строка URL
 * @return string
 *
 * ===== Examples in Twig notations
 *
 * make_valid_url("/api/test_api");
 * * if url-prefix not set => "/api/test_api"
 * * if url-prefix set ("app1") => "/app1/api/test_api"
 *
 */
public function make_valid_url(string $url): string {...}
```

## 15.7. FormTagHelper.php

Данный класс предоставляет базовый набор вспомогательных методов по гибкому формированию элементов web-страницы. Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/Helpers/FormTagHelper.php».

### 15.7.1. label

Данный метод позволяет добавить на страницу HTML элемент «`<label>...</label>`».

Описание метода:

```
/**
 * Add label tag
 * @param string $text
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set label id
 * - for - Set label for
 * - class - Set label class
 *
 * ===== Examples in Twig notations
 *
 * {{ label('Test text:', {'for': 'my-input'}) }}
 * {{ text_field({'id': 'my-input', 'name': 'my-input-text'}) }}
 *
 * Result:
 * <label for="my-input" >Test text:</label>
 * <input type="text" id="my-input" name="my-input-text" />
 */
public function label(string $text, array $options = []): string {...}
```

### 15.7.2. form\_with

Данный метод позволяет добавить на страницу HTML-форму (`<form>...</form>`). Если в приложении включен режим CSRF Protection, то в форму автоматически добавляется уникальный ключ запроса, если данное поведение не отключено входным аргументом «`authenticity_token`».

## Описание метода:

```
/**
 * Add form tag
 * @param array $options
 * @return string
 * @throws \Exception
 *
 * ===== Options
 *
 * - url - Set form action url
 * - method - Set form method (get/post/put/patch/delete)
 * - authenticity_token - Enable/Disable authenticity token (true/false)
 * - encrypt_type - Set form encrypt type (default: empty)
 * - html_body - Set form html body
 *
 * ===== Examples in Twig notations
 *
 * - Example 1:
 * {% set form_html_body %}
 * <label for="my-input">Test text:</label>
 * <input type="text" id="my-input" name="my-input-text" />
 * <input type="submit" name="commit" value="Commit data">
 * {% endset %}
 * {% set form_url = make_valid_url('test') %}
 * {{ form_with({'html_body': form_html_body, 'url': form_url,
 * 'encrypt_type': 'multipart/form-data'}) }}
 *
 * Result:
 * <form accept-charset="UTF-8" enctype="multipart/form-data" action="/my_project/test"
method="post">
 * <input name="authenticity_token" type="hidden"
value="et2tJwUsPjBz05i6lzlHPHcR1CctzZcpI3Gj/F+YyQ7I+cIO+LqMiK3RVVh1olxjiA+C9rNdy4WrdZGFhdiUw==" />
 * <label for="my-input">Test text:</label>
 * <input type="text" id="my-input" name="my-input-text" />
 * <input type="submit" name="commit" value="Commit data" />
 * </form>
 *
 * - Example 2:
 * {% set form_html_body %}
 * {{ label('Test text:', {'for': 'my-input'}) }}
 * {{ text_field({'id': 'my-input', 'name': 'my-input-text'}) }}
 * {{ submit() }}
 * {% endset %}
 * {% set form_url = make_valid_url('test') %}
 * {{ form_with({'html_body': form_html_body, 'url': form_url,
 * 'encrypt_type': 'multipart/form-data'}) }}
 *
 * Result:
 * <form accept-charset="UTF-8" enctype="multipart/form-data" action="/my_project/test"
method="post">
 * <input name="authenticity_token" type="hidden"
value="et2tJwUsPjBz05i6lzlHPHcR1CctzZcpI3Gj/F+YyQ7I+cIO+LqMiK3RVVh1olxjiA+C9rNdy4WrdZGFhdiUw==" />
 * <label for="my-input" >Test text:</label>
 * <input type="text" id="my-input" name="my-input-text" />
 * <input type="submit" name="commit" value="Commit data" />
 * </form>
 *
 * - Example 3 - disabled authenticity token:
 * {{ form_with({'html_body': form_html_body, 'url': form_url,
 * 'encrypt_type': 'multipart/form-data', 'authenticity_token': false}) }}
 *
 * Result:
 * <form accept-charset="UTF-8" enctype="multipart/form-data" action="/my_project/test"
method="post">
 * <label for="my-input">Test text:</label>
 * <input type="text" id="my-input" name="my-input-text" />
 * <input type="submit" name="commit" value="Commit data" />
 * </form>
 */
public function form_with(array $options = array()): string {...}
```

### 15.7.3. raw

Данный метод позволяет добавить на страницу произвольный HTML блок.

Описание метода:

```
/**
 * Add raw html data
 * @param string $val
 * @return string
 *
 * ===== Examples in Twig notations
 *
 * {{ raw('<label for="my-input" >Test text:</label>') }}
 *
 * Result:
 * <label for="my-input" >Test text:</label>
 */
public function raw(string $val): string {...}
```

### 15.7.4. button\_tag

Данный метод позволяет добавить на страницу HTML элемент <<input type="button" />>.

Описание метода:

```
/**
 * Add input button tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name (default: commit)
 * - class - Set input class
 * - value - Set input value (default: Commit data)
 * - type - Set input type (button/reset/submit) (default: button)
 * - disabled - Set input disabled (default: false)
 *
 * ===== Examples in Twig notations
 *
 * {{ button_tag({'id': 'my-button-id', 'name': 'my-button', 'value': 'some button value',
 'type': 'button'}) }}
 *
 * or
 *
 * {{ button_tag({'id': 'my-button-id', 'name': 'my-button', 'value': 'some button value'}) }}
 *
 * Result:
 * <input type="button" id="my-button-id" name="my-button" value="some button value" />
 */
public function button_tag(array $options = []): string {...}
```

## 15.7.5. submit

Данный метод позволяет добавить на страницу HTML элемент `<input type="submit" />`.

ПРИМЕЧАНИЕ: это вспомогательный метод, вызывающий метод `button_tag(...)`.

Описание метода:

```
/**
 * Add input submit tag button
 * @param array $options
 * @return string
 *
 * NOTE: this overload function -> see button_tag function
 *
 * ===== Examples in Twig notations
 *
 * {{ submit() }}
 *
 * Result:
 * <input type="submit" name="commit" value="Commit data" />
 */
public function submit(array $options = []): string {...}
```

## 15.7.6. file\_field

Данный метод позволяет добавить на страницу HTML элемент `<input type="file" />`.

Описание метода:

```
/**
 * Add input file tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name (default: commit)
 * - class - Set input class
 * - accept - Set to one or multiple mime-types, the user will be suggested a filter when
 * choosing a file.
 * - multiple - Set to true, *in most updated browsers* the user will be allowed to select
 * multiple files (default: false)
 *
 * ===== Examples in Twig notations
 *
 * file_field({'id': 'files', 'multiple': true})
 * * => <input type="file" id="files" name="commit[]" multiple="multiple" />
 *
 * file_field({'id': 'file', 'accept': 'image/png,image/gif,image/jpeg'})
 * * => <input type="file" id="file" name="commit" accept="image/png,image/gif,image/jpeg" />
 */
public function file_field(array $options = []): string {...}
```

## 15.7.7. hidden\_field

Данный метод позволяет добавить на страницу скрытый HTML элемент «`<input type="hidden" />`».

Описание метода:

```
/***
 * Add input hidden tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value
 *
 * ===== Examples in Twig notations
 *
 * hidden_field({'id': 'my-hidden-id', 'name': 'my-hidden-obj', 'value': 'some hidden value'})
 * * => <input type="hidden" id="my-hidden-id" name="my-hidden-obj" value="some hidden value" />
 */
public function hidden_field(array $options = []): string {...}
```

## 15.7.8. check\_box

Данный метод позволяет добавить на страницу HTML элемент «`<input type="checkbox" />`».

Описание метода:

```
/***
 * Add checkbox tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - class - Set input class
 * - checked_value - Set checked value (default: 1)
 * - unchecked_value - Set unchecked value (default: 0)
 *
 * ===== Examples in Twig notations
 *
 * {{ check_box({'name': 'chk_test'}) }}
 * * => <input type="hidden" name="chk_test" value="0" />
 * <input type="checkbox" checked="checked" name="chk_test" checked="checked" value="1" />
 *
 * {{ check_box({'name': 'chk_test_2', 'checked_value': 'yes', 'unchecked_value': 'no'}) }}
 * * => <input type="hidden" name="chk_test_2" value="no" />
 * <input type="checkbox" checked="checked" name="chk_test_2" checked="checked" value="yes" />
 */
public function check_box(array $options = []): string {...}
```

## 15.7.9. color\_field

Данный метод позволяет добавить на страницу HTML элемент `<input type="color" />`.

Описание метода:

```
/**
 * Add color field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value (format: #rrggbba)
 *
 * ===== Examples in Twig notations
 *
 * color_field({'id': 'my-color-id', 'name': 'my-color', 'value': '#4c4c4c'})
 * * => <input type="color" id="my-color-id" name="my-color" value="#4c4c4c" />
 */
public function color_field(array $options = []): string {...}
```

## 15.7.10. date\_field

Данный метод позволяет добавить на страницу HTML элемент `<input type="date" />`.

Описание метода:

```
/**
 * Add date field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value (format: yyyy-MM-dd)
 * - min - Set input min value (format: yyyy-MM-dd)
 * - max - Set input max value (format: yyyy-MM-dd)
 *
 * ===== Examples in Twig notations
 *
 * date_field({'id': 'my-date-id', 'name': 'my-date', 'value': '2021-10-18'})
 * * => <input type="date" id="my-date-id" name="my-date" value="2021-10-18" />
 */
public function date_field(array $options = []): string {...}
```

## 15.7.11. datetime\_field

Данный метод позволяет добавить на страницу HTML элемент «`<input type="datetime-local" />`».

**ПРИМЕЧАНИЕ:** HTML элемент «`<input type="datetime" />`» считается устаревшим и запрещен для использования, так как может быть удален из поддержки в браузерах в любой момент!

Описание метода:

```
/**
 * Add date-time field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value (fromat: yyyy-MM-ddThh:mm)
 * - min - Set input min value (fromat: yyyy-MM-ddThh:mm)
 * - max - Set input max value (fromat: yyyy-MM-ddThh:mm)
 *
 * ===== Examples in Twig notations
 *
 * datetime_field({'id': 'my-datetime-id', 'name': 'my-datetime', 'value': '2021-10-18T15:10'})
 * * => <input type="datetime-local" id="my-datetime-id" name="my-datetime"
 * value="2021-10-18T15:10" />
 */
public function datetime_field(array $options = []): string {...}
```

## 15.7.12. datetime\_local\_field

Данный метод позволяет добавить на страницу HTML элемент «`<input type="datetime-local" />`».

**ПРИМЕЧАНИЕ:** это вспомогательный метод, вызывающий метод `datetime_field(...)`.

Описание метода:

```
/**
 * Add date-time-local field tag
 * @param array $options
 * @return string
 *
 * NOTE: this is alias for datetime_field function.
 *
 * ===== Examples in Twig notations
 *
 * datetime_local_field({'id': 'my-datetime-id', 'name': 'my-datetime',
 * 'value': '2021-10-18T15:10'})
 * * => <input type="datetime-local" id="my-datetime-id" name="my-datetime"
 * value="2021-10-18T15:10" />
 */
public function datetime_local_field(array $options = []): string {...}
```

## 15.7.13. email\_field

Данный метод позволяет добавить на страницу HTML элемент `<input type="email" />`.

Описание метода:

```
/**
 * Add email field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - pattern - Set input regular expression the input's contents must match in order
 * to be valid
 * - size - Set input number indicating how many characters wide the input field should be
 * - maxlength - Set input maximum number of characters the input should accept
 * - minlength - Set input minimum number of characters long the input can be and still be
 * considered valid
 * - required - Set input is required (default: false)
 *
 * ===== Examples in Twig notations
 *
 * email_field({'id': 'my-email-id', 'name': 'my-email', 'pattern': '.+@my\.com', 'size': 20,
 * 'required': true})
 * * => <input type="email" id="my-email-id" name="my-email" pattern=".+@my.com" size="20"
 * required />
 */
public function email_field(array $options = []): string {...}
```

## 15.7.14. month\_field

Данный метод позволяет добавить на страницу HTML элемент `<input type="month" />`.

Описание метода:

```
/**
 * Add month field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value (format: yyyy-MM)
 * - min - Set input min value (format: yyyy-MM)
 * - max - Set input max value (format: yyyy-MM)
 *
 * ===== Examples in Twig notations
 *
 * month_field({'id': 'my-month-id', 'name': 'my-month', 'value': '2021-10'})
 * * => <input type="month" id="my-month-id" name="my-month" value="2021-10" />
 */
public function month_field(array $options = []): string {...}
```

## 15.7.15. number\_field

Данный метод позволяет добавить на страницу HTML элемент `<input type="number" />`.

Описание метода:

```
/**
 * Add number field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value
 * - min - Set input min value
 * - max - Set input max value
 * - step - Set input stepping interval to use when using up and down arrows to adjust the
 value, as well as for validation
 *
 * ===== Examples in Twig notations
 *
 * number_field({'id': 'my-number-id', 'name': 'my-number', 'value': 2021, 'min': 2000,
 'max': 2500, 'step': 5})
 * * => <input type="number" id="my-number-id" name="my-number" value="2021" min="2000"
 max="2500" step="5" />
 */
public function number_field(array $options = []): string {...}
```

## 15.7.16. password\_field

Данный метод позволяет добавить на страницу HTML элемент `<input type="password" />`.

Описание метода:

```
/**
 * Add input password field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - class - Set input class
 * - size - Set input size
 * - value - Set input value
 * - onchange - Set input onchange handler
 *
 * ===== Examples in Twig notations
 *
 * {% set jquery_content %}
 * if ($('#my-password-id').val().length > 30) { alert('Your password needs to be shorter!'); }
 * {% endset %}
 * {{ password_field({'id': 'my-password-id', 'name': 'my-password', 'size': 10,
 'onchange': jquery_content}) }}
 *
 * Result:
 * <input type="password" id="my-password-id" name="my-password" size="10"
 onchange="if ($('#my-password-id').val().length > 30) {
 alert('Your password needs to be shorter!'); }" />
 */
public function password_field(array $options = []): string {...}
```

## 15.7.17. telephone\_field

Данный метод позволяет добавить на страницу HTML элемент `<input type="tel" />`.

Описание метода:

```
/**
 * Add telephone field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - pattern - Set input regular expression the input's contents must match in order
 * to be valid
 * - size - Set input number indicating how many characters wide the input field should be
 * - maxlength - Set input maximum number of characters the input should accept
 * - minlength - Set input minimum number of characters long the input can be
 * and still be considered valid
 * - required - Set input is required (default: false)
 *
 * ===== Examples in Twig notations
 *
 * telephone_field({'id': 'my-tel-id', 'name': 'my-tel', 'pattern': '[0-9]{3}-[0-9]{3}-[0-9]{4}',
 * 'size': 10, 'required': true})
 * * => <input type="tel" id="my-tel-id" name="my-tel" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}"
 * size="10" required />
 */
public function telephone_field(array $options = []): string {...}
```

## 15.7.18. phone\_field

Данный метод позволяет добавить на страницу HTML элемент `<input type="tel" />`.

ПРИМЕЧАНИЕ: это вспомогательный метод, вызывающий метод `<telephone_field(...)>`.

Описание метода:

```
/**
 * Add phone field tag
 * @param array $options
 * @return string
 *
 * NOTE: this is alias for telephone_field function.
 *
 * ===== Examples in Twig notations
 *
 * phone_field({'id': 'my-tel-id', 'name': 'my-tel', 'pattern': '[0-9]{3}-[0-9]{3}-[0-9]{4}',
 * 'size': 10, 'required': true})
 * * => <input type="tel" id="my-tel-id" name="my-tel" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}"
 * size="10" required />
 */
public function phone_field(array $options = []): string {...}
```

## 15.7.19. radio\_button

Данный метод позволяет добавить на страницу HTML элемент `<<input type="radio" />>`.

Описание метода:

```
/**
 * Add radio button tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value
 * - checked - Set input is checked (default: false)
 *
 * ===== Examples in Twig notations
 *
 * radio_button({'id': 'post_category_rails', 'name': 'post[category]', 'value': 'rails',
 * 'checked': true})
 * radio_button({'id': 'post_category_rails', 'name': 'post[category]', 'value': 'java'})
 * * => <input type="radio" id="post_category_rails" name="post[category]" value="rails"
 * checked="checked" />
 * <input type="radio" id="post_category_java" name="post[category]" value="java" />
 *
 */
public function radio_button(array $options = []): string {...}
```

## 15.7.20. range\_field

Данный метод позволяет добавить на страницу HTML элемент `<<input type="range" />>`.

Описание метода:

```
/**
 * Add range field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value
 * - min - Set input min value
 * - max - Set input max value
 * - step - Set input stepping interval to use when using up and down arrows
 * to adjust the value, as well as for validation
 *
 * ===== Examples in Twig notations
 *
 * range_field({'id': 'my-range-id', 'name': 'my-range', 'value': 5, 'min': 0, 'max': 10,
 * 'step': 1})
 * * => <input type="range" id="my-range-id" name="my-range" value="5" min="0" max="10"
 * step="1" />
 */
public function range_field(array $options = []): string {...}
```

## 15.7.21. search\_field

Данный метод позволяет добавить на страницу HTML элемент `<input type="search" />`.

Описание метода:

```
/**
 * Add search field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - pattern - Set input regular expression the input's contents
 * must match in order to be valid
 * - size - Set input number indicating how many characters wide the input field should be
 * - maxlength - Set input maximum number of characters the input should accept
 * - minlength - Set input minimum number of characters long the input
 * can be and still be considered valid
 *
 * ===== Examples in Twig notations
 *
 * search_field({'id': 'my-search-id', 'name': 'my-search', 'pattern': '[A-z]{2}[0-9]{4}',
 * 'size': 6})
 * * => <input type="search" id="my-search-id" name="my-search" pattern="[A-z]{2}[0-9]{4}"
 * size="6" />
 */
public function search_field(array $options = []): string {...}
```

## 15.7.22. text\_area

Данный метод позволяет добавить на страницу HTML элемент `<textarea>...</textarea>`.

Описание метода:

```
/**
 * Add text area tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - cols - The visible width of the text control, in average character widths.
 * If it is specified, it must be a positive integer.
 * If it is not specified, the default value is 20.
 * - rows - The number of visible text lines for the control.
 * - class - Set input class
 * - text_body - Set input text body (auto escape enabled)
 * - disabled - Set input disabled (default: false)
 *
 * ===== Examples in Twig notations
 *
 * text_area({'id': 'my-text-area-id', 'name': 'my-text-area', 'rows': 10,
 * 'text_body': 'Some text...'})
 * * => <textarea id="my-text-area-id" rows="10" name="my-text-area">
 * Some text...
 * </textarea>
 */
public function text_area(array $options = []): string {...}
```

## 15.7.23. text\_field

Данный метод позволяет добавить на страницу HTML элемент  
«`<input type="text" />`».

Описание метода:

```
/***
 * Add input text field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - class - Set input class
 * - size - Set input size
 * - value - Set input value
 * - maxlength - Set input max length
 * - minlength - Set input min length
 * - onchange - Set input onchange handler
 *
 * ===== Examples in Twig notations
 *
 * {%
 * set jquery_content %}
 * if ($('#my-text-id').val().length > 30) { alert('Your text needs to be shorter!'); }
 * {% endset %}
 * {{ text_field({'id': 'my-text-id', 'name': 'my-text', 'size': 10, 'value': '123-qwe-456-rty',
 * 'onchange': jquery_content}) }}
 *
 * Result:
 * <input type="text" id="my-text-id" name="my-text" size="10" value="123-qwe-456-rty"
 * onchange="if ($('#my-text-id').val().length > 30) {
 * alert('Your text needs to be shorter!'); }" />
 */
public function text_field(array $options = []): string {...}
```

## 15.7.24. time\_field

Данный метод позволяет добавить на страницу HTML элемент  
«`<input type="time" />`».

Описание метода:

```
/***
 * Add time field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value (format: hh:mm)
 * - min - Set input min value (format: hh:mm)
 * - max - Set input max value (format: hh:mm)
 * - step - Set input stepping interval to use when using up and down arrows
 * to adjust the value, as well as for validation
 * - required - Set input is required (default: false)
 *
 * ===== Examples in Twig notations
 *
 * time_field({'id': 'my-time-id', 'name': 'my-time', 'value': '17:16', 'min': '13:00',
 * 'max': '18:00', 'required': true})
 * => <input type="time" id="my-time-id" name="my-time" value="17:16" min="13:00"
 * max="18:00" required />
 */
public function time_field(array $options = []): string {...}
```

## 15.7.25. url\_field

Данный метод позволяет добавить на страницу HTML элемент  
«`<input type="url" />`».

Описание метода:

```
/**
 * Add input url field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value
 * - pattern - Set input regular expression the input's contents must match
 * in order to be valid
 * - placeholder - Set input exemplar value to display in the input field whenever it is empty
 * - size - Set input size
 * - maxlength - Set input max length
 * - minlength - Set input min length
 * - required - Set input is required (default: false)
 *
 * ===== Examples in Twig notations
 *
 * url_field({'id': 'my-url-id', 'name': 'my-url', 'placeholder': 'https://example.com',
 * 'pattern': 'https://.*', 'size': 30, 'required': true})
 * * => <input type="url" id="my-url-id" name="my-url" size="30"
 * placeholder="https://example.com" pattern="https://.*" required />
 */
public function url_field(array $options = []): string {...}
```

## 15.7.26. week\_field

Данный метод позволяет добавить на страницу HTML элемент  
«`<input type="week" />`».

Описание метода:

```
/**
 * Add input week field tag
 * @param array $options
 * @return string
 *
 * ===== Options
 *
 * - id - Set input id
 * - name - Set input name
 * - value - Set input value (format: yyyy-Wnn, where nn is a number with a leading zero)
 * - min - Set input min value (format: yyyy-Wnn, where nn is a number
 * with a leading zero)
 * - max - Set input max value (format: yyyy-Wnn, where nn is a number
 * with a leading zero)
 * - step - Set input stepping interval to use when using up and down arrows
 * to adjust the value, as well as for validation
 * - required - Set input is required (default: false)
 *
 * ===== Examples in Twig notations
 *
 * week_field({'id': 'my-week-id', 'name': 'my-week', 'value': '2021-W42'})
 * * => <input type="week" id="my-week-id" name="my-week" value="2021-W42" />
 */
public function week_field(array $options = []): string {...}
```

## 15.8. JavascriptTagHelper.php

Данный класс предоставляет базовый набор вспомогательных методов для добавления javascript блоков на web-страницу. Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/Helpers/JavascriptTagHelper.php».

### 15.8.1. javascript\_tag

Данный метод позволяет добавить на страницу javascript блок. Если в приложении включен режим CSP Protection и задан входной аргумент «'nonce': true», то в форму автоматически добавляется уникальный ключ блока данных.

Описание метода:

```
/**
 * Добавить тэг скрипта
 * @param string $data
 * @param array $options
 * @return mixed|string
 *
 * ===== Options
 *
 * - type - Set script type
 * - nonce - Enable/Disable nonce tag for script (only true/false) (default: false)
 *
 * NOTE: for use nonce - enable CSP Protection.
 *
 * ===== Examples
 *
 * javascript_tag("alert('Hi!');")
 * * => <script>
 * //<![CDATA[
 * alert('Hi!');
 * //]]>
 * </script>
 *
 * javascript_tag("alert('Hi!');", {"type": "application/javascript"})
 * * => <script type="application/javascript">
 * //<![CDATA[
 * alert('Hi!');
 * //]]>
 * </script>
 *
 * javascript_tag("alert('Hi!');", {"type": "application/javascript", "nonce": true})
 * * => <script nonce="e8cf820e1e236731eea842ef95b592f7" type="application/javascript">
 * //<![CDATA[
 * alert('Hi!');
 * //]]>
 * </script>
 *
 * ===== Examples in Twig notations
 *
 * {% set js_str %}
 * console.log("Hi! I am test console!");
 * console.log("Hi! I am test console-2!");
 * console.log("Hi! I am test console-3!");
 * {% endset %}
 * {{ javascript_tag(js_str, {"nonce": true}) }}
 */
public function javascript_tag(string $data, array $options = array()) {...}
```

## 15.9. StylesheetTagHelper.php

Данный класс предоставляет базовый набор вспомогательных методов для добавления блоков стилей на web-страницу. Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/Helpers/StylesheetTagHelper.php».

### 15.9.1. stylesheet\_tag

Данный метод позволяет добавить на страницу блок стиля. Если в приложении включен режим CSP Protection и задан входной аргумент «'nonce': true», то в форму автоматически добавляется уникальный ключ блока данных.

Описание метода:

```
/**
 * Добавить тэг скрипта
 * @param string $data
 * @param array $options
 * @return mixed|string
 *
 * ===== Options
 *
 * - nonce - Enable/Disable nonce tag for script (only true/false) (default: false)
 *
 * NOTE: for use nonce - enable CSP Protection.
 *
 * ===== Examples
 *
 * stylesheet_tag("html,body { background-color: red !important; }")
 * * => <script type="text/css">
 * //<![CDATA[
 * html,body { background-color: red !important; }
 * //]]>
 * </script>
 *
 * stylesheet_tag("html,body { background-color: red !important; }", {"nonce": true})
 * * => <script nonce="e8cf820e1e236731eea842ef95b592f7" type="text/css">
 * //<![CDATA[
 * html,body { background-color: red !important; }
 * //]]>
 * </script>
 *
 * ===== Examples in Twig notations
 *
 * {% set css_str %}
 * html,
 * body {
 * background-color: red !important;
 * }
 * {% endset %}
 * {{ stylesheet_tag(css_str, {"nonce": true}) }}
 */
public function stylesheet_tag(string $data, array $options = array()) {...}
```

## 15.10. ProtectionTagHelper.php

Данный класс предоставляет базовый набор вспомогательных методов для добавления различных HTML элементов для повышения безопасности вашего приложения при рендеринге страницы в браузере или при выполнении различных запросов. Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/Helpers/ProtectionTagHelper.php».

### 15.10.1. csrf\_meta\_tags

Данный метод позволяет добавить на страницу HTML мета-теги обеспечивающие работу в режиме CSRF Protection.

Описание метода:

```
/**
 * Добавить HTML-мета-тэги CSRF Protection
 * @return string
 * @throws \Exception
 *
 * ===== Examples in Twig notations
 *
 * csrf_meta_tags()
 * * => <meta name="csrf-param" content="authenticity_token" />
 * <meta name="csrf-token" content="y/ISRjCq7t/FjB0ZWuACd9Tb1QiR5rjRu/n/SvL1c
 * PU7DG0xGxWLmyRmB1AGr+X2wlTU3mpR0NWPNrUxyTwv1A==" />
 */
public function csrf_meta_tags() {...}
```

### 15.10.2. csp\_meta\_tag

Данный метод позволяет добавить на страницу HTML мета-тег обеспечивающий работу в режиме CSP Protection.

Описание метода:

```
/**
 * Добавить HTML-мета-тэг CSP Protection
 * @return string
 *
 * ===== Examples in Twig notations
 *
 * csp_meta_tag()
 * * => <meta name="csp-nonce" content="760230a0ce05ba6aa546549d362eb55f" />
 */
public function csp_meta_tag() {...}
```

## 16. Дополнительные инструменты управления проектом

Исполняемый файл `bin/fly_cube_php`, расположенный в каталоге проекта предоставляет дополнительный перечень команд для управления текущим проектом. Перечень дополнительных доступных команд приводится ниже:

```
MyProject/bin> ./fly_cube_php -h
Usage: ./fly_cube_php [options]
Options include:
--help Show this message [-h, -?]
--version Print the version [-v]
--output=[true/false] Show output (optional)
--env=[VALUE] Set current environment (production/development;
 default: development)

...
--secret Create application secret key
--secret-length=[VALUE] Set application secret key length (value > 0)

--assets-precompile Build assets cache
--clear-cache Clear all application cache
--clear-logs Clear all application logs
--clear-php-sessions Clear all php sessions
```

Более подробно эти команды рассматриваются ниже в отдельных главах.

### 16.1. --secret - генерация секретных ключей приложения

Данная команда позволяет генерировать случайные ключи для приложения. По-умолчанию размер ключа составляет 128 символов (64 байта). Пример генерации ключа приложения:

```
MyProject/bin> ./fly_cube_php --secret
== FlyCubePHP: Application secret ==
0ffc1f121f4c5e5a7753e1d91c9a6dbb9a67478d48badcd93c3b95381b656add339e32af245420a2018625209202723c
830cb8339edd0a24170c4050c97522d
== FlyCubePHP =====
```

Так же можно воспользоваться блоком команд для генерации ключа приложения и автоматического создания файла «`[MyProject]/config/secret.key`»:

```
MyProject/bin> ./fly_cube_php --secret | grep -E --only-matching -e "[0-9a-f]{128}" > ../config/secret.key
```

Содержимое файла секретного ключа должно выглядеть следующим образом:

```
MyProject/bin> cat ../config/secret.key
ff46f41e287f005a659c08493f158315f25ff98f7ff1732599612630109edf879ba49c00cc157af0698f7fb3fbad43d14
0c7c8c888c74aadd02548d14eab47e0
```

Для генерации ключей, на основании которых будет производиться шифрование данных cookie, рекомендуется использовать команду с указанием размера ключа в 32 символа (16 байт) «--secret-length=16»:

```
MyProject/bin> ./fly_cube_php --secret --secret-length=16
== FlyCubePHP: Application secret ==
878ade99eaaf1ca779fa9308282d43b1
== FlyCubePHP =====
```

## 16.2. --check-dirs - проверка каталогов проекта

Данная команда позволяет выполнить проверку каталогов проекта и создать все недостающие каталоги, или изменить права каталогов на необходимые. Пример проверки каталогов проекта:

```
MyProject/bin> ./fly_cube_php --check-dirs
== FlyCubePHP: Checking project catalogs ==
Project name: MyProject
Project path: /home/[USER]/FlyCubePHProjects/MyProject
- Checking project catalogs: OK
== FlyCubePHP =====
```

## 16.3. --assets-precompile - сборка ресурсов

Данная команда позволяет выполнить сборку ресурсов приложения и сформировать корректный кэш. Этот механизм позволяет в значительной степени увеличить производительность во время работы, так как максимально снижает время на поиск ядром запрашиваемых ресурсов. Пример сборки ресурсов приложения:

```
MyProject/bin> ./fly_cube_php --assets-precompile
== FlyCubePHP: Assets precompile ==
[Render] TestController->root: OK
[Render] TestController->test_2: OK
== FlyCubePHP =====
```

## **16.4. --clear-cache - очистка кэша приложения**

Данная команда позволяет удалить все данные приложения, сохраненные в кэш.  
Пример очистки кэша приложения:

```
MyProject/bin> ./fly_cube_php --clear-cache
==== FlyCubePHP: Clear application cache ====
[Delete] tmp/cache/FlyCubePHP/asset_pipeline/4a83a2c819d6a05350305a609ba25c699b3895ddc0d0d9556251055320cfb099
[Delete] tmp/cache/FlyCubePHP/asset_pipeline
[Delete] tmp/cache/FlyCubePHP/css_builder/4a83a2c819d6a05350305a609ba25c699b3895ddc0d0d9556251055320cfb099
[Delete] tmp/cache/FlyCubePHP/css_builder/pre_build/test.css
[Delete] tmp/cache/FlyCubePHP/css_builder/pre_build
[Delete] tmp/cache/FlyCubePHP/css_builder
[Delete] tmp/cache/FlyCubePHP/js_builder/4a83a2c819d6a05350305a609ba25c699b3895ddc0d0d9556251055320cfb099
[Delete] tmp/cache/FlyCubePHP/js_builder/pre_build/test.js
[Delete] tmp/cache/FlyCubePHP/js_builder/pre_build
[Delete] tmp/cache/FlyCubePHP/js_builder
[Delete] tmp/cache/FlyCubePHP
[Delete] tmp/cache/Twig/84/84edd80a1179696679d2ff93bd1c717463556a5816b63b41d046cf8fb679009.php
[Delete] tmp/cache/Twig/84
[Delete] tmp/cache/Twig/e9/e9fa3a6e54a033f991b3c66a80bcdbe9923a0f8511b5ce9fe1523df57a3113.php
[Delete] tmp/cache/Twig/e9
[Delete] tmp/cache/Twig/f4/f4c19303f8311c3e059910782be2ba33980f13a927166576cc7b2f984442ba7f.php
[Delete] tmp/cache/Twig/f4
[Delete] tmp/cache/Twig
[Delete] tmp/cache
==== FlyCubePHP =====
```

ПРИМЕЧАНИЕ: в случае ошибок доступа рекомендуется использовать данную команду совместно с командой «`sudo`»:

```
MyProject/bin> sudo ./fly_cube_php --clear-cache
```

## **16.5. --clear-logs - очистка журналов функционирования приложения**

Данная команда позволяет удалить все журналы функционирования приложения из каталога «`log`». Пример очистки log-файлов приложения:

```
MyProject/bin> ./fly_cube_php --clear-logs
==== FlyCubePHP: Clear application logs ====
[Delete] log/development.log
[Delete] log/production.log
==== FlyCubePHP =====
```

ПРИМЕЧАНИЕ: в случае ошибок доступа рекомендуется использовать данную команду совместно с командой «`sudo`»:

```
MyProject/bin> sudo ./fly_cube_php --clear-logs
```

## **16.6. --clear-php-sessions - очистка PHP сессий**

Данная команда позволяет удалить все файлы PHP сессий. Пример очистки файлов PHP сессий:

```
MyProject/bin> ./fly_cube_php --clear-php-sessions
==== FlyCubePHP: Clear php sessions ====
[Delete] /var/lib/php7/sess_8fpaqv2kv15k95p94qaqhmac9
[Delete] /var/lib/php7/sess_plrhbspaiohuarknjbtfk9gulb
[Delete] /var/lib/php7/sess_vorvrjcv18di8a1boa04b2h7qf
==== FlyCubePHP =====
```

**ПРИМЕЧАНИЕ:** в случае ошибок доступа рекомендуется использовать данную команду совместно с командой «`sudo`»:

```
MyProject/bin> sudo ./fly_cube_php --clear-php-sessions
```

**ВНИМАНИЕ:** данная команда удаляет все файлы PHP сессий, поэтому используйте её с осторожностью!

## 17. Менеджер компонентов (плагинов) и его архитектура

Менеджер компонентов представляет из себя набор классов, которые позволяют описывать плагины, их зависимости, и загружать эти плагины в приложение при его запуске.

Ниже приводится перечень основных файлов менеджера компонентов:

1. `BaseComponent.php` - базовый класс плагина (содержит набор основных методов, которые должен реализовывать плагин);
2. `ComponentDependency.php` - класс, описывающий зависимости плагина (необходим менеджеру, для формирования дерева зависимостей);
3. `ComponentsManager.php` - основной класс менеджера компонентов (данный класс производит проверку, загрузку и работу с плагинами приложения);
4. `DependencyTreeElement.php` - класс описания узла зависимости плагина.

Менеджер компонентов производит следующие действия:

1. загрузка игнор-листа плагинов «`config/ignore-list.conf`» - данный файл может содержать перечень плагинов, которые система должна игнорировать при загрузке (имя плагина, начинающееся с символа # считается комментарием и пропускается);
2. проверка каталога «`plugins/`» и загрузка плагинов в порядке нахождения их в каталоге, если отсутствуют в игнор-листе;
3. инициализация плагинов - на данном этапе производится поиск дерева зависимостей плагина и попытка их загрузить (так же выполняется проверка на цикличность).

Основные методы менеджера компонентов, доступные в приложении для поиска необходимых зависимостей плагинов:

- `\FlyCubePHP\ComponentsCore\ComponentsManager::instance()` - получить объект менеджера компонентов;
- `...instance()>version()` - получить версию менеджера компонентов;
- `...instance()>state()` - получить состояние менеджера компонентов;
- `...instance()>plugins()` - получить список всех плагинов;
- `...instance()>pluginsControllers()` - получить массив информации о контроллерах плагинов;
- `...instance()>applicationAllControllers()` - получить массив информации о контроллерах всего приложения;
- `...instance()>pluginsDir()` - получить название каталога с плагинами;
- `...instance()>ignoreList()` - получить список игнорируемых плагинов;
- `...instance()>appendPlugin(BaseComponent &$plugin)` - добавить плагин в менеджер;
- `...instance()>pluginByClassName(string $class_name)` - поиск плагина по имени класса;
- `...instance()>pluginByControllerName(string $controller_name)` - поиск плагина по имени контроллера.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/ComponentsCore/ComponentsManager.php».

## 17.1. Архитектура компонентов (плагинов)

Условно, плагины можно разделить на две части:

1. Поставщики объектов компонента;
2. Потребители объектов компонента.

Поставщики объектов представляют собой плагины (модули), которые могут предоставить другим плагинам объекты своих классов для выполнения задач, на которые нацелен плагин «поставщик». Этими объектами могут быть PHP-классы. Все возможные манипуляции с предоставляемыми объектами описываются в виде публичных функций класса.

Плагин может предоставлять как один объект, так и несколько. Все созданные объекты, которые плагин предоставляет для других компонентов, хранятся непосредственно в плагине и могут быть запрошены соответствующими методами. При выгрузке плагина, все созданные им объекты удаляются.

Потребители объектов представляют собой плагины (модули), которые используют в своей работе объекты других плагинов. На момент инициализации плагина «потребителя», из менеджера компонентов запрашиваются плагины, объекты которых требуются для работы данного плагина. Если необходимые плагины были загружены менеджером компонентов, то они будут возвращены плагину «потребителю». Если необходимые объекты отсутствуют в менеджере, то инициализация плагина, который их запросил, завершается с ошибкой.

Необходимые объекты всегда можно запросить из менеджера компонентов на любом этапе выполнения приложения.

Так же следует отметить, что плагин может быть одновременно «поставщиком» и «потребителем».

На рисунке 24 показана схема менеджера компонентов и его взаимодействие с плагинами:

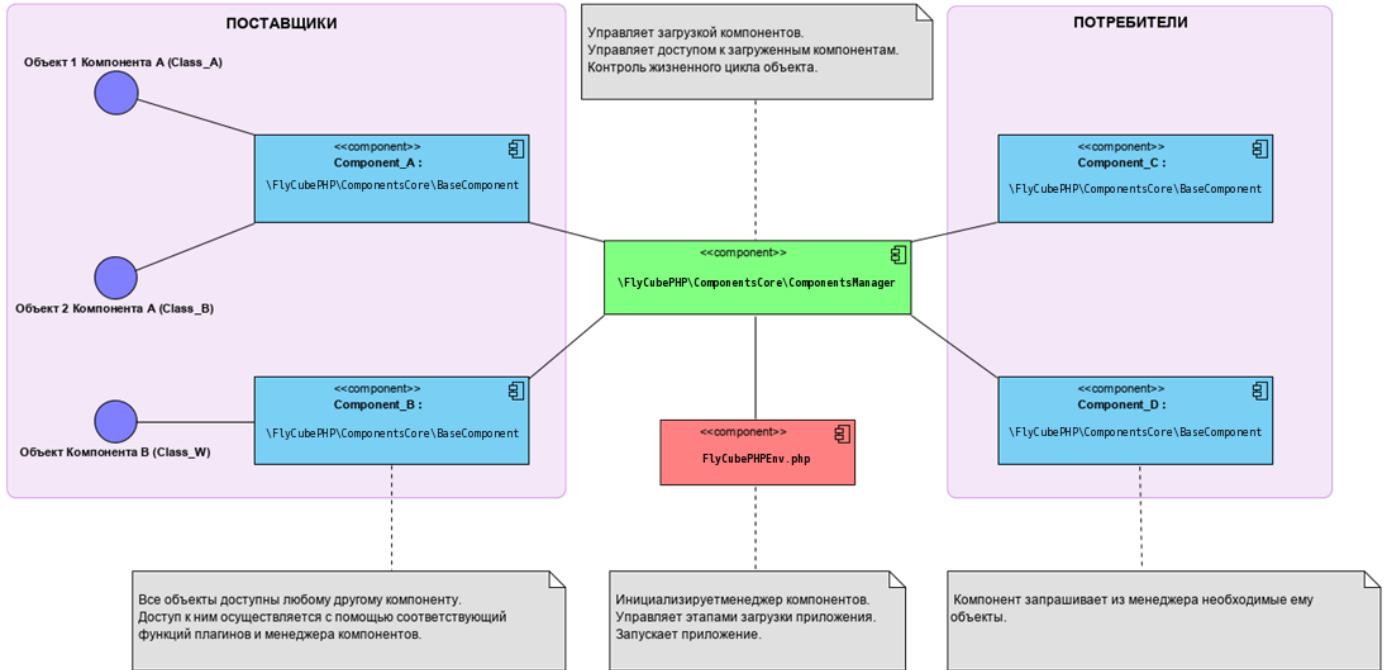


Рис. 24. Схема менеджера компонентов и его взаимодействие с плагинами

Все объекты плагинов должны быть унаследованы от базового класса плагина системы «\FlyCubePHP\ComponentsCore\BaseComponent», описание которого можно найти в файле «FlyCubePHP/src/ComponentsCore/BaseComponent.php».

Базовый класс плагина предоставляет следующий набор методов, обязательных к реализации:

- \FlyCubePHP\ComponentsCore\BaseComponen->init() - метод инициализации объекта плагина. В данном методе разработчик выполняет все необходимые действия по созданию внутренних объектов класса; поиск зависимых плагинов, указанных в списке зависимостей плагина. Поиск объектов неявных зависимостей на данном этапе производить не стоит, так как они могут быть еще не загружены менеджером. В данном методе нужно вернуть состояние инициализации объекта (true / false).

Перечень базовых методов, доступных объекту плагина:

- ->name() - название объекта плагина;
- ->version() - версия объекта плагина;
- ->compatVersion() - версия совместимости объекта плагина;
- ->description() - описание объекта плагина;
- ->dependence() - список обязательных зависимостей объекта плагина (без которых объект не может функционировать);
- ->optionalDependence() - список необязательных зависимостей объекта плагина (без которых функционирование возможно);

- `->state()` - состояние объекта плагина (задается менеджером компонентов на этапах загрузки и инициализации);
- `->hasWarnings()` - есть ли предупреждения;
- `->warnings()` - список предупреждений объекта плагина;
- `->lastWarning()` - последнее предупреждение объекта плагина;
- `->setLastWarning(string $warning)` - добавить предупреждение объекта плагина;
- `->hasErrors()` - есть ли ошибки;
- `->errors()` - список ошибок объекта плагина;
- `->lastError()` - последняя ошибка объекта плагина;
- `->setLastError(string $error)` - добавить ошибку объекта плагина;
- `->directory()` - имя каталога плагина (задается менеджером на этапе инициализации);
- `->hasControllers()` - задан ли список контроллеров;
- `->controllers()` - список контроллеров объекта плагина (задается менеджером на этапе инициализации);
- `->addDependence(string $name, string $version)` - добавить обязательную зависимость объекта плагина;
- `->addOptionalDependence(string $name, string $version)` - добавить необязательную зависимость объекта плагина.

Остальные методы запрещены к использованию, так как разработаны исключительно для взаимодействия менеджера компонентов с объектом плагина. Попытки переопределения данных методов могут привести к некорректной работе как объекта плагина, так и всей системы в целом.

**ПРИМЕЧАНИЕ:** методы `«->addDependence(...)`» и `«->addOptionalDependence(...)`» должны использоваться в конструкторе класса компонента, так как при вызове метода `«init()`» ядро системы плагинов уже не проверят дерево зависимостей.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле `«FlyCubePHP/src/ComponentsCore/BaseComponent.php»`.

## 17.2. Зависимости плагинов

Зависимостями плагина можно назвать объекты других плагинов, необходимых для функционирования первого. В системе разделяются два типа зависимостей:

- Обязательные - зависимости, без которых объект плагина не может функционировать; при отсутствии обязательных зависимостей менеджер компонентов прекращает инициализацию объекта плагина и переходит к следующему.
- Необязательные - зависимости, без которых объект плагина может функционировать (возможно, в ограниченном функционале); при отсутствии необязательных

зависимостей менеджер компонентов выдает предупреждение и продолжает инициализацию объекта плагина.

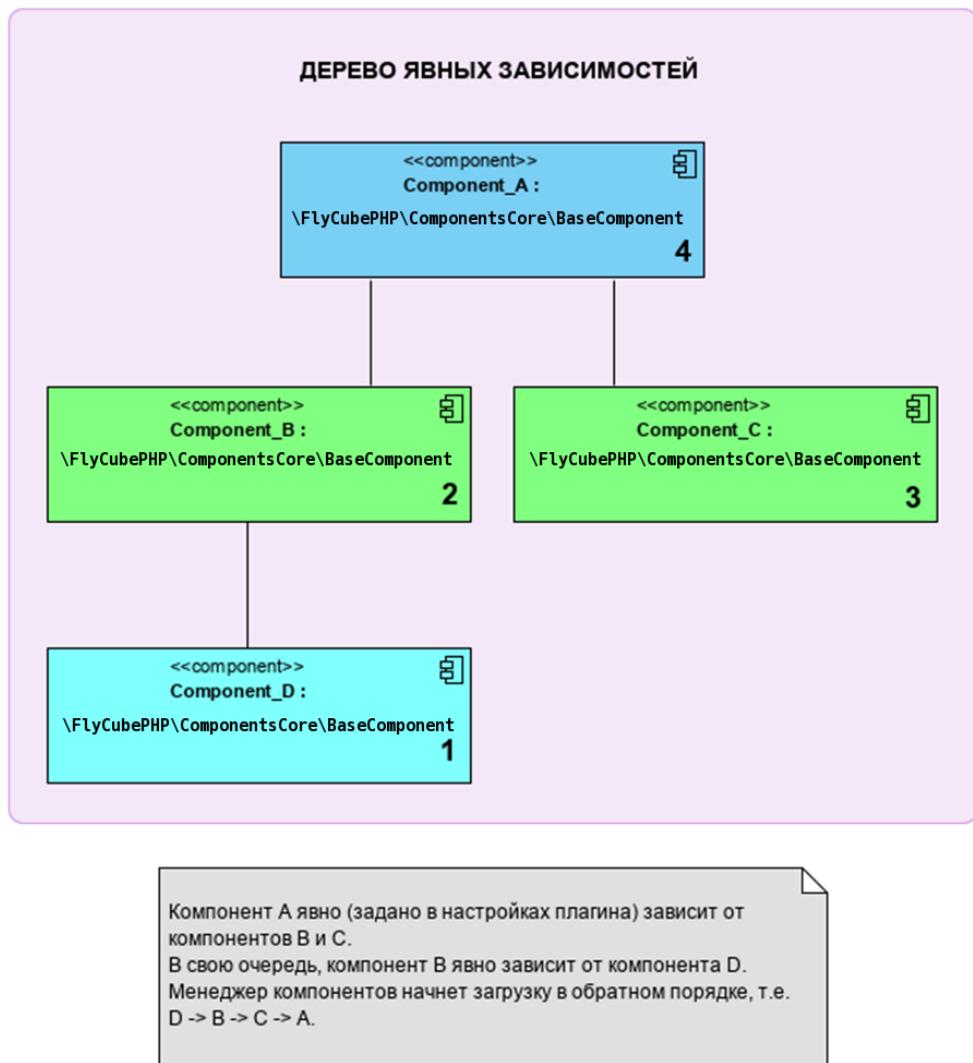


Рис. 25. Дерево явных зависимостей

### 17.3. Структура каталогов плагина

Плагины в системе можно условно разделить на две группы:

- Консольные плагины
- Графические плагины.

Консольные плагины не предоставляют графического интерфейса системы. В основном они являются поставщиками объектов данных или функционала, необходимого для работы других плагинов. Так же консольные плагины могут предоставлять внешний API функционал, регистрируя в системе API контроллеры.

Графические плагины так же могут предоставлять внешний API функционал, но данный подход не рекомендуется к реализации, так как графический плагин должен отвечать только за визуализацию данных.

## 17.3.1. Создание консольного плагина и описание структуры его каталогов

Для создания нового консольного плагина требуется выполнить следующую команду генератора «./fly\_cube\_php --new --plugin --name=ExamplePlugin»:

```
MyProject/bin> ./fly_cube_php --new --plugin --name=ExamplePlugin

==== FlyCubePHP: Create new plugin ===
[Created] plugins/ExamplePlugin
[Created] plugins/ExamplePlugin/app/controllers
[Created] plugins/ExamplePlugin/app/models
[Created] plugins/ExamplePlugin/config
[Created] plugins/ExamplePlugin/db/migrate
[Created] plugins/ExamplePlugin/doc/api
[Created] plugins/ExamplePlugin/lib/assets/images
[Created] plugins/ExamplePlugin/lib/assets/javascripts
[Created] plugins/ExamplePlugin/lib/assets/stylesheets
[Created] plugins/ExamplePlugin/init.php
[Created] plugins/ExamplePlugin/ExamplePlugin.php
[Created] plugins/ExamplePlugin/config/routes.php
==== FlyCubePHP =====
```

Консольный плагин не предоставляет графический интерфейс, поэтому в структуре его каталог отсутствуют каталог для представлений. В остальном он схож со структурой графического плагина.

Структура каталогов представлена ниже:

- app [Каталог приложения плагина]
  - controllers [Контроллеры плагина. Только API контроллеры!]
  - models [Классы моделей]
- db [Файлы баз данных]
  - migrate [Файлы миграций]
- doc [Файлы генерации API-DOC и справки]
  - api [Файлы генерации API-DOC]
- config [Каталог конфигураций плагина]
  - routes.php [Файл URL путей доступа плагина]
- lib [Каталог библиотек, разработанных в рамках плагина]
  - assets [Ассеты]
    - images [Изображения: png, jpg, jpeg, svg, gif]
    - javascripts [Скрипты: \*.js, \*.js.php]
    - stylesheets [Стили: \*.css, \*.scss]
- [PLUGIN\_NAME].php [Основной файл объекта плагина]
- init.php [Основной файл для загрузки плагина в менеджер компонентов]

Подробное описание каталогов и файлов плагина:

- app/controllers - каталог контроллеров плагина; в консольном плагине данный каталог может содержать только API контроллеры;
- app/models - каталог классов моделей данных;
- db/migrate - каталог миграций БД;
- doc/api - каталог файлов генерации API-Doc;
- config/routes.php - файл url путей доступа плагина; при генерации контроллера автоматически заполняется путями доступа, которые разработчик должен

- отредактировать, учитывая решаемые методом контроллера задачи; автоматически загружается системой при инициализации плагина;
- `lib` - каталог для библиотек, разработанных в рамках плагина; обычно тут располагается реализация объектов, предоставляемых плагином для функционирования других плагинов;
  - `lib/assets` - каталог скриптовых библиотек, разработанных в рамках плагина; обычно тут располагается реализация объектов, предоставляемых плагином для функционирования других плагинов;
  - `[PLUGIN_NAME].php` - основной файл объекта плагина; должен быть унаследован от базового класса плагина системы «\FlyCubePHP\ComponentsCore\BaseComponent»; при генерации кода `[PLUGIN_NAME]` заменяется на имя генерируемого плагина;
  - `init.php` - основной файл для загрузки плагина в менеджер компонентов; данный файл загружается менеджером компонентов при поиске плагинов; если данный файл не будет найден, менеджер компонентов проигнорирует каталог.

### 17.3.2. Создание графического плагина и описание структуры его каталогов

Для создания нового графического плагина требуется выполнить следующую команду генератора `./fly_cube_php --new --plugin-gui --name=ExamplePluginGui`:

```
MyProject/bin> ./fly_cube_php --new --plugin-gui --name=ExamplePluginGui

==== FlyCubePHP: Create new gui plugin ====
[Created] plugins/ExamplePluginGui
[Created] plugins/ExamplePluginGui/app/assets/images
[Created] plugins/ExamplePluginGui/app/assets/javascripts
[Created] plugins/ExamplePluginGui/app/assets/stylesheets
[Created] plugins/ExamplePluginGui/app/controllers
[Created] plugins/ExamplePluginGui/app/helpers
[Created] plugins/ExamplePluginGui/app/models
[Created] plugins/ExamplePluginGui/app/views
[Created] plugins/ExamplePluginGui/config
[Created] plugins/ExamplePluginGui/db/migrate
[Created] plugins/ExamplePluginGui/doc/help/images
[Created] plugins/ExamplePluginGui/doc/api
[Created] plugins/ExamplePluginGui/lib
[Created] plugins/ExamplePluginGui/help/images
[Created] plugins/ExamplePluginGui/init.php
[Created] plugins/ExamplePluginGui/ExamplePluginGui.php
[Created] plugins/ExamplePluginGui/config/routes.php
==== FlyCubePHP =====
```

Структура каталогов графического плагина почти полностью повторяет структуру каталогов приложения FlyCubePHP, за исключением небольших изменений, которые накладывает ядро приложения.

Структура каталогов представлена ниже:

- `app` [Каталог приложения плагина]
  - `assets` [Ассыты]
    - `images` [Изображения: `png, jpg, jpeg, svg, gif`]

- `javascripts` [*Скрипты: \*.js, \*.js.php*]
  - `stylesheets` [*Стили: \*.css, \*.scss*]
- `controllers` [*Контроллеры плагина*]
- `helpers` [*Вспомогательные классы представлений (views)*]
- `models` [*Модели данных*]
- `views` [*Представления: \*.html, \*.html.twig*]
- `db` [*Файлы баз данных*]
  - `migrate` [*Файлы миграций*]
- `doc` [*Файлы генерации API-DOC и справки*]
  - `api` [*Файлы генерации API-DOC*]
  - `help` [*Файлы генерации справки*]
    - `images` [*Изображения для генерации справки*]
- `config` [*Каталог конфигураций плагина*]
  - `routes.php` [*Файл URL путей доступа плагина*]
- `lib` [*Каталог библиотек, разработанных в рамках плагина*]
- `help` [*Каталог справки*]
  - `images` [*Каталог изображений для справки*]
- `[PLUGIN_NAME].php` [*Основной файл объекта плагина*]
- `init.php` [*Основной файл для загрузки плагина в менеджер компонентов*]

Подробное описание каталогов и файлов плагина:

- `app/assets/javascripts` - каталог скриптов плагина; должен содержать скрипты для всех контроллеров, которые предоставляет плагин (при генерации контроллера автоматически создается скрипт для данного контроллера, который будет загружен на страницу при доступе к контроллеру плагина из графического интерфейса); так же системой поддерживаются скрипты с расширением `*.js.php` для вставки в js файл php кода;
- `app/assets/images` - каталог изображений плагина; рекомендовано использовать изображения с расширениями `png, jpg, jpeg, gif, svg`; все изображения автоматически загружаются в систему при инициализации плагина и доступны для поиска в Asset Pipeline по их имени;
- `app/assets/stylesheets` - каталог CSS (SCSS) стилей плагина; должен содержать скрипты для всех контроллеров, которые предоставляет плагин (при генерации контроллера автоматически создается скрипт для данного контроллера, который будет загружен на страницу при доступе к контроллеру плагина из графического интерфейса); так же поддерживаются Sass стили (`scss`); автоматически загружаются в систему при инициализации плагина;
- `app/controllers` - каталог контроллеров плагина; в консольном плагине данный каталог может содержать только API контроллеры;
- `app/helpers` - каталог вспомогательных классов представлений контроллеров; автоматически загружается при отрисовке страницы контроллера;
- `app/models` - каталог классов моделей данных;
- `app/views` - каталог представлений для методов контроллеров;
- `db/migrate` - каталог с миграциями для базы данных;
- `doc/help` - каталог файлов генерации справки;
- `doc/help/images` - каталог изображений для генерации справки;
- `doc/api` - каталог файлов генерации API-Doc;
- `config/routes.php` - файл url путей доступа плагина; при генерации контроллера автоматически заполняется путями доступа, которые разработчик должен

отредактировать, учитывая решаемые методом контроллера задачи; автоматически загружается системой при инициализации плагина;

- `lib` - каталог для библиотек, разработанных в рамках плагина; обычно тут располагается реализация объектов, необходимых для отображения графического интерфейса плагина;
- `help` - основной каталог файлов справки; может содержать подкаталоги с именами локалей (en, ru и т.д.), что укажет системе загружать файлы справки из соответствующего имени текущей локали системы каталога; файлы справки могут иметь расширения \*.hlp и \*.help (подробное описание по формированию файла справки будет рассмотрено дальше);
- `help/images` - каталог изображений для файлов справки;
- `[PLUGIN_NAME].php` - основной файл объекта плагина; должен быть унаследован от базового класса плагина системы «\FlyCubePHP\ComponentsCore\BaseComponent»; при генерации кода `[PLUGIN_NAME]` заменяется на имя генерируемого плагина;
- `init.php` - основной файл для загрузки плагина в менеджер компонентов; данный файл загружается менеджером компонентов при поиске плагинов; если данный файл не будет найден, менеджер компонентов проигнорирует каталог.

**ПРИМЕЧАНИЕ:** на текущий момент ядро FlyCubePHP не поддерживает систему справки и каталог `«[PLUGIN_NAME]/doc/help/*»` игнорируется при загрузке.

**ПРИМЕЧАНИЕ:** JavaScript библиотеки, разработанные в рамках плагина и которые могут быть полезны другим разработчикам, рекомендуется располагать в основном каталоге приложения `«lib/assets/javascripts/»`. В противном случае расположение данных библиотек рекомендовано оставить в каталоге плагина `«[PLUGIN_NAME]/app/assets/javascripts/»`.

## 18. API Doc - создание описаний для API

Начиная с версии FlyCubePHP 1.4.0 поддерживается создание описаний для API на основе json. Ядро FlyCubePHP API Doc поддерживает следующий функционал:

- автоматический поиск, загрузку и разбор файлов описания API в каталогах:
  - «doc/api/»
  - «plugins/\*/doc/api/»
- преобразование разобранных описаний в формат markdown с автоматическим кешированием результата.

Имя файла с описанием API должно полностью соответствовать имени класса контроллера, для которого содержит описание, и иметь расширение json. Методы API, указанные в файле api-doc должны полностью соответствовать именам методов, реализованных в классе контроллера. Если данные условия будут не соблюдены, содержимое файла будет проигнорировано при разборе.

Пример файла api-doc:

```
{
 "api-block-name": "[DEFAULT Controller class name]",
 "api-block-description": "[DEFAULT Empty]",

 "[Controller action name)": {
 "description": "",
 "version": "[API Version aka 0.0.1]",
 "deprecated": false,
 "output-formats": ["json"],

 "param-[Param Name)": {
 "name": "[Param Name]",
 "description": "",
 "type": "Int",
 "optional": false,
 "empty": false,
 "min": 0,
 "max": 100,
 "available-values": ["aa", "bb"],

 "param-[Param Name N)": {
 ...
 },
 },

 "param-group-[Param Group Name)": {
 "name": "[Param Group Name]",

 "param-[Param Name N1)": {
 ...
 },
 "param-[Param Name N2)": {
 ...
 },
 "param-[Param Name Nn)": {
 ...
 },
 },

 "headers": {
 "[HTTP Header name)": "HTTP Header Description"
 },
 },
}
```

```

"return": {
 "code": 200,
 "type": "Hash",
 "description": "",

 "param-[Param Name N1)": {
 ...
 },
 "param-[Param Name Nn)": {
 ...
 }
},

"error-[Error Name)": {
 "name": "[Error Name]",
 "code": 500,
 "description": "",

 "param-[Param Name N1)": {
 ...
 },
 "param-[Param Name Nn)": {
 ...
 }
},

"example-[Example Name)": {
 "name": "[Example Name]",
 "description": "",
 "request": "This is one-line example",
 "response": "This is one-line example"
},

"example-[Example With Multilines)": {
 "name": "[Example With Multilines]",
 "description": "",
 "request": [
 "This is multiline",
 "example for request",
 "...etc"
],
 "response": [
 "This is multiline",
 "example for response",
 "...etc"
]
}
}
}

```

Более подробное описание разделов файла api-doc будет рассмотрено в последующих подразделах.

## 18.1. Загрузка API Doc и доступ к их объектам

За загрузку и доступ к разобранным объектам API Doc отвечает класс «FlyCubePHP\Core\ApiDoc\ApiDoc». По умолчанию поддержка API Doc в проектах FlyCubePHP отключена. Для его включения требуется добавить в конфигурационный файл вашего проекта «[MyProject]/config/application\_env.conf» следующую строку с настройками: «FLY\_CUBE\_PHP\_ENABLE\_API\_DOC: true». С этого момента ядро FlyCubePHP будет автоматически производить поиск и загрузку файлов API Doc.

Класс ApiDoc предоставляет следующие методы:

- `->isEnabled()` - метод проверки, включена ли система API Doc;
- `->apiDocFiles()` - получить список всех загруженных API Doc файлов;
- `->apiDocDirs(bool $fullPath = false)` - получить список всех добавленных в ядро каталогов API Doc; если «\$fullPath = true» - вернуть полные пути к каталогам;
- `->appendApiDocDir(string $dir)` - добавить и просканировать каталог API Doc;
- `->apiDoc(string $name)` - получить объект с разобранным описанием API Doc, где «name» - название файла API Doc; если разбор успешен, то возвращается объект класса «FlyCubePHP\Core\ApiDoc\ApiDocObject», в противном случае возвращается «null»;

**ПРИМЕЧАНИЕ:** данный метод не производит кэширование; разбор запрашиваемого файла описания API производится каждый раз при вызове метода.

- `->apiDocMarkdown(string $name, string $action = "")` - получить описание API Doc в формате markdown, где:
  - «name» - название файла API Doc;
  - «action» - название метода контроллера, описанного в API Doc.

**ПРИМЕЧАНИЕ:** если «action» не указан, то формируется markdown файл по всему файлу описания API Doc, в противном случае формируется markdown файл только с описанием требуемого метода.

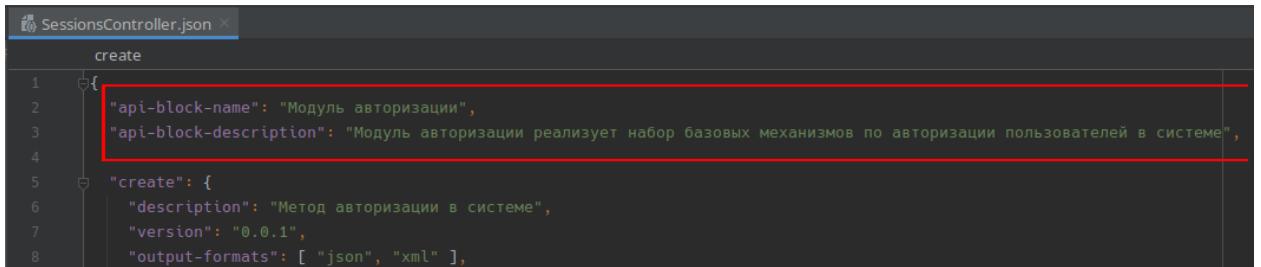
**ПРИМЕЧАНИЕ:** в «production mode» данный метод автоматически кэширует сгенерированные markdown файлы; все последующие вызовы выполняют обращение уже к сформированному кэшу.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core\ApiDoc\ApiDoc.php».

## 18.2. Разделы API Doc и их PHP классы

Файл API Doc должен содержать два основных ключа:

- [string] «api-block-name» - содержит название блока API методов; на момент автогенерации содержит имя класса контроллера;
- [string] «api-block-description» - содержит описание блока API методов; может отсутствовать.



```
1 "create": {
2 "api-block-name": "Модуль авторизации",
3 "api-block-description": "Модуль авторизации реализует набор базовых механизмов по авторизации пользователей в системе",
4
5 "create": {
6 "description": "Метод авторизации в системе",
7 "version": "0.0.1",
8 "output-formats": ["json", "xml"],
9 }
10 }
11 }
```

Рис. 26. Пример раздела описания API в json файле

### Модуль авторизации

Модуль авторизации реализует набор базовых механизмов по авторизации пользователей в системе

Resource	Description
POST /login	Метод авторизации в системе
GET /login_pass_through	Метод запроса на вход при сквозной аутентификации
POST /login_refresh	Метод обновления состояния активного пользователя
DELETE /logout	Метод выхода из системы

Рис. 27. Пример возможного отображения раздела описания API

PHP классом, содержащим всю информацию API Doc, является «FlyCubePHP\Core\ApiDoc\ApiDocObject». Данный класс предоставляет следующий набор методов:

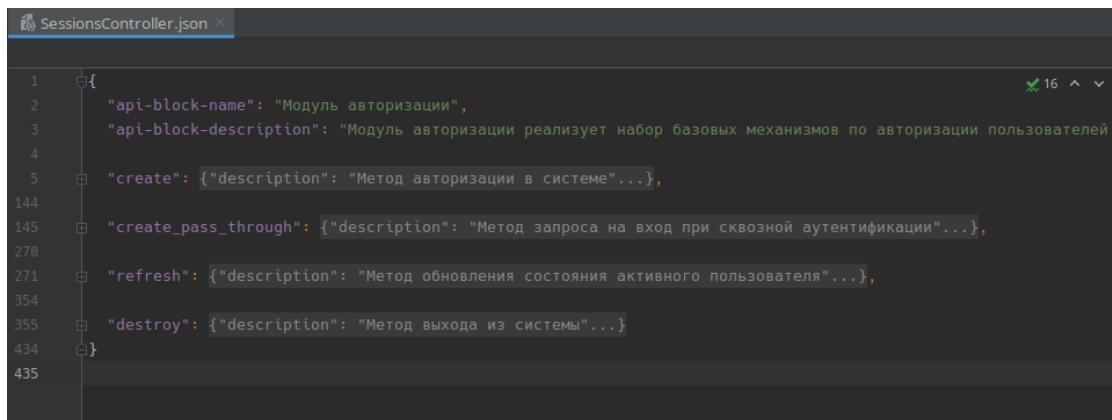
- ->name() - название блока API по имени файла описания;
- ->blockName() - название блока API методов;
- ->blockDescription() - описание блока API методов;
- ->actions() - список методов блока API (массив объектов «FlyCubePHP\Core\ApiDoc\ApiDocAction»);
- ->sourceFilePath() - путь до файла описания блока API методов контроллера;
- ->buildMarkdown() - сгенерировать markdown файл с описанием API.

**ПРИМЕЧАНИЕ:** данный метод не использует кэширование. Для быстрого доступа к сгенерированному markdown файлу с использованием механизмов кэширования пользуйтесь методом «ApiDoc::instance()->apiDocMarkdown(string \$name, string \$action = "")».

Так же файл API Doc должен содержать перечень методов с их описанием. Имена методов должны соответствовать именам функций, реализующих описываемый

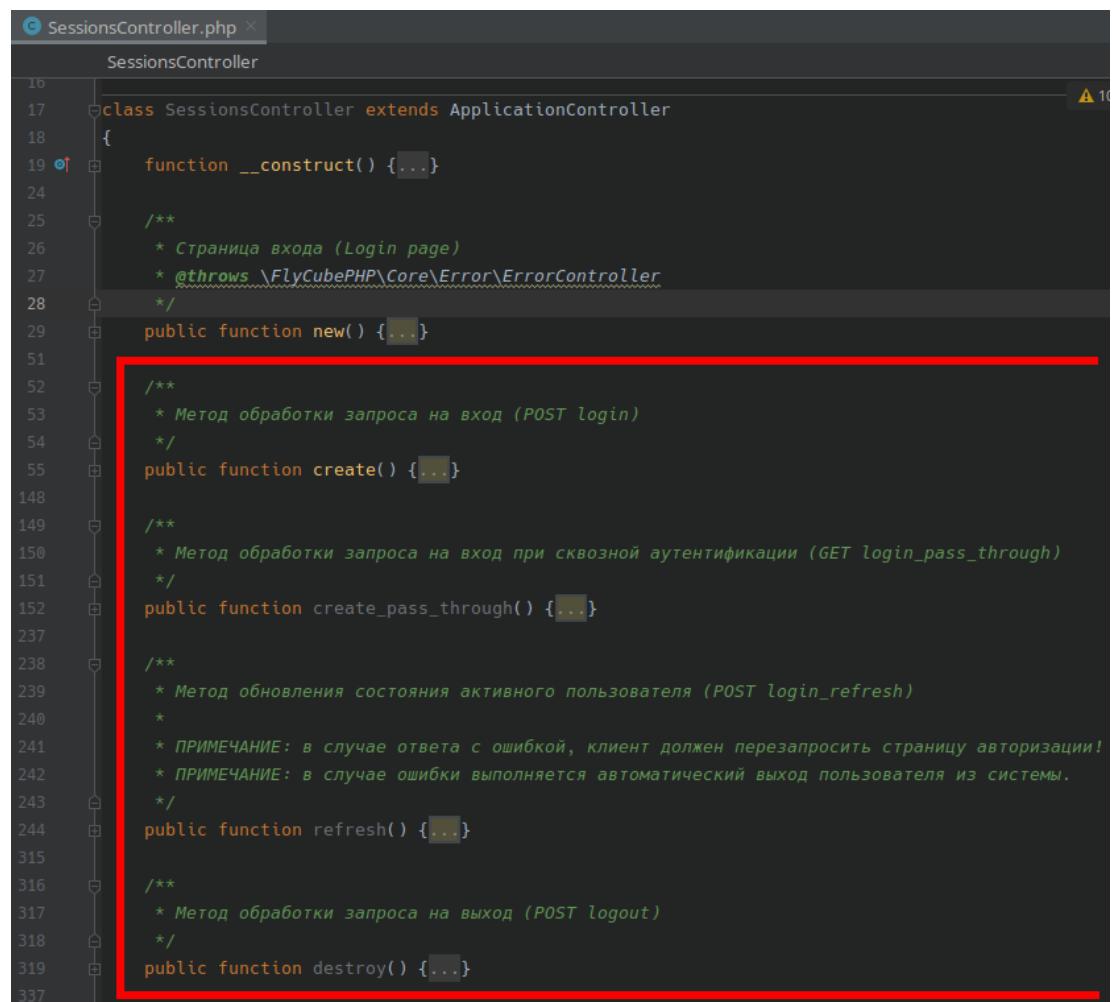
функционал и указанных в файле маршрутов. Если имя метода не будет совпадать с именем функции класса контроллера, описание будет проигнорировано.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «`FlyCubePHP/src/Core/ApiDoc/ApiDocObject.php`».



```
SessionsController.json
1 {
2 "api-block-name": "Модуль авторизации",
3 "api-block-description": "Модуль авторизации реализует набор базовых механизмов по авторизации пользователей"
4
5 "create": {"description": "Метод авторизации в системе"...},
144
145 "create_pass_through": {"description": "Метод запроса на вход при сквозной аутентификации"...},
270
271 "refresh": {"description": "Метод обновления состояния активного пользователя"...},
354
355 "destroy": {"description": "Метод выхода из системы..."}
434
435 }
```

Рис. 28-1. Пример перечня методов раздела описания API и их соответствие методам класса контроллера



```
SessionsController.php
10
11 class SessionsController extends ApplicationController
12 {
13 function __construct() {...}
14
15 /**
16 * Страница входа (Login page)
17 * @throws \FlyCubePHP\Core\Exception\ErrorController
18 */
19 public function new() {...}
20
21 /**
22 * Метод обработки запроса на вход (POST login)
23 */
24 public function create() {...}
25
26 /**
27 * Метод обработки запроса на вход при сквозной аутентификации (GET login_pass_through)
28 */
29 public function create_pass_through() {...}
30
31 /**
32 * Метод обновления состояния активного пользователя (POST login_refresh)
33 *
34 * ПРИМЕЧАНИЕ: в случае ответа с ошибкой, клиент должен перезапросить страницу авторизации!
35 * ПРИМЕЧАНИЕ: в случае ошибки выполняется автоматический выход пользователя из системы.
36 */
37 public function refresh() {...}
38
39 /**
40 * Метод обработки запроса на выход (POST logout)
41 */
42 public function destroy() {...}
43 }
```

Рис. 28-2. Пример перечня методов раздела описания API и их соответствие методам класса контроллера

## 18.3. Описание метода API и его PHP класс

Описание метода API состоит из нескольких блоков, каждый из которых содержит свою информацию. В зависимости от блока информации, он может отсутствовать вовсе, или встречаться более одного раза. В примечаниях к каждому блоку это будет указано.

Перечень ключей, позволяющих задать базовое описание метода API:

- `[string] «description»` - описание метода (может отсутствовать или быть пустым);
- `[string] «version»` - версия метода в формате `«maj.min.patch»` (может отсутствовать);
- `[bool] «deprecated»` - позволяет указать, что метод является устаревшим (может отсутствовать; `default: false`);
- `[array] «output-formats»` - содержит массив возможных форматов возвращаемых данных.

Перечень возможных блоков, включаемых в описание метода API:

- `«param-[NAME]»` - блок описания входного параметра и его свойств, где `[NAME]` - название параметра (может отсутствовать или встречаться более одного раза);
- `«param-group-[PARAM GROUP NAME]»` - блок описания группы входных параметров, где `[PARAM GROUP NAME]` - название группы параметров (может отсутствовать или встречаться более одного раза);

**ПРИМЕЧАНИЕ:** группы входных параметров введены для описания взаимоисключающих входных параметров в рамках одного API метода контроллера (пример описания будет рассмотрен ниже).

- `«headers»` - блок описания требуемых входных HTTP заголовков (может отсутствовать);
- `«return»` - блок описания ответа сервера (возвращаемые данные);
- `«error-[NAME]»` - блок описания ошибки, где `[NAME]` - название ошибки (может отсутствовать или встречаться более одного раза);
- `«example-[NAME]»` - блок описания примера, где `[NAME]` - название примера (может отсутствовать или встречаться более одного раза).

**ПРИМЕЧАНИЕ:** тип HTTP метода и URL, требуемые для доступа к описываемому методу API, ядро FlyCubePHP API Doc получает автоматически, используя загруженные в систему маршруты и контроллеры.

PHP классом, содержащим всю информацию метода API Doc, является `«FlyCubePHP\Core\ApiDoc\ApiDocAction»`. Данный класс предоставляет следующий набор методов:

- `->httpMethod( )` - тип HTTP метода (`get/post/put/patch/delete`, смотри `RouteType::...`);

- `->url()` - URL доступа к методу без аргументов, если они были заданы в файле маршрутов;
- `->urlFull()` - URL доступа к методу с аргументами, если они были заданы в файле маршрутов;
- `->name()` - название метода API;
- `->description()` - описание метода API;
- `->version()` - версия метода API;
- `->outputFormats()` - перечень форматов возвращаемых данных (массив строк);
- `->isDeprecated()` - проверка, является ли метод API устаревшим;
- `->hasParameters()` - проверка, задан ли массив входных параметров;
- `->parameters()` - массив входных параметров (массив объектов `«FlyCubePHP\Core\ApiDoc\ApiDocParameter»`);
- `->hasParametersGroups()` - проверка, задан ли массив групп параметров;
- `->parametersGroups()` - массив групп параметров (массив объектов `«FlyCubePHP\Core\ApiDoc\ApiDocParametersGroup»`);
- `->hasHttpHeaders()` - проверка, заданы ли требуемые HTTP заголовки;
- `->httpHeaders()` - массив требуемых HTTP заголовков (массив ключ-значение, где ключ - HTTP заголовок, а значение - описание этого HTTP заголовка);
- `->returnData()` - объект описания ответа сервера (возвращаемые данные) (объект класса `«FlyCubePHP\Core\ApiDoc\ApiDocReturn»`);
- `->hasErrors()` - проверка, заданы ли возможные ошибки;
- `->errors()` - массив возможных ошибок (массив объектов `«FlyCubePHP\Core\ApiDoc\ApiDocError»`);
- `->hasExamples()` - проверка, заданы ли примеры;
- `->examples()` - массив примеров (массив объектов `«FlyCubePHP\Core\ApiDoc\ApiDocExample»`);
- `->buildMarkdown()` - сгенерировать markdown файл с описанием метода API.

**ПРИМЕЧАНИЕ:** данный метод не использует кэширование. Для быстрого доступа к сгенерированному markdown файлу с использованием механизмов кэширования пользуйтесь методом `«ApiDoc::instance()->apiDocMarkdown(string $name, string $action = "")»`.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле `«FlyCubePHP/src/Core/ApiDoc/ApiDocAction.php»`.

```

SessionsController.json

1 {
2 "api-block-name": "Модуль авторизации",
3 "api-block-description": "Модуль авторизации реализует набор базовых механизмов по авторизации пользователей в системе",
4
5 "create": {
6 "description": "Метод авторизации в системе",
7 "version": "0.0.1",
8 "output-formats": ["json", "xml"],
9
10 "param-output": { "name": "output" },
11
12 "param-session": { "name": "session" },
13
14 "headers": { "Accept": "Указывает, в каком формате вернуть данные клиенту (application/json; application/xml)" },
15
16 "return": { "code": 200 },
17
18 "error-500-1": { "code": 500 },
19 "error-500-2": { "code": 500 },
20 "error-500-3": { "code": 500 },
21 "error-500-4": { "code": 500 },
22 "error-500-5": { "code": 500 },
23 "error-500-6": { "code": 500 },
24
25 },
26
27 "example-Curl-JSON-Output": { "name": "Curl - JSON Output" },
28 "example-Curl-XML-Output": { "name": "Curl - XML Output" }
29
30 }

```

Рис. 29. Пример описания метода API в json файле

**Version:**0.0.1

*Метод авторизации в системе*

---

**Output formats:**

- json
- xml

---

**Required HTTP headers:**

HTTP Header name	Description
Accept	Указывает, в каком формате вернуть данные клиенту (application/json; application/xml)

---

**Input params:**

Param name	Description
<b>output</b> <small>?</small> <small>optional</small>	Формат возвращаемых данных <b>Validation:</b> <ul style="list-style-type: none"> <li>Type: String</li> <li>Available values: json, xml</li> </ul>
<b>session</b> <small>!</small> <small>required</small>	Массив с данными для авторизации <b>Validation:</b> <ul style="list-style-type: none"> <li>Type: Hash</li> </ul>
<b>login</b> <small>!</small> <small>required</small>	Логин пользователя <b>Validation:</b> <ul style="list-style-type: none"> <li>Type: String</li> </ul>
<b>password</b> <small>!</small> <small>required</small>	Пароль пользователя <b>Validation:</b> <ul style="list-style-type: none"> <li>Type: String</li> </ul>

Рис. 30. Пример возможного отображения описания метода API

## 18.4. Описание параметра API и его PHP класс

Описание параметра API состоит из нескольких элементов, каждый из которых содержит свою информацию и позволяет описать все необходимые характеристики. Так же параметр может включать в себя неограниченное число блоков с дочерними параметрами.

Перечень ключей, позволяющих задать описание параметра API:

- [string] «name» - название параметра;
- [string] «description» - описание метода (может отсутствовать или быть пустым);
- [string] «type» - тип данных параметра;
- [bool] «optional» - позволяет указать, что параметр является необязательным (может отсутствовать; default: false);
- [bool] «empty» - позволяет указать, что значение параметра может быть пустым (может отсутствовать; default: false);
- [int] «min» - позволяет задать минимальное значение параметра (может отсутствовать; рекомендовано только для числовых типов);
- [int] «max» - позволяет задать максимальное значение параметра (может отсутствовать; рекомендовано только для числовых типов);
- [array] «available-values» - содержит массив возможных значений параметра (может отсутствовать);
- [Parameter Block] «param-[Param Name N]» - блок описания вложенного параметра и его свойств, где [Param Name N] - название параметра (может отсутствовать или встречаться более одного раза).

**ПРИМЕЧАНИЕ:** группы входных параметров не могут быть включены в описание параметра и игнорируются при разборе.

PHP классом, содержащим всю информацию параметра API Doc, является «FlyCubePHP\Core\ApiDoc\ApiDocParameter». Данный класс предоставляет следующий набор методов:

- ->name() - название параметра;
- ->description() - описание параметра;
- ->dataType() - тип данных параметра;
- ->isOptional() - проверка, является ли параметр необязательным (опциональным);
- ->canBeEmpty() - проверка, может ли параметр иметь пустое значение;
- ->min() - минимальное значение параметра (актуально для числовых типов);
- ->max() - максимальное значение параметра (актуально для числовых типов);
- ->hasAvailableValues() - проверка, задан ли массив возможных значений;
- ->availableValues() - массив возможных значений;
- ->hasParameters() - проверка, задан ли массив вложенных параметров;
- ->parameters() - массив вложенных параметров.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/ApiDoc/ApiDocParameter.php».

```
9
10 "param-output": {
11 "name": "output",
12 "description": "Формат возвращаемых данных",
13 "type": "String",
14 "empty": false,
15 "optional": true,
16 "available-values": ["json", "xml"]
17 },
18
19 "param-session": {
20 "name": "session",
21 "description": "Массив с данными для авторизации",
22 "type": "Hash",
23 "empty": false,
24
25 "param-login": {
26 "name": "login",
27 "description": "Логин пользователя",
28 "type": "String",
29 "empty": false
30 },
31
32 "param-password": {
33 "name": "password",
34 "description": "Пароль пользователя",
35 "type": "String",
36 "empty": false
37 }
38 },
39 }
```

Рис. 31. Пример описания параметров метода API в json файле

Input params:	
Param name	Description
output ?	Формат возвращаемых данных <b>Validation:</b> <ul style="list-style-type: none"><li>Type: String</li><li>Available values: json, xml</li></ul>
session !	Массив с данными для авторизации <b>Validation:</b> <ul style="list-style-type: none"><li>Type: Hash</li></ul>
└ login !	Логин пользователя <b>Validation:</b> <ul style="list-style-type: none"><li>Type: String</li></ul>
└ password !	Пароль пользователя <b>Validation:</b> <ul style="list-style-type: none"><li>Type: String</li></ul>

Рис. 32. Пример возможного отображения описания параметров метода API

## 18.5. Описание группы параметров API и её PHP класс

Группы входных параметров введены для описания взаимоисключающих входных параметров в рамках одного API метода контроллера.

Перечень ключей, позволяющих задать описание группы параметров API:

- `[string] «name»` - название группы параметров;
- `[Parameter Block] «param-[Param Name N]»` - блок описания вложенного параметра и его свойств, где `[Param Name N]` - название параметра (может отсутствовать или встречаться более одного раза).

**ПРИМЕЧАНИЕ:** для описания взаимоисключающих параметров необходимо создать две или более группы входных параметров.

PHP классом, содержащим всю информацию группы параметров API Doc, является `«FlyCubePHP\Core\ApiDoc\ApiDocParametersGroup»`. Данный класс предоставляет следующий набор методов:

- `->name()` - название группы параметров;
- `->description()` - описание группы параметров;
- `->isEmpty()` - проверка, является ли группа параметров пустой (это синоним метода `«hasParameters»`);
- `->hasParameters()` - проверка, задан ли массив вложенных параметров;
- `->parameters()` - массив вложенных параметров.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле `«FlyCubePHP/src/Core/ApiDoc/ApiDocParameterGroup.php»`.

```

40
41 "param-group-1": {
42 "name": "1",
43
44 "param-guid": {
45 "name": "guid",
46 "description": "Ид объекта данных",
47 "type": "String",
48 "empty": false
49 }
50 },
51 "param-group-2": {
52 "name": "2",
53
54 "param-key": {
55 "name": "key",
56 "description": "Ключ объекта данных",
57 "type": "String",
58 "empty": false
59 }
60 },
61

```

Рис. 33. Пример описания групп параметров метода API в json файле

#### Input params:

Param name	Description
<b>output</b> ! required	Output data format <b>Validation:</b> <ul style="list-style-type: none"> <li>Type: String</li> <li>Available values: json, xml</li> </ul>
<b>guid</b> ! required	Ид объекта данных <b>Validation:</b> <ul style="list-style-type: none"> <li>Type: String</li> </ul>
<b>key</b> ! required	Ключ объекта данных <b>Validation:</b> <ul style="list-style-type: none"> <li>Type: String</li> </ul>

Рис. 34. Пример возможного отображения описания групп параметров метода API

## 18.6. Описание требуемых входных HTTP заголовков API

Блок описания требуемых входных HTTP заголовков (может отсутствовать) задается в описании метода API ключом «headers», и является массивом ключ-значение, где:

- ключ - требуемый HTTP заголовок;
- значение - описание требуемого HTTP заголовка.

```
368
369 "headers": [
370 "Accept": "Указывает, в каком формате вернуть данные клиенту (application/json; application/xml)",
371 "X-CSRF-Token": "HTTP заголовок с сессионным ключом сервера"
372 },
373]
```

Рис. 35. Пример описания требуемых входных HTTP заголовков API в json файле

### Required HTTP headers:

HTTP Header name	Description
Accept	Указывает, в каком формате вернуть данные клиенту (application/json; application/xml)
X-CSRF-Token	HTTP заголовок с сессионным ключом сервера

Рис. 36. Пример возможного отображения описания требуемых входных HTTP заголовков API

## 18.7. Описание ответа сервера API и его PHP класс

Блок описания ответа сервера API состоит из нескольких элементов, каждый из которых содержит свою информацию и позволяет описать все необходимые характеристики.

Перечень ключей, позволяющих задать описание ответа сервера API:

- [int] «code» - код ответа HTTP;
- [string] «description» - описание ответа (может отсутствовать или быть пустым);
- [string] «type» - тип данных ответа;
- [Parameter Block] «param-[Param Name N]» - блок описания вложенного параметра и его свойств, где [Param Name N] - название параметра (может отсутствовать или встречаться более одного раза).

ПРИМЕЧАНИЕ: группы входных параметров не могут быть включены в описание ответа сервера и игнорируются при разборе.

PHP классом, содержащим всю информацию ответа сервера API Doc, является «FlyCubePHP\Core\ApiDoc\ApiDocReturn». Данный класс предоставляет следующий набор методов:

- ->httpCode() - код ответа HTTP;
- ->description() - описание ответа;
- ->dataType() - тип возвращаемых данных;
- ->hasParameters() - проверка, задан ли массив вложенных параметров;
- ->parameters() - массив вложенных параметров.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/ApiDoc/ApiDocReturn.php».

```
43
44 "return": {
45 "code": 200,
46 "type": "Hash",
47 "description": "Результат успешной авторизации",
48
49 "param-message": {
50 "name": "message",
51 "description": "Сообщение системы",
52 "type": "String"
53 },
54
55 "param-state_code": {
56 "name": "state_code",
57 "description": "Код состояния",
58 "type": "String"
59 },
60
61 "param-authenticity_token": {
62 "name": "authenticity_token",
63 "description": "Токен авторизации сервера",
64 "type": "String"
65 },
66
67 "param-refresh_auth_info_interval": {
68 "name": "refresh_auth_info_interval",
69 "description": "Интервал обновления информации о пользователе (сек)",
70 "type": "Number"
71 },
72
73 "param-refresh_auth_info_support": {
74 "name": "refresh_auth_info_support",
75 "description": "Требуется ли обновление информации о пользователе",
76 "type": "Bool"
77 }
78 },
79 }
```

Рис. 37. Пример описания ответа сервера API в json файле

### Response:

Результат успешной авторизации	
Param name	Description
message	Сообщение системы <b>Validation:</b> <ul style="list-style-type: none"><li>Type: String</li></ul>
state_code	Код состояния <b>Validation:</b> <ul style="list-style-type: none"><li>Type: String</li></ul>
authenticity_token	Токен авторизации сервера <b>Validation:</b> <ul style="list-style-type: none"><li>Type: String</li></ul>
refresh_auth_info_interval	Интервал обновления информации о пользователе (сек) <b>Validation:</b> <ul style="list-style-type: none"><li>Type: Number</li></ul>
refresh_auth_info_support	Требуется ли обновление информации о пользователе <b>Validation:</b> <ul style="list-style-type: none"><li>Type: Bool</li></ul>

Рис. 38. Пример возможного отображения описания ответа сервера API

## 18.8. Описание ошибки сервера API и её PHP класс

Блок описания ошибки сервера API состоит из нескольких элементов, каждый из которых содержит свою информацию и позволяет описать все необходимые характеристики.

Перечень ключей, позволяющих задать описание ошибки сервера API:

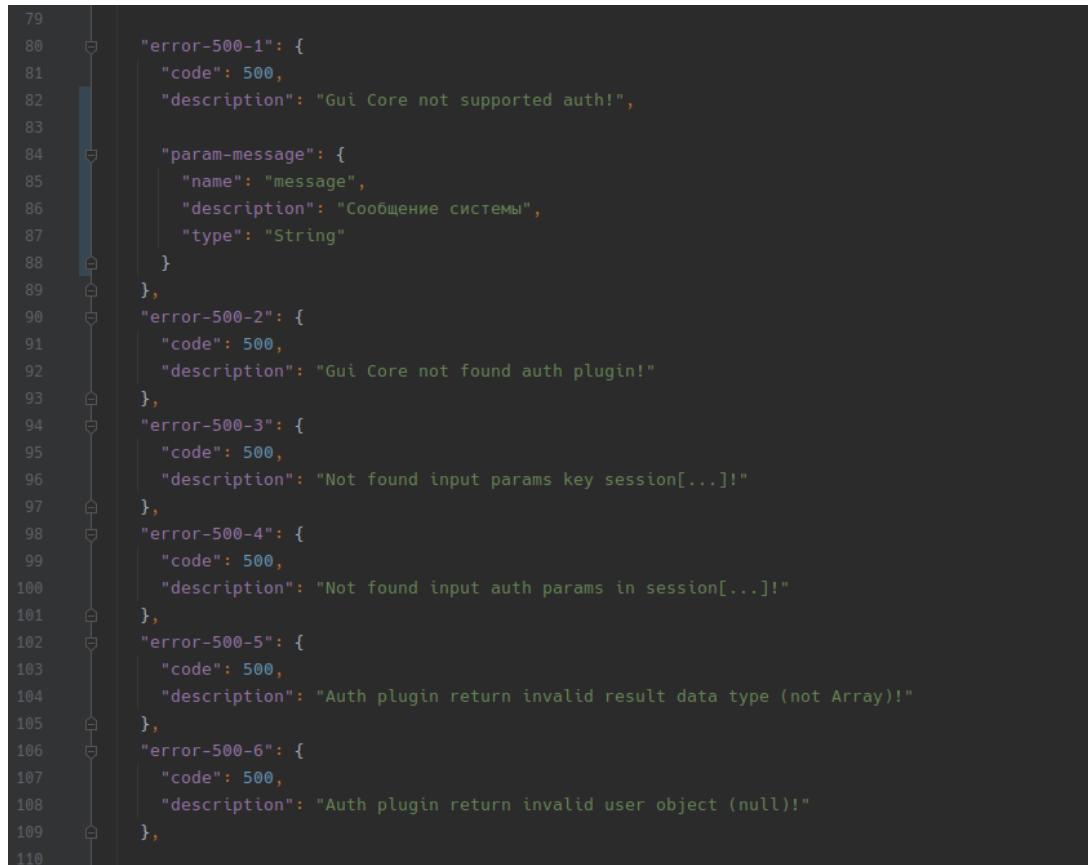
- [string] «name» - название ошибки;
- [int] «code» - код ответа HTTP;
- [string] «description» - описание ошибки (может отсутствовать или быть пустым);
- [Parameter Block] «param-[Param Name N]» - блок описания вложенного параметра и его свойств, где [Param Name N] - название параметра (может отсутствовать или встречаться более одного раза).

ПРИМЕЧАНИЕ: группы входных параметров не могут быть включены в описание ошибки сервера и игнорируются при разборе.

PHP классом, содержащим всю информацию ошибки сервера API Doc, является «FlyCubePHP\Core\ApiDoc\ApiDocError». Данный класс предоставляет следующий набор методов:

- `->httpCode()` - код ответа HTTP;
- `->name()` - название ошибки;
- `->description()` - описание ошибки;
- `->hasParameters()` - проверка, задан ли массив вложенных параметров;
- `->parameters()` - массив вложенных параметров.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/ApiDoc/ApiDocError.php».



```

79
80 "error-500-1": {
81 "code": 500,
82 "description": "Gui Core not supported auth!",
83
84 "param-message": {
85 "name": "message",
86 "description": "Сообщение системы",
87 "type": "String"
88 }
89 },
90 "error-500-2": {
91 "code": 500,
92 "description": "Gui Core not found auth plugin!"
93 },
94 "error-500-3": {
95 "code": 500,
96 "description": "Not found input params key session[...]!"
97 },
98 "error-500-4": {
99 "code": 500,
100 "description": "Not found input auth params in session[...]!"
101 },
102 "error-500-5": {
103 "code": 500,
104 "description": "Auth plugin return invalid result data type (not Array)!"
105 },
106 "error-500-6": {
107 "code": 500,
108 "description": "Auth plugin return invalid user object (null)!"
109 },
110

```

*Рис. 39. Пример описания ошибок сервера API в json файле*

#### Errors:

HTTP Status Code	Description	Response params
500	Gui Core not supported auth!	<i>String message</i> - Сообщение системы
500	Gui Core not found auth plugin!	
500	Not found input params key session[...]!	
500	Not found input auth params in session[...]!	
500	Auth plugin return invalid result data type (not Array)!	
500	Auth plugin return invalid user object (null)!	

*Рис. 40. Пример возможного отображения описания ошибок сервера API*

## 18.9. Описание примера использования API и его PHP класс

Блок описания примера использования API состоит из нескольких элементов, каждый из которых содержит свою информацию и позволяет описать все необходимые характеристики.

Перечень ключей, позволяющих задать описание примера использования API:

- [string] «name» - название примера;
- [string] «description» - описание примера (может отсутствовать или быть пустым);
- [string|array] «request» - описание запроса клиента (может отсутствовать или быть пустым);
- [string|array] «response» - описание ответа сервера (может отсутствовать или быть пустым).

**ПРИМЕЧАНИЕ:** для создания многострочных значений полей «request» и «response» используйте массив строк. На этапе разбора json файла все значения массива будут «собраны» в одну строку с добавлением символа переноса строки «\n».

PHP классом, содержащим всю информацию примера использования API Doc, является «FlyCubePHP\Core\ApiDoc\ApiDocExample». Данный класс предоставляет следующий набор методов:

- ->name() - название ошибки;
- ->description() - описание ошибки;
- ->hasRequest() - проверка, задан ли пример запроса клиента;
- ->request() - пример запроса клиента;
- ->hasResponse() - проверка, задан ли пример ответа сервера;
- ->response() - пример ответа сервера.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core\ApiDoc\ApiDocExample.php».

```

104
105 "example-Curl-JSON-Output": {
106 "name": "Curl - JSON Output",
107 "description": "Использование curl для авторизации (формат ответа: JSON)",
108 "request": [
109 "curl -c cookie-jar.txt -d \\\"output=json&session[login]=Admin&session[password]=12345678\\\" -X POST http://127.0.0.1:8080/login",
110 "",
111 "где:",
112 "- cookie-jar.txt - файл с cookie,"
113],
114 "response": [
115 "{",
116 " \"message\": \"success\",
117 " \"state_code\": 200,
118 " \"authenticity_token\": \"Ta6CCUcKfyEUTtp5UXvmEMs3pUXAlDd7nT+JHxwxRNoLB21AEK5qNfSESxhT027ugQzIm8dfirzmSfFl45xAQ==\",
119 " \"refresh_auth_info_interval\": 300,
120 " \"refresh_auth_info_support\": true,
121 \"}"
122]
123 },
124 "example-Curl-XML-Output": {
125 "name": "Curl - XML Output",
126 "description": "Использование curl для авторизации (формат ответа: XML)",
127 "request": [
128 "curl -c cookie-jar.txt -d \\\"output=xml&session[login]=Admin&session[password]=12345678\\\" -X POST http://127.0.0.1:8080/login",
129 "",
130 "где:",
131 "- cookie-jar.txt - файл с cookie,"
132],
133 "response": [
134 "<DataList>",
135 " <message>success</message>",
136 " <state_code>200</state_code>",
137 " <authenticity_token>Ta6CCUcKfyEUTtp5UXvmEMs3pUXAlDd7nT+JHxwxRNoLB21AEK5qNfSESxhT027ugQzIm8dfirzmSfFl45xAQ==</authenticity_token>",
138 " <refresh_auth_info_interval>300</refresh_auth_info_interval>",
139 " <refresh_auth_info_support>true</refresh_auth_info_support>",
140 "</DataList>"
141]
142 }

```

*Рис. 41. Пример описания примеров использования API в json файле*

## Examples:

### Curl - JSON Output

Использование curl для авторизации (формат ответа: JSON)



#### Request:

```
curl -c cookie-jar.txt -d "output=json&session[login]=Admin&session[password]=12345678" -X POST http://127.0.0.1:8080/login
```

где:

- cookie-jar.txt - файл с cookie,

#### Response:

```
{
 "message": "success",
 "state_code": 200,
 "authenticity_token": "Ta6CCUcKfyEUTtp5UXvmEMs3pUXAIDd7nT+JHxfwxRNoLB21AEK5qNfSESxhTO27ugQzIm8di",
 "refresh_auth_info_interval": 300,
 "refresh_auth_info_support": true
}
```

### Curl - XML Output

Использование curl для авторизации (формат ответа: XML)

#### Request:

```
curl -c cookie-jar.txt -d "output=xml&session[login]=Admin&session[password]=12345678" -X POST http://127.0.0.1:8080/login
```

где:

- cookie-jar.txt - файл с cookie,

#### Response:

```
<DataList>
<message>success</message>
<state code>200</state code>
```

Рис. 42. Пример возможного отображения примеров использования API

## 19. Система логирования FlyCubePHP

За логирование работы приложения в FlyCubePHP отвечает класс «\FlyCubePHP\Core\Logger\Logger». Данный класс ядра логирования реализует интерфейс PSR-3 и в своей основе использует библиотеку Monolog. Все сообщения, отправляемые в ядро, проходят фильтрацию на значение текущего уровня логирования. Если сообщение ниже текущего уровня логирования, то они отбрасываются.

Перечень доступных уровней логирования, описанных в PSR-3:

- DEBUG - сообщения данного типа содержат подробную информацию об отладке;
- INFO - сообщения данного типа содержат базовую информацию (журналы SQL запросов, log-in пользователя и т.д.);
- NOTICE - сообщения данного типа могут содержать базовую информацию, не входящую в «INFO»;
- WARNING - сообщения данного типа содержат информацию о предупреждениях (использование устаревшего API, старыж библиотек и т.д.);
- ERROR - сообщения данного типа содержат данные ошибок выполнения;
- CRITICAL - сообщения данного типа содержать данные критических ошибок (компонент приложения недоступен, непредвиденное исключение и т.д.);
- ALERT - сообщения данного типа содержат данные для молниеносного принятия решения со стороны администраторов (сайт не работает, база данных недоступна и т.д.);
- EMERGENCY - сообщения данного типа содержат данные с самым высоким уровнем приоритета для принятия решения со стороны администраторов.

Перечень методов, предоставляемых ядром логирования:

- `->setLogger(LoggerInterface $logger)` - задать новый объект системы логирования:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::instance()->setLogger(new \Monolog\Logger('FlyCubePHP'));
```
- `->setLogLevel(int $level)` - задать уровень логирования:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::instance()->setLogLevel(Logger::DEBUG);
```
- `->logLevel()` - пролучить текущий уровень логирования:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
var_dump(Logger::instance()->logLevel());
// 100
```
- `->isValid()` - проверка, является ли корректным объект системы логирования (не является ли он «null»):

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
var_dump(Logger::instance()->isValid());
// true
```

- `->isEnabled()` - проверка, включена ли поддержка системы логирования (будет ли вестись логирование):

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
var_dump(Logger::instance()->isEnabled());
// true
```

- `->logFolder()` - получить полный путь к каталогу для log-файлов:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
var_dump(Logger::instance()->logFolder());
// '/home/test/FlyCubePHProjects/MyProject/log'
```

- `Logger::emergency($message, array $context = array())` - записать сообщение формата «EMERGENCY»:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::emergency('Sample emergency message...', ['context key' => 'context value']);
// [19.10.2021 10:35:08][EMERGENCY] Sample emergency message... {"context key":"context value"}
```

- `Logger::alert($message, array $context = array())` - записать сообщение формата «ALERT»:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::alert('Sample alert message...', ['context key' => 'context value']);
// [19.10.2021 10:37:14][ALERT] Sample alert message... {"context key":"context value"}
```

- `Logger::critical($message, array $context = array())` - записать сообщение формата «CRITICAL»:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::critical('Sample critical message...', ['context key' => 'context value']);
// [19.10.2021 10:38:41][CRITICAL] Sample critical message... {"context key":"context value"}
```

- `Logger::error($message, array $context = array())` - записать сообщение формата «ERROR»:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::error('Sample error message...', ['context key' => 'context value']);
// [19.10.2021 10:40:23][ERROR] Sample error message... {"context key":"context value"}
```

- `Logger::warning($message, array $context = array())` - записать сообщение формата «WARNING»:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::warning('Sample warning message...', ['context key' => 'context value']);
// [19.10.2021 10:41:38][WARNING] Sample warning message... {"context key":"context value"}
```

- `Logger::notice($message, array $context = array())` - записать сообщение формата «NOTICE»:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::notice('Sample notice message...', ['context key' => 'context value']);
// [19.10.2021 10:43:07][NOTICE] Sample notice message... {"context key":"context value"}
```

- `Logger::info($message, array $context = array())` - записать сообщение формата «INFO»:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::info('Sample info message...', ['context key' => 'context value']);
// [19.10.2021 10:44:13][INFO] Sample info message... {"context key":"context value"}
```

- `Logger::debug($message, array $context = array())` - записать сообщение формата «DEBUG»:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::debug('Sample debug message...', ['context key' => 'context value']);
// [19.10.2021 10:45:23][DEBUG] Sample debug message... {"context key":"context value"}
```

- `Logger::log($level, $message, array $context = array())` - записать сообщение с указанием произвольного уровня:

```
use \FlyCubePHP\Core\Logger\Logger as Logger;
Logger::log(Logger::DEBUG, 'Sample debug message...', ['context key' => 'context value']);
// [19.10.2021 10:45:23][DEBUG] Sample debug message... {"context key":"context value"}
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «`FlyCubePHP/src/Core/Logger/Logger.php`».

## 20. Система обработки ошибок FlyCubePHP

Система обработки ошибок в FlyCubePHP построена на базе стандартных механизмов PHP. Для усовершенствования форматов вывода и обработки сообщений об ошибках и исключениях был разработан специальный класс, реализующий данный функционал. «\FlyCubePHP\Core>Error>ErrorHandlingCore» является ядром обработки ошибок и исключений и предоставляет следующий набор методов:

- `ErrorHandlingCore::fileCodeTrace(string $filePath, int $line = -1, int $visibleLines = 10)` - получить отформатированную часть содержимого файла, в котором произошла ошибка, где:
  - `$filePath` - полный путь до файла;
  - `$line` - строка с ошибкой (если -1, то зачитывается все содержимое файла);
  - `$visibleLine` - количество строк до и после ошибки;
- `ErrorHandlingCore::debugBacktrace(array $backtrace)` - получить отформатированную строку со списком вызовов, где:
  - `$backtrace` - результат вызова метода `debug_backtrace()`;
- `->setErrorHandler(BaseErrorHandler $handler)` - задать новый обработчик ошибок.

Для переопределения стандартного обработчика ошибок можно воспользоваться описанным выше методом `«->setErrorHandler(...)`». Для этого новый обработчик ошибок должен быть унаследован от базового абстрактного класса `«\FlyCubePHP\Core>Error\BaseErrorHandler»`. В новом обработчике должны быть реализованы следующие абстрактные методы:

- `evalError(int $errCode, string $errStr, string $errFile, int $errLine, string $errFileContent, string $backtrace)` - метод обработки ошибки, где:
  - `$errCode` - код ошибки;
  - `$errStr` - текст с ошибкой;
  - `$errFile` - файл с ошибкой;
  - `$errLine` - номер строки с ошибкой;
  - `$errFileContent` - часть содержимого файла с ошибкой;
  - `$backtrace` - стек вызовов;
- `evalException(\Throwable $ex)` - метод обработки исключений.

Более подробную информацию о доступных методах и их подробное описание можно найти в файле `«FlyCubePHP/src/Core/Error/ErrorHandlingCore.php»`.

Пример регистрации нового обработчика ошибок:

```
class MyErrorHandler extends \FlyCubePHP\Core\Error\BaseErrorHandler
{
 final public function evalError(int $errCode,
 string $errStr,
 string $errFile,
 int $errLine,
 string $errFileContent,
 string $backtrace): bool {
 //
 // TODO place this your code...
 //
 }

 final public function evalException(\Throwable $ex) {
 //
 // TODO place this your code...
 //
 }
}

// Set new error handler:
\FlyCubePHP\Core\Error\ErrorHandlingCore::instance() -> setErrorHandler(new MyErrorHandler());
```

## 21. Система расширений FlyCubePHP

Различные модули ядра FlyCubePHP поддерживают механизмы расширения базового функционала. Каждый из модулей предоставляет один или несколько базовых абстрактных классов, позволяющих разработать свои модули расширения и загрузить их при старте приложения. Для включения механизма заргузки расширений требуется выставить переменную окружения «FLY\_CUBE\_PHP\_ENABLE\_EXTENSION\_SUPPORT: true» в файле конфигурации приложения «[MyProject]/config/application\_env.conf» (где «MyProject» - название нового проекта):

```

--- extensions settings ---

In production mode:
- enable extension support - default false
- extensions folder - default "extensions"

In development mode:
- enable extension support - default false
- extensions folder - default "extensions"

FLY_CUBE_PHP_ENABLE_EXTENSION_SUPPORT: true
#FLY_CUBE_PHP_EXTENSIONS_FOLDER: "extensions"
```

Поумолчанию корневым каталогом расширений является «[MyProject]/extensions/». Для его изменения требуется задать новое значение в переменную окружения «FLY\_CUBE\_PHP\_EXTENSIONS\_FOLDER». Более подробно о расширении функционала отдельных модулей ядра FlyCubePHP будет рассмотрено ниже в отдельных главах.

### 21.1. Расширение для системы доступа к базе данных

В текущей версии FlyCubePHP ядро системы по работе с базами данных позволяет расширить набор адаптеров по доступу к базе данных, загружая их из каталога «[MyProject]/extensions/db/adapters/». Новый класс адаптера должен быть унаследован от базового абстрактного класса «\FlyCubePHP\Core\Database\BaseDatabaseAdapter» и должны быть реализованы следующие методы:

- `->makeDSN(array $settings)` - метод создания строки с настройками подключения к базе данных для PHP PDO, где:
  - `$settings` - массив с настройками для подключения;
- `->name()` - название адаптера;
- `->tables()` - метод запроса списка таблиц базы данных;
- `->serverVersion()` - метод запроса версии сервера базы данных;
- `->quoteTableName(string $name)` - получить корректное экранированное имя таблицы.

Для регистрации нового адаптера системы, ядро по работе с базами данных предоставляет следующий метод:

- `->registerDatabaseAdapter(string $name, string $className)` - метод позволяет зарегистрировать адаптер по работе с базой данных, где:
  - `$name` - название адаптера, используемое в конфигурационном файле для доступа к БД «[MyProject]/config/database.json»;
  - `$className` - имя класса адаптера с namespace.

Пример регистрации нового адаптера:

```
namespace MyNamespace;

class SQLiteAdapter extends \FlyCubePHP\Core\Database\BaseDatabaseAdapter
{
 public function __construct(array $settings) {
 parent::__construct($settings);
 }

 final public function serverVersion(): string {
 $res = $this->query("select sqlite_version() as version");
 if (!empty($res))
 return $res[0]->version;
 return "";
 }

 final public function tables(): array|null *{
 $res = $this->query("SELECT tbl_name FROM sqlite_master WHERE type = 'table';");
 if (!is_null($res)) {
 $tmpArr = array();
 foreach ($res as $item)
 $tmpArr[] = $item->tbl_name;
 return $tmpArr;
 }
 return [];
 }

 final public function name(): string {
 return "SQLite";
 }

 final public function quoteTableName(string $name): string {
 return "\"$name\"";
 }

 final protected function makeDSN(array $settings): string {
 if (empty($settings))
 return "";
 $database = "";
 if (isset($settings['database']))
 $database = $settings['database'];
 return "sqlite:$database";
 }
}

// Register new adapter in database core:
\FlyCubePHP\Core\Database\DatabaseFactory::instance()->registerDatabaseAdapter('sqlite',
'MyNamespace\SQLiteAdapter');
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Database/BaseDatabaseAdapter.php».

## 21.2. Расширение для системы миграций

В текущей версии FlyCubePHP ядро системы миграций позволяет расширить набор миграторов по работе с базой данных, загружая их из каталога «[MyProject]/extensions/db/migrators/». Новый класс мигратора должен быть унаследован от базового абстрактного класса «\FlyCubePHP\Core\Migration\BaseMigrator». Данный класс является преобразователем кода миграций на PHP в SQL представление для соответствующей базы данных. Базовый класс мигратора предоставляет обширный набор методов, часть из которых уже реализована. Если СУБД, под которую разрабатывается мигратор, не поддерживает те или иные методы, или поведение отличается от описанного в базовом классе, то следует переопределить соответствующие методы с внесением в их реализацию изменений.

Так же базовый класс мигратора предоставляет доступ к следующим внутренним объектам:

- `$_dbAdapter` - объект базового класса адаптера по работе с базой данных «\FlyCubePHP\Core\Database\BaseDatabaseAdapter»; создается автоматически ядром миграций при создании объекта мигратора.

Для регистрации нового мигратора по работе с базой данных, ядро миграций предоставляет следующий метод:

- `->registerMigrator(string $name, string $className)` - метод позволяет зарегистрировать новый мигратор по работе с базой данных, где:
  - `$name` - название адаптера, используемое в конфигурационном файле для доступа к БД «[MyProject]/config/database.json»;
  - `$className` - имя класса мигратора с namespace.

Пример регистрации нового мигратора:

```
namespace MyNamespace;

class SQLiteMigrator extends \FlyCubePHP\Core\Migration\BaseMigrator
{
 public function createDatabase(string $name, array $props = []) {
 if (empty($name) || is_null($this->_dbAdapter))
 return;
 $settings = $this->_dbAdapter->settings();
 $settings['database'] = $name;
 $this->_dbAdapter->recreatePDO($settings);
 }

 public function dropDatabase(string $name) {
 if (empty($name) || is_null($this->_dbAdapter))
 return;
 if (!is_file($name))
 return;
 unlink($name);
 }
}

// Register new migrator in migrations core:
\FlyCubePHP\Core\Migration\MigrationsCore::instance()->registerMigrator('sqlite',
'MyNamespace\SQLiteMigrator');
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Migration/BaseMigrator.php».

## 21.3. Расширение вспомогательных методов шаблонов представлений (views helpers)

Для расширения набора вспомогательных методов шаблонов представлений ядро FlyCubePHP предоставляет возможность загрузки данных классов из каталога «[MyProject]/extensions/controller/helpers/». Все классы расширений, загружаемые как вспомогательные методы шаблонов представлений, должны быть унаследованы от базового абстрактного класса «\FlyCubePHP\Core\Controllers\Helpers\BaseControllerHelper». Все методы данного класса будут автоматически загружены как расширения системы Twig. Для указания дополнительных свойств вспомогательных методов, базовый класс предоставляет следующие возможности:

- `->setFunctionSettings(string $name, array $settings = [])` - задать свойства вспомогательной функции, где:
  - `$name` - название функции;
  - `$settings` - массив свойств с возможными ключами:
    - `safe` - безопасная функция (вывод без экранирования) (default: false);
    - `need_context` - требуется Twig Context (default: false);
    - `needs_environment` - требуется Twig Environment (default: false);
- `->appendSafeFunction(string $name, bool $val = true)` - указать, что функция является безопасной (вывод без экранирования);
- `->appendNeedContext(string $name, bool $val = true)` - указать, что функции требуется Twig Context;
- `->appendNeedEnvironment(string $name, bool $val = true)` - указать, что функции требуется Twig Environment.

**ПРИМЕЧАНИЕ:** задание дополнительных свойств вспомогательных методов должно производиться в конструкторе класса, предоставляющего данные методы.

Пример нового helper-a:

```
class MyHelper extends \FlyCubePHP\Core\Controllers\Helpers\BaseControllerHelper
{
 function __construct() {
 $this->setFunctionSettings('my_f_1', ['safe'=>true]);
 $this->setFunctionSettings('my_f_2', ['need_context'=>true]);
 $this->setFunctionSettings('my_f_3', ['need_context'=>true, 'needs_environment'=>true]);
 }

 public function my_f_1($a, $b) { ... }
 public function my_f_2($context, $a, $b) { ... }
 public function my_f_3(\Twig\Environment $env, $context, $string) { ... }
}
```

Более подробную информацию о доступных методах и их подробное описание можно найти в файле «FlyCubePHP/src/Core/Controllers/Helpers/BaseControllerHelper.php».

## **21.4. Расширение для системы обработки ошибок**

В текущей версии FlyCubePHP ядро обработки ошибок и исключений позволяет изменить текущий обработчик ошибок. Для этого новый обработчик ошибок должен быть унаследован от базового абстрактного класса «\FlyCubePHP\Core>Error\BaseErrorHandler», а файл с реализацией класса должен располагаться в каталоге «[MyProject]/extensions/error\_handling/».

Подробное описание замены стандартного обработчика ошибок приводилось выше в главе [Система обработки ошибок FlyCubePHP](#).

## 22. Обновление проекта до последней версии ядра FlyCubePHP

Начиная с версии 1.3.0 стало возможным автоматическое обновление вашего проекта до последней версии ядра FlyCubePHP. Для этого можно воспользоваться двумя способами, которые будут описаны ниже.

**ПРИМЕЧАНИЕ:** для принудительной установки или переустановки ядра FlyCubePHP используйте флаг «`--force`».

### 22.1. Обновление проекта с версией ядра FlyCubePHP ниже 1.3.0

Для обновления вашего проекта с версией ядра FlyCubePHP ниже 1.3.0 требуется скачать последний релиз проекта с GitHub, разаковать архив и перейти в каталог «`FlyCubePHP/bin/`». Далее выполните команду:

```
FlyCubePHP/bin> php ./fly_cube_php --upgrade --path=[PROJECT FULL PATH]
```

Пример выполнения команды:

```
FlyCubePHP/bin> php ./fly_cube_php --upgrade --path=/home/test/FlyCubePHProjects/MyProject
== FlyCubePHP: Upgrade project ==

Project name: MyProject
Project path: /home/test/FlyCubePHProjects/MyProject
Project ver.: 1.1.0
Latest ver.: 1.3.0

The project will be updated to version 1.3.0. Continue? [yes/no] (yes): y

- Checking project catalogs: OK
- Download latest version [FlyCubePHP]: OK
- Unzip latest version [FlyCubePHP]: OK
- Download requires [Twig]: OK
- Download requires [JShrink]: OK
- Download requires [ScssPhp]: OK
- Download requires [Psr/Log]: OK
- Download requires [Monolog]: OK
- Unzip requires [Twig]: OK
- Unzip requires [JShrink]: OK
- Unzip requires [ScssPhp]: OK
- Unzip requires [Psr/Log]: OK
- Unzip requires [Monolog]: OK

Look at the upgrade files:
- upgrade to v1.2.0: /vendor/FlyCubePHP/UPGRADE-1.2.0.md

Upgrade project to latest version: SUCCESS
== FlyCubePHP =====
```

## **22.2. Обновление проекта с версией ядра FlyCubePHP 1.3.0 или старше**

Для обновления вашего проекта с версией ядра FlyCubePHP 1.3.0 или старше выполните команду:

```
[PROJECT]/bin> ./fly_cube_php --upgrade
```

Пример выполнения команды:

```
MyProject/bin> ./fly_cube_php --upgrade
== FlyCubePHP: Upgrade project ==

Project name: MyProject
Project path: /home/test/FlyCubePHProjects/MyProject
Project ver.: 1.3.0
Latest ver.: 1.4.0

The project will be updated to version 1.4.0. Continue? [yes/no] (yes): y

- Checking project catalogs: OK
- Download latest version [FlyCubePHP]: OK
- Unzip latest version [FlyCubePHP]: OK
- Download requires [Twig]: OK
- Download requires [JShrink]: OK
- Download requires [ScssPhp]: OK
- Download requires [Psr/Log]: OK
- Download requires [Monolog]: OK
- Unzip requires [Twig]: OK
- Unzip requires [JShrink]: OK
- Unzip requires [ScssPhp]: OK
- Unzip requires [Psr/Log]: OK
- Unzip requires [Monolog]: OK

Look at the upgrade files:
- upgrade to v1.4.0: /vendor/FlyCubePHP/UPGRADE-1.4.0.md

Upgrade project to latest version: SUCCESS

== FlyCubePHP =====
```

## 23. Рекомендации по настройке и развертыванию приложения

Разработанное приложение можно запустить под различными web-серверами, поддерживающими PHP. FlyCubePHP при создании нового проекта автоматически генерирует конфигурационный файл для web-сервера Apache 2.4.

Для корректного запуска приложения рекомендуется выполнить следующие шаги:

1. Скопировать проект в каталог «/opt»:

```
opt> sudo cp -r /home/test/FlyCubePHProjects/MyProject /opt/
```

2. Для работы с использованием web-сервера Apache требуются пакеты:
  - apache2 >= 2.4.33
  - apache2-prefork >= 2.4.33
  - apache2-utils >= 2.4.33

3. Включить модули RewriteEngine и RewriteRule для Apache:

```
#> a2enmod rewrite
```

**ПРИМЕЧАНИЕ:** Ошибки в apache вида:

```
Invalid command 'RewriteEngine', perhaps misspelled or defined by a module not included in the server configuration
Invalid command 'RewriteRule', perhaps misspelled or defined by a module not included in the server configuration
```

означают, что не включен модуль для Rewrite правил.

4. Скопировать конфигурационный файл в каталог web-сервера:

```
MyProject> cp my_project.apache24.conf /etc/apache2/conf.d/my_project.conf
```

5. Произвести редактирование путей до проекта в конфигурационном файле для web-сервера.
6. Произвести генерацию секретного ключа приложения:

```
MyProject/bin> ./fly_cube_php --secret | grep -E --only-matching -e "[0-9a-f]{128}" >/config/secret.key
```

7. Произвести редактирование конфигурационного файла приложения «[MyProject]/config/application\_env.conf», указав режим работы «production mode»:

```

--- env type ---

NOTE: default env-type: development
NOTE: supported env-types: production / development

NOTE: enable 'FLY_CUBE_PHP_ENV: production' in production mode!

FLY_CUBE_PHP_ENV: production
```

8. Произвести редактирование остальных конфигурационных файлов приложения, расположенных в каталоге «[MyProject]/config/», если это требуется.
9. Выполнить очистку log-файлов:

```
MyProject/bin> sudo ./fly_cube_php --clear-logs
```

10. Выполнить очистку кэша приложения:

```
MyProject/bin> sudo ./fly_cube_php --clear-cache
```

11. Выполнить сборку ресурсов и обновление кэша приложения:

```
MyProject/bin> ./fly_cube_php --assets-precompile --env=production
```

12. Выполнить миграцию базы данных:

```
MyProject/bin> ./fly_cube_php --db-migrate --env=production
```

13. Выполнить смену прав каталога и файлов приложения для пользователя, под которым работает web-сервер (для apache2 в OpenSUSE: пользователь «wwwrun», группа «wwwrun»):

```
opt> sudo chown wwwrun:wwwrun -R /opt/MyProject
```

14. Выполнить перезапуск web-сервера:

```
#> sudo systemctl restart apache2
```

## 15. Выполнить проверку работоспособности web-сервера:

```
#> sudo systemctl status apache2
● apache2.service - The Apache Webserver
 Loaded: loaded (/usr/lib/systemd/system/apache2.service; enabled; vendor preset: disabled)
 Active: active (running) since Wed 2021-10-20 10:50:27 MSK; 1h 17min ago
 Main PID: 1970 (httpd-prefork)
 Status: "Total requests: 0; Current requests/sec: 0; Current traffic: 0 B/sec"
 Tasks: 6
 CGroup: /system.slice/apache2.service
 ├─1970 /usr/sbin/httpd-prefork -DSYSCONFIG ...
 ├─2117 /usr/sbin/httpd-prefork -DSYSCONFIG ...
 ├─2118 /usr/sbin/httpd-prefork -DSYSCONFIG ...
 ├─2119 /usr/sbin/httpd-prefork -DSYSCONFIG ...
 ├─2120 /usr/sbin/httpd-prefork -DSYSCONFIG ...
 └─2121 /usr/sbin/httpd-prefork -DSYSCONFIG ...

oct 20 10:50:26 test.my.com systemd[1]: Starting The Apache Webserver...
oct 20 10:50:27 test.my.com systemd[1]: Started The Apache Webserver.
```