**Документация программного комплекса PhpZero**

# Главная

## Просто

CMF “PhpZero” относительно проста по своему устройству и не требовательна к программной среде и ресурсам. Это позволяет быстро освоиться и начать ее использовать.

## Быстро

CMF “PhpZero” обладает всем необходимым набором инструментария для разработки конкретного пользовательского функционала. Скорость работы системы высока за счет лаконичного построения и взаимодействия компонентов между собой. А также их простоты.

## Универсально

В системе реализован необходимый минимум для самой работы системы.

Простая и в тоже время гибкая и функциональная система прав и пользовательских условий.

Все остальное это инструментарий (с возможность переопределения и расширения) универсально подходящий для создания практически любого проекта в сети интернет.

# О проекте

Рады видеть Вас на нашем сайте.

Сайт посвящен программному комплексу (далее система) CMF “PhpZero”

CMF “PhpZero” – это каркас для проектирования систем управления контентом.

На его основе можно создавать системы управления содержимым (CMS), а также другие веб-приложения и сервисы как-то: сайты визитки, интернет магазины, информационные сайты, веб ориентированные ERP и/или CPM системы т. д.

## Преимущества

Если основная задача универсальных CMS — простота создания сайта без вмешательства программиста,

CMF “PhpZero” — это конструктор CMS (в том числе узкоспециализированных) для программиста. Благодаря такому подходу сайт, разрабатываемый с помощью CMF, по сравнению с сайтом на базе CMS, может иметь более простую и безопасную в работе административную панель (в которой отсутствуют функции настройки сайта под любые функции) и быть менее требовательным к ресурсам системы (каждый разрабатываемый модуль реализует именно те функции, которые необходимы в работе сайта).

При наличии грамотно спроектированной CMF и готовых базовых модулей, программисту остаётся только доработка (по необходимости отсутствующих функций) и сборка этих модулей в единое пространство с оформлением вывода в соответствии с пожеланиями клиента, то есть разработка сайта на основе CMF вполне сравнима в большинстве случаев по трудоёмкости с разработкой сайта на универсальной CMS. Но c очевидными преимуществами в производительности, управляемости и что самое главное дальнейшей масштабируемости.

С другой стороны при работе с голыми фреймворками пользователь-программист может столкнуться с программными ошибками. Не будучи экспертом в архитектуре сложного фреймворка, он может испытывать затруднения с их локализацией, не может быть уверен, сделал ли он ошибку сам, или столкнулся с ошибкой в фреймворке. Глубокие знания архитектуры фреймворка могут понадобиться и при его тонкой настройке, если программист желает получить максимальную производительность.

## Архитектура

CMF “PhpZero” на уровне файловой структуры реализована паттерном MVC

Веб-фреймворк обеспечивает прозрачную интеграцию всех трёх слоёв MVC архитектуры.

Фреймворк скрывает от программиста детали подключения к базе данных и формирования веб страниц с помощью шаблонов-представлений, позволяя ему сконцентрироваться на реализации бизнес-логики. Процесс создания приложения с использованием MVC-фреймворка заключается в написании классов контроллеров, моделей и представлений, каждый из которых является наследником базовых классов для компонентов каждого слоя. Что в свою ускоряет разработку благодаря использованию наследуемого функционала. CMF “PhpZero” обеспечивает полуавтоматическое создание прототипа веб-приложения, выполняющего основные (CRUD) операции с определенным путём скаффолдинга (метапрогарммирвания).

Любой компонент системы можно переопределить частично либо полностью!

Классы слоя Model могут быть реализованы по разным технологиям.

Одна из основных это технология ORM (Объектно-реляционное отображение)

ORM — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

### Модели

Взаимодействие модели с различными компонентами системы реализовано с помощью паттерна Композиция с делегированием. В основе создания моделей лежит паттерн фабрика и фабричный метод. Что позволяет переопределять и гибко настраивать последние.

### Контролеры

В основе (базовом контролере) работы любого контролера лежит паттерн Цепочка команд (чанки).

Что позволяет разбивать логические участки кода на чанки (куски) и гибко управлять их выполнением. Также это позволяет избежать дублирования кода при полиморфизме (необходимости продублировать большую часть кода из-за не большего изменения в логике наследуемого контроллера).

Оптимизация и ускорения работы при таком подходе на лицо.

### Виды или представления

В систему встроен простой и в тоже время универсальный шаблонизатор.

Который позволяет использовать обычный php синтаксис удобно обрамленный в сигнатуру верстки.

Одна из его сильных сторон реализация директивы плагинов. Что позволяет иерархически (рекурсивно) подключать их друг в друга и тем самым использовать один и тот же плагин многократно в любом месте, где это нужно без дублирования кода. Плагины (plugin|bar) это те же контроллеры, их упрощенная разновидность. Что в свою очередь уменьшает вероятность ошибок и сокращает время разработки. Для многократного подключения простых блоков верстки реализована директива подключений (inc|include).

Также реализована директива переводов (translation|lang)

Работа и взаимодействие вышеописанной архитектуры и его частей в целом централизованно производится и управляется компонентом приложения (Zero\_App) .

Этот компонент инициализирует систему и среду приложения, запускает на выполнение контроллеры,

формирует вывод результата и отдает его браузеру. Также отслеживает и фиксирует всю производимую работу и возникающие ошибки.

# Быстрый старт (Quick start)

В данном разделе будет описано, как максимально быстро установить систему и начать с ней работать.

## Установка системы

Предполагается, что у Вас уже имеется настроенный UNIX хостинг (БД Mysql, Php, Apache или Nginx).

Работающий на нем домен.

1. Скачиваем CMF “PhpZero” в виде zip архива [здесь](https://github.com/kshamiev/phpzero) и распаковываем. Или клонируем в корень сайта. В одноименную папку “phpzero”. К примеру: /home/domains/www.domain.com/www/phpzero
2. Запускаем процесс инсталляции, пройдя по ссылке: [http://www.domain.com/phpzero/setup](http://you-domain.com/phpzero/setup)

Следуем инструкциям установки

1. Все. Заходим на наш сайт.

### Базовый интерфейс системы

После установки системы управляющая часть сайта далее CP будет доступна по адресу (по умолчанию)

[http://www.domain.com/zero](http://you-domain.com/zero)

В системе изначально заведен системный пользователь. Логин и пароль “dev”

Системные разделы:

Система

Файловый менеджер Файловый менеджер. Редактирование текстовых файлов.

Обслуживание Обслуживающий инструментарий системы.

Контроллеры Контроллеры сайта (синонимы шаблонам).

Опции Опции проекта (Различные параметры)

Реквизиты API Реквизиты доступа к API внешних ресурсов

Пользователи

Пользователи Работа с пользователями

Группы Работа с группами и правами доступа

Сайт

Разделы Управление разделами (страницами) сайта (роутинг), контентом разделов.

Контент Контент блоки на страницах сайта (реализуются через плагины)

*Примечание:*

Самую актуальную версию CMF “PhpZero” можно скачать отсюда (ветка master):

<https://github.com/kshamiev/phpzero>

## Принятая нумерация версий

X.X.X или R.A.B

R (Release) - Первая цифра обозначает номер релиза. Итерация этого порядка привязывается к завершению этапа разработок и выкатыванию. Как правило, это происходит при разработке каких-либо новых функциональных возможностей (модулей) или фундаментальных изменений самой системы. Или когда их количество и качество становится слишком большим.

A (Advanced) – Вторая цифра обозначает номер дополнения к релизу. Итерация этого порядка происходит при разработке нового или доработке и расширении ранее реализованного функционала.

Эта цифра обнуляется во время итерации первой цифры R (при выкатывании нового релиза).

B (Bugfix) – Третья цифра обозначает номер исправленной ошибки в работающем проекте. Итерация этого порядка происходит при исправлении выявленных ошибок в уже работающем проекте.

Эта цифра обнуляется во время итерации первой цифры R (при выкатывании нового релиза).

# Руководство (Guide, Manual)

## Терминология

**Определение, Утверждение:**

Таким заголовком обозначаются различного рода утверждения и определения, используемые в системе.

*Примечание, Совет:*

Таким заголовком обозначаются дополнительные сведения, раскрывающие тонкости содержания раздела и/или рекомендации

*Важно:*

Таким заголовком обозначаются важные предупреждения, в том числе и критические. Не соблюдение которых, может привести к потере данных или работоспособности.

Инструкции, конструкции

Таким текстом пишутся конкретные конструкции языка, системы и примеры программного кода.

## Технические директивы и стиль кодирования

Информация написанная в данном разделе и его подразделах носит рекомендательный характер.

И строго справедливо только для ядра системы.

Стиль кодирования для IDE PhpStorm (/phpzero/data/phphstorm\_code\_style.jar).

Имена переменных, констант, функций, классов, методов в подавляющем большинстве состоять из 3 словоформ. Наличие каждого из них и их порядок опциональны и зависят от контекста.

1. Группирующий префикс, обозначающий принадлежность, к какому либо значимому объединению, либо действию. Например: имя модуля, метод действия, чанк, системная переменная, операции получения, присвоения, загрузки.
2. Имя нарицательное. Собственно с чем работаем. Например: новость, группа, товар, раздел.
3. Суффикс, определяющий характер операции. Например: Добавление, удаление, калькуляция

$shop\_order\_count;

$order\_count;

$count;

Переменные, содержащие в себе публичные данные - используемые во многих местах пишутся с заглавной буквы.

Переменные, содержащие в себе приватные данные - используемые инкапсулировано в небольшом участке кода, алгоритмичные пишутся со строчной буквы.

Имена всех переменных могут использовать верблюжью нотацию либо символ нижнего подчеркивания.

Рекомендуется верблюжья нотация.

Имена констант пишутся в верхнем регистре и используют символ нижнего подчеркивания.

Все имена любых переменных и констант должны быть правильными словами языка. Сугубо личные ассоциации в названиях, не понятные для большинства категорически не приветствуются.

### Определенные имена переменных

$Object, $Model Абстрактный объект (новость, пользователь, товар).

$View Представление

$source, source\_name Имя источника (таблицы) хранящей целевые объекты

Пользовательские функции начинаются с префикса app\_ и располагаются в app/function.php

Системные функции начинаются с префикса zero\_ и располагаются в phpzero/function.php

Пользовательские константы начинаются с префикса APP\_ и располагаются в app/constant.php

Системные константы начинаются с префикса ZERO\_

API контроллеры содержат в своем имени суффикс Api

Консольные контроллеры содержат в своем имени суффикс Console или Cron

## Файловая структура

В основе реализации файлового представления системы лежит паттерн MVC.

Имена таблиц, их моделей, контроллеров, шаблонов представления, хранилища бинарных данных и кеша как правило в корне совпадают (но не аксиома). Удобство реализации и простота понимания.

Старая структура:

/application

Область модулей приложения

/modulName Модуль

Стандартный модуль содержит в себе следующие папки:

/assets Хранилище оформления шаблонов представления (img/css/js)

/class Модели, компоненты и контроллеры.

/ModelName.php Модель

/ComponentName.php Компонент

(характерно для базового модуля приложения -zero)

/ControllerFoder Контроллеры модуля во вложенных папках

/ControllerName.php

…

/i18n – Переводы

/view – Представления модуля

В модуле могут быть и другие папки с нужным по смыслу и функционалу содержанием

config.php – Общая конфигурация для всего приложения (может лежать в корне сайта).

function.php – Пользовательские функции приложения.

routeApi.php - Конфигурация роутинга Api контроллеров (в случае работы без БД).

routeWeb.php - Конфигурация роутинга Web контроллеров (в случае работы без БД).

routeConsole.php - Конфигурация роутинга Cron контроллеров (в случае работы без БД).

Новая структура:

/app

/assets Хранилище оформления шаблонов представления (img/css/js)

/i18n Переводы

/class Модели, компоненты и контроллеры.

/modulName Модуль

/ModelName.php Модель

/ComponentName.php Компонент

(характерно для базового модуля приложения -zero)

/ControllerFoder Контроллеры модуля во вложенных папках

/ControllerName.php

…

/view Представления и виджеты

/modulName Модуль

/LayoutName.php Шаблон макета

/ViewFoder Шаблоны модуля во вложенных папках

/ViewName.php

…

Могут быть и другие папки с нужным по смыслу и функционалу содержанием

config.php – Общая конфигурация для всего приложения (может лежать в корне сайта).

function.php – Пользовательские функции приложения.

routeApi.php - Конфигурация роутинга Api контроллеров (в случае работы без БД).

routeWeb.php - Конфигурация роутинга Web контроллеров (в случае работы без БД).

routeConsole.php - Конфигурация роутинга Cron контроллеров (в случае работы без БД).

/assets

Хранилище оформления и клиентского интерактива сайта (img/css/js).

/exchange

Служебная папка для импорта и экспорта данных в систему и из нее.

(папка может располагаться на уровень выше)

/upload

Хранилище загружаемых бинарных данных пользователем.

/phpzero

Система или ядро. Его структура аналогична /app

/console.php Менеджер консольных задач и их выполнение

/function.php Системные функции приложения.

.htaccess – Дополнительные настройки для сервера apache

index.php – Точка входа в приложение (web, api, console).

За пределами DOCUMENT\_ROOT

/cache

Хранилище кешированных данных. Содержимое не важно, и может быть удалено в любой момент.

(папка может располагаться на уровень выше)

/logs

Профилированные статистические данные работы приложений.

(папка может располагаться на уровень выше)

Все программные компоненты представлены классами (ОПП).

Их подключение происходит автоматически при первом обращении к ним через автозагрузчик.

Алгоритм поиска и подключений классов на примере Zero\_Users\_Profile:

1. В области приложений (/application). По следующей маске: zero/class/Users/Profile.php
2. В области приложений (/app). По следующей маске: class/Zero/Users/Profile.php
3. В области ядра (/phpzero). По следующей маске: class/Zero/Users/Profile.php

Представления реализованы в виде html шаблонов.

Структура и алгоритм поиска шаблонов на примере шаблона Zero\_Users\_Profile:

1. В области приложений (/application): zero/view/Users/Profile.html
2. В области приложений (/app): view/Zero/Users/Profile.html
3. В области ядра (/phpzero): view/Zero/Users/Profile.html

Оформление располагается в папках (/assets). Структура на усмотрение разработчика.

## Приложение

Алгоритм работы системы в целом описан в компоненте приложение (Zero\_App)..

### Модули (установка не реализовано)

При работе с модулями перечитывается каталог (/application), в котором они находятся.

Справочная информация по модулям читается из их конфигурационных файлов.

Каждый модуль проверяется на статус инсталляции наличием в нем файла:

ModulName/setup/INSTALL (в нем хранится версия установленного модуля).

И выводится в виде списка модулей со следующим функционалом:

Если модуль не установлен его можно установить.

Если он установлен его можно удалить.

*Совет:*

1. Разработку модуля надо начинать с проектирования БД. Не экономьте на этом время.
2. После завершения проектирования в разделе Система - Обслуживания автоматизировано сформируете новый модуль и его модели.
3. На базе сформированных моделей и необходимого функционала (задач) создавайте контроллеры. По необходимости дополняя модуль библиотеками, хелперами.
4. Наиболее удачные модули могут быть включены в официальные модули CMF PhpZero

#### Обновление БД

В отличие от программной части, которая при обновлении просто копируется. БД так обновлять нельзя по причине наличия данных. Причем обновляемый модуль может отставать на несколько версий.

В каждой из которых могут быть произведены изменения в структуре БД.

То есть необходимо выполнение миграции БД по каждой версии отдельно. Начиная с исходной до обновляемой. Поскольку последнее изменение будет опираться на предыдущую версию БД которая отсутствует в исходном модуле более старой версии.

Поэтому структура и алгоритм обновления БД следующие:

ModulName/setup/schema/

mysql.sql полная схема БД модуля текущей версии.

mysql\_1.1.1.sql миграционный скрипт для указанной версии

mysql\_1.1.2.sql миграционный скрипт для указанной версии

mysql\_1.2.1.sql миграционный скрипт для указанной версии

mysql\_2.4.6.sql миграционный скрипт для указанной версии

*Примечание:*

Известно, что миграция данных очень сложный вопрос, который здесь подробно обсуждать не имеет смысла.

Поэтому выше написанный раздел справедлив для простых случаев и стандартных модулей. В остальных случаях как правило это ручная работа.

mysql\_0.0.0.sql этот скрипт предназначен для удаления модуля из системы (удаление его из БД).

## Архитектура проектирования БД (рекомендации)

### Таблицы

В именах таблиц используются буквенные символы.

Не рекомендуется использование символа нижнего подчеркивания и цифр.

### Свойства

В каждой объектной таблице первым полем всегда идет идентификатор:

(не кросс таблица)

ID

После него (если таковые имеются). Формируются поля связи с другими таблицами.

Правило их именования: (Имя целевой таблицы + “\_” + ID)

GoodCatalog\_ID, Groups\_ID

Для решения конфликтных множественных одноименных связей, имена полей могут быть расширены уникальными однобуквенными суффиксами.

GoodsP\_ID, GoodsC\_ID

*Важно:*

Правило (сигнатура) именования связующих полей строго для соблюдения. Так как абстрактная часть системы повсеместно на это опирается в своей работе. Также на этой архитектуре строятся системы прав.

Далее располагаются все остальные прямые поля (свойства объекта)

Имена этих полей могут состоять только из латинских символов и цифр

Можно использовать верблюжью нотацию.

Name, Description, Price. IsAccess

*Важно:*

Символ нижнего подчеркивания разрешен только в полях связи.

#### Реализация абстрактных горизонтальных прав

Обычные поля связи могут выступать в роли УП если они спроектированы в таблице пользователей.

CarsMarka\_ID (пользователь работает только с моделями определенной марки)

Структура хранения пользователей каталогизирована. Это позволяет реализовывать реальную и необходимую структуру компании в частности и целом. Впрочем, это относится ко всем пользователям проекта. Условие пользователя (если оно установлено) включает самого пользователя и его подчиненных (связанных с ним родительской связью) на один уровень вниз.

Такое решение позволяет организовывать систему прав как между различными подразделениями (группами) по функциональному назначению, так и в внутри одно подразделения дифференцировать согласно ее иерархии.

Тонкости реализации:

CarsMarka\_ID

УП определяется по наличию полей связи в структуре таблицы и статуса УП а также установленным в них значениям.

При обнаружении одноименных полей связи в целевой таблице УП начинают работать.

## Компоненты

Обзор компонентов системы:

1. Приложение ( Zero\_App )
2. Система кеширования ( Zero\_Cache )
3. Конфигурация ( Zero\_Config )
4. Контроллер базовый абстрактный ( Zero\_Controller )
5. Нативная работа с БД ( Zero\_DB )
6. Объектная работа с БД ( Zero\_AR )
7. Инженеринг ( Zero\_Engine )
8. Фильтр для моделей ( Zero\_Filter )
9. Интернационализация ( Zero\_I18n )
10. Логирование ( Zero\_Logs )
11. Модель базовая абстрактная ( Zero\_Model )
12. Сессия ( Zero\_Session )
13. Валидатор свойств моделей ( Zero\_Validator )
14. Представление - шаблонизатор ( Zero\_View )
15. Опции приложения ( Zero\_OptionsV )
16. Ответ. Вывод результатов работы приложения ( Zero\_Response )
17. Запрос. Реализация запросов к сторонним ресурсам ( Zero\_Request )

### Приложение (Zero\_App)

#### Назначение

Используемый класс Zero\_App

Реализует выполнение и управление приложением в целом.

Является его основой и содержит в себе ключевые данные для его работы.

Инициализирует системные константы.

#### Реализация

Браузер:

Точкой входа в приложение является файл index.php находящий в корне сайта.

В нем подключается сам компонент.

Далее запускается инициализация приложения и его выполнение.

Инициализация приложения:

Zero\_App::Init()

1. Подключается конфигурация и происходит инициализация программного окружения и глобальных настроек. Прямое назначение свойств компонента (Zero\_Config) значениями из массива конфигурации.
2. Подключение системных и пользовательских функций (…/function.php)
3. Определяется автозагрузчик классов
4. Определяется обработчик исключений
5. Определяется режим работы приложения (Web, Api, Console)
6. Разбирается роутинг, язык, параметры запроса. На основании запрошенного uri
7. Инициализируются компоненты:

Zero\_App::$Config Конфигурация

Zero\_App::$Users Пользователь

Zero\_App::$Section Раздел

Zero\_App::$Controller Контроллер

Zero\_App::$RouteParams Параметры uri

Zero\_App::$Request Запросы к внешним ресурсам (загрузка реквизитов)

Zero\_App::$Options Опции приложения (загрузка)

Zero\_Logs Подсистема мониторинга и профилированного работы приложения

Zero\_DB Работа с БД

Zero\_Cache Подсистема кеширования

Zero\_Session Сессия

Выполнение приложения:

Zero\_App::Execute() (использование БД)

1. Режим обслуживания приложения по IP адресу. Если указаны в конфигурации
2. Ограничение работы приложения по IP адресу. Если указаны в конфигурации
3. Стандартная авторизация через заголовки. Если заданы логи и пароль в конфигурации
4. Инициализация пользователя (Zero\_Users). Если передан токен то по нему.
5. Инициализация запрошенного раздела (страницы сайта) на основе роутинга (Zero\_Section).
   1. Проверка существования (404)
   2. Проверка редиректа (301)
   3. Проверка прав на раздел (403)
   4. Инициализация контроллера
   5. Если контроллер указан проверка прав на действие (403)
6. Инициализация контроллера на основе роутинга. (! Если перед этим раздел не был найден !)
   1. Проверка существования (404)
   2. Проверка прав на действие (403)
7. Если контроллер указан. То инициализация его как исполняемого (Zero\_Controller)
8. Работа контроллера. Выполнение метода действия.
9. Сборка макета страницы и вставка результата работы контроллера. (! если не api !)

Вывод:

Реализован централизованный метод отправки необходимых заголовков браузеру.

И выдача результата в нужном формате (по умолчанию html).

Отдаваемым результатом можно управлять внутри контроллера.

Что и происходит в контроллерах API

Консоль:

Точкой входа в консольное приложение является файл /phpzero/console.php.

Инициализация приложения аналогична обычному браузерному приложению.

После нее непосредственно выполняется консольный контроллер.

Либо отрабатывает менеджер запуска консольных задач.

#### Зависимости

Zero\_Config, Zero\_Logs, Zero\_Cache, Zero\_Session, Zero\_Section, Zero\_Users, Zero\_Controller, Zero\_View,

### Кеш (Zero\_Cache)

#### Назначение

Используемый класс Zero\_Cache

Обеспечить увеличение производительности за счет выдачи сформированного ранее и сохраненного контента.

#### Реализация

Инициализация компонента происходит во время инициализации приложения.

Во время инициализации определяется хранилище данных кеша:

1. Хранение данных в файловой структуре
2. Хранение данных в оперативной памяти (Memcache)

Компонент может работать в двух режимах:

1. Через модель, данные которой нужно кешировать.
   1. $Model->Cache->Set($index, $data, $time)
   2. $Model->Cache->Get($index, $time)
   3. $Model->Cache->Reset()
2. Напрямую
   1. Zero\_Cache::Set\_Data($index, $data, $time)
   2. Zero\_Cache::Get\_Data($index, $time)

Реализован механизм связанного или зависимого кеша.

Zero\_Cache::Set\_Link($source\_name, $id);

Правило формирование индекса:

$Host хост сайта (устанавливается автоматически)

$Source источник (через модель устанавливается автоматически)

$id идентификатор объекта (через модель устанавливается автоматически)

$index вольная строка (указывается в любом случае)

Примеры:

$Model->Cache->Get(‘Content’, 600);

$Model->Cache->Set(‘Content’, $Value, 600);

Zero\_Cache::Get\_Data(‘Zero\_Section/www/paga/page/url’, 600);

Zero\_Cache::Set\_Data(‘Zero\_Section/www/paga/page/url’, $Value, 600);

Zero\_Cache::Get\_Data(‘Zero\_Section/EnumSet/IsVisible/ru-ru’, 600);

Zero\_Cache::Set\_Data(‘Zero\_Section/EnumSet/IsVisible/ru-ru’, $Value, 600);

600 – время жизни кеша (по умолчанию установлен 0 – вечный кеш)

*Важно:*

Время жизни кеша нужно указывать и при получении и при его сохранении кеша.

В главном конфигурационном файле можно управлять использованием компонента в целом

#### Зависимости

Zero\_App, Zero\_Helper

### Конфигурация (Zero\_Config)

#### Назначение

Используемый класс Zero\_Config

Конфигурирование приложения и инициализация программной среды.

В этом же файле после компонента располагаются служебные функции самого общего системного назначения. Их названия начинаются с префикса zero\_

#### Реализация

Инициализация компонента происходит во время инициализации приложения.

Запрашивается агрегированный массив конфигурации конкретного приложения с базовой конфигурацией.

Параметры конфигурации приложения переопределяют базовые параметры конфигурации.

Проверяется общая авторизация (если установлена) на все приложение в целом.

После чего происходит прямое присвоение свойствам конфигуратора значений массива.

В конце происходит настройка программного окружения и профилированных обработчиков служебных событий (исключения, перехват программный ошибок и завершения работы).

#### Зависимости

Zero\_App, Zero\_Logs

### Контроллер (Zero\_Controller)

#### Назначение

Используемый класс Zero\_Controller.

Компонент является базой для создания конкретных контроллеров решающих прикладные задачи (реализующих бизнес логику) проекта. Имеет 3 предопределенных рабочих свойства:

$ModelОбрабатываемая модель (объект).

$View Представление (возвращается контроллером на выходе)

$Params Массив служебных параметров (принимает в него параметры на входе).

Контроллеры делятся на 3 вида:

Api контроллеры,

Web контроллеры (обычные контроллеры которые возвращают представление),

Console консольные контроллеры (запускаемые по крону)

#### Реализация

Контроллеры создаются через фабричные методы с сохранением в сессии (Factory) и без (Make).

Главный контроллер запрошенный по роутингу, выполняется на уровне приложения, работает через сессию.

Точкой входа (выполнения) контроллера является запрошенный (определенный) метод действия.

Методы Чанки:

Реализован (заложен) механизм разделения программного кода контроллера на логически и функционально завершенные блоки кода в отдельные методы (Чанки) и управление ими через методы действия.

Начинаются с префикса Chunk\_

Такие методы должны возвращать булево значение (false or true).

Это позволит управлять ходом выполнения последовательности чанков

Примеры:

Chunk\_Init() инициализация входных параметров, какие-то общие проверки…

Chunk\_View() формирование представления, …

Методы Действия:

Это управляющие методы. Реализующие собственно управление бизнес (функциональной) логикой приложения. Вызываются централизовано системой либо там где это нужно в других методах действия.

Эти методы могут быть статичны.

Начинаются с префикса Action\_

Примеры:

Action\_Default() контроллер по умолчанию

Action\_Save() сохранение чего-либо

Action\_Remove() удаление чего-либо

Внутри выполнения каждого метода к. можно сохранять сообщения о ходе выполнения его работы.

В дальнейшем их можно вывести пользователю и в профилированные лог файлы. Что и реализовано системно.

$Controller->GetMessage()

$Controller->SetMessage(…)

$Controller->SetMessageError(…)

Возвращаемые значения методами действия:

Api Данные в формате json

Web Объект View либо собранный шаблон с данными (string)

Console Числа 0 (ок) или 1 (ошибка)

Разработка:

В системе реализованы абстрактные контроллеры для выполнения стандартных операций с данными. (CRUD)

Zero\_Controller -> Zero\_Crud\_Grid -> Info\_Article\_Grid

Zero\_Controller -> Zero\_Crud\_Edit -> Info\_Article\_Edit

Грид работает в 4 режимах

1. Работа с данными без перехода по связи
2. Работа с данными с переходом по связи один ко многим
3. Работа с данными с переходом по рекурсивной связи один ко многим (каталог)
4. Работа с данными с переходом по связи многие ко многим

Режим работы как и другие настройки определяются/переопределяются конкретным контроллером.

Шаблоны контроллеров для разработок лежат здесь:

/phpzero/zero/data

Контроллер, работающий по связи должен располагаться в той же папке, в которой лежат контроллеры работающие с моделью без переходов по связи.

zero/class/Content

EditLayout.php

EditSection.php

GridLayout.php

GridSection.php

Таким образом, он точно идентифицирует режим работы и необходимые настройки. Поскольку одновременно может существовать контроллер, работающий с теми же объектами без переходов по связи (напрямую).

Edit.php

Grid.php

#### Зависимости

Zero\_Session, Zero\_I18n, Zero\_App, Zero\_View

### Плагин (Zero\_Plugin)

#### Назначение

Используемый класс Zero\_Plugin.

Компонент является базой для создания конкретных плагинов реализующий вспомогательный и общий функционал, для решения прикладных задач проекта. Имеет 2 предопределенных рабочих свойства:

$View Представление (возвращается плагином на выходе)

$Params Массив служебных параметров (принимает в него параметры на входе).

Плагины, это упрощенные контроллеры, их главное отличие отсутствие основных действий (таковых методов). Также при своей работе они не сохраняются в сессию.

Их задача реализация вспомогательного функционала и представления для и вокруг целевого контроллера выполняющего основную работу.

#### Реализация

Как было написано выше, плагин это тот же контроллер. Поэтому здесь будет описано только принципиальное отличие от него.

Создаются через фабричный метод без сохранения в сессии (Make).

Плагин не имеет и не выполняет никаких методов основных действий.

При создании плагина вторым необязательным параметром, можно передать дополнительные данные для его работы. Эти параметры сохраняются в служебном свойстве $Params.

Это очень часто нужно и используется при рекурсивном вызове плагинов в представлении. Когда оно собирается. К примеру: пагинатор (постраничный вывод), или комментарии к статье.

В основной своей массе плагины работают от представления через директиву plugin.

Когда оно формирует конечный результат.

Благодаря идентичности контроллеру, плагин может также выполняться и как основной контроллер раздела (но без действий). В этом статусе он также как и контроллер может управлять результатом вывода.

Примеры где использование “легкого” плагина предпочтительней чем “тяжеловеса” контроллера:

Вывод защитной картинки от ботов: Zero\_Users\_Kcaptcha

Загрузка файла или показ картинки из БД: Zero\_Crud\_UploadFile

Контент раздела: Zero\_Content\_Page

#### Зависимости

Нет

### БД (Zero\_DB)

#### Назначение

Используемый класс Zero\_DB

Инкапсулирует в себе всю работу с БД.

Централизованное выполнение запросов.

Мониторинг и снятие статистики их выполнения.

Реализует механизм работы с хранимыми процедурами.

Реализует работу с БД на уровне объектов. (ORM)

#### Реализация

Инициализация соединения с БД происходит при первом обращении к компоненту.

Содержит в себе следующие группы методов

* Метод инициализации соединения с БД (Init).
* Методы прямых запросов с формированием результата в нужно виде (Query\_).
* Методы проверок входных данных для запросов (I,F,T,S,E,B,D).
* Методы формирования условия для последующих запросов (работает от модели Sql\_)
* Методы выборки данных в нужном виде по заданным условиям (работает от модели Select\_)
* Методы загрузки, обновления и удаления по заданным условиям (работает от модели)

С компонентом можно работать в двух режимах.

1. Напрямую посылая запросы в определенные для этого статичные методы. Обычно используется при формировании сложных (реляционных) запросов.
2. От модели, к которой и формируются запросы. Через ее свойство, которое является объектом компонента.

*Совет:*

Сложные и уникальные запросы хранятся в статичных методах той модели, к которой преимущественно и относится запрос. Такие методы имеют префикс “DB\_”

#### Зависимости

Zero\_App, Zero\_Logs, Zero\_Helper, Zero\_Cache, Zero\_I18n, Zero\_Model

### Инженеринг (Zero\_Engine)

#### Назначение

Используемый класс Zero\_Engine

Реализует функционал автоматического создания и/или конфигурирования модели (класса) на основании таблицы хранящей ее объекты.

Производится конфигурирование самой модели (определения), ее свойств и связей.

Развернуто конфигурирует свойства. Создает конфигурацию для фильтров, табличного представления, представления в формах и работы с БД.

#### Реализация

На входе фабрика принимает группирующий префикс таблиц()ы.

Производится анализ соответствующей таблиц(ы) и на основе его результата конфигурируется модель (модели).

Шаблон для создания модели:

/zero/data/Template\_Model.php

Также формируется файл интернационализации из шаблона:

/zero/data/Template\_I18n.php

Также производит первичное создание модуля.

Создается файловая структура, главный раздел.

Дополнительную информацию и возможностях можно почерпнуть из самого компонента.

Реализованные формы:

Формы и типы свойств (определяемые при анализе):

Number - (text) Число

Text - (text) Строка текста

Select - (select) Перечисление

Radio - (radio) Перечисление с умолчанием (определяет по не нулевому значению поля Enum)

Checkbox - (checkbox) Множество

Textarea - (textarea) Текст

Date - (text) Дата

Time - (text) Время

DateTime - (text) Дата и время

Link - (select) Свойство связи с другим объектом

Формы и типы свойств (не определяемые при анализе):

Hidden - (hidden) Скрытое поле

ReadOnly - (-) Только для чтения

Password - (password) Строка пароль

File - (file) Ссылка на файл (пытается определить по имени поля)

FileB - (file) Бинарные данные файла (пытается определить по имени поля и типу)

Img - (file) Ссылка на картинку (пытается определить по имени поля)

ImgB - (file) Бинарные данные картинки (пытается определить по имени поля и типу)

Content - (textarea) Большей текст (визевик) (пытается определить по имени поля)

LinkMore - (select) Свойство связи с другим объектом (многочисленным - ajax)

#### Зависимости

Zero\_DB, Zero\_Logs, Zero\_Model, Zero\_App, Zero\_Helper, Zero\_I18n

### Фильтр (Zero\_Filter)

#### Назначение

Используемый класс Zero\_Filter

Формирование фильтров, условий поиска, сортировок и деления на страницы для получения определенного среза данных.

#### Реализация

Инициализируется по необходимости в конкретном контроллере.

Работает в тесной связке с моделью, для которой и формирует фильтр.

Формируется через статичный фабричный метод и сохраняется в сессии, принимая в качестве аргумента целевую модель. Формирование фильтра:

Zero\_Filter::Factory($Model);

Передается компоненту работы с БД, который централизованно формирует условия для последующего запроса к источнику.

Фильтр содержит в себе:

Фабричный метод формирования фильтра

Методы добавления фильтров (их настроек), установки их значений и их получение.

Методы добавления полей для поиска, установки их значений и их получение.

Методы добавления полей для сортировок, установки их значений и их получение.

Метод сброса фильтра в целом или для определенного свойства.

Конфигурация фильтра заложена в статичном методе модели, для которой он формируется.

Config\_Filter()

Вызывается конфигурация через динамический метод базовой модели.

Get\_Config\_Filter()

По умолчанию фильтры строятся для полей:

Перечисления (Select, Radio)

Дата и время (DateTime)

Поля связи (Link, LinkMore)

Множества (CheckBox)

По умолчанию поиск и сортировка формируется для всех остальных числовых и строковых полей (поля в фильтрах исключаются).

Эти настройки можно переопределять по необходимости.

*Важно:*

Персональные фильтры свойств реализуются в модели, к которой они относятся.

Имена таких методов должны начинаться с префикса “ FL\_”

#### Зависимости

Zero\_App, Zero\_Model, Zero\_Session

### Интернационализация (Zero\_I18n)

#### Назначение

Перевод контента на нужный язык. Интернационализация проекта.

Используемый класс Zero\_I18n

Специальной инициализации не требуется

Системно переводит:

Название контроллеров и их действий (методов)

Сообщения действий контроллеров.

Название моделей и их свойств.

Значение для полей Enum и Set

Сообщения ошибок валидации свойств.

#### Реализация

Переводы реализованы с помощью одномерных ассоциативных массивов находящихся в языковых файлах.

Эти массивы являются возвращаемым значением этих файлов.

Таким образом, поиск нужно перевода производится по языковому файлу и ключевой фразе (индексу).

Поиск файла перевода происходит аналогично поиску класса с моделью с той лишь разницей, что поиск ведется в папке “i18n” модуля, а не “class”

Info\_Article

1. /application/Info/i18n/Article.php
2. /zero/i18n/Article.php
3. /application/Info/i18n/App.php
4. /zero/i18n/App.php

Если указан файл Info\_App то поиск идет напрямую только этого файла:

1. /application/Info/i18n/App.php
2. /zero/i18n/App.php

Этот языковой файл (его имя) зарезервирован для перевода общих и ключевых элементов.

(Название проекта, Имя почтового адреса по умолчанию для всего приложения, …)

Это реализовано для удобства. Вы вправе сами решать, где и какой перевод хранить.

Если проект маленький можно весь перевод хранить в одном файле (Www\_App)

Далее в полученном массиве из файла производится поиск элемента с нужной ключевой фразой (индексом)

Если по ключевой фразе перевод не был найден, к ней добавляется приставка “translation ” и поиск повторяется. Пример перевода:

$translation = Zero\_I18n::T(‘Info\_Article’, ‘translation Head’, ‘Head’);

Файлы перевода формируются на основе моделей или функционала, для которых нужен перевод.

Info\_Article

Ключевые фразы формируются с использование в начале ключевого слова “translation ”

“translation Head”

Зарезервированные системные ключевые фразы:

controller ControllerName => Имя контроллера

controller ControllerName action ActionName => Название действия

controller ControllerName message Key => Сообщение на действие

model => Имя модели

model prop PropName => Название свойства модели

model prop PropName validate Key => Сообщение валидации свойства

model prop PropName option Key => Варианты значений для свойств ENUM, SET

Для удобства переводов внутри html шаблонов представления реализована директива translation.

#### Реализация переводов в БД

В системе используется асинхронный подход к реализации переводов.

Структура в таблице:

News(ID, Language\_ID, Name, …)

В том числе и для требующих синхронного подхода.

В БД для таких объектов создаются отдельные языковые таблицы (синхронный перевод):

News(ID, Name), NewsLanguage(ID, News\_ID, Language\_ID, Name)

Но на уровне приложения подход также остается асинхронным. А именно:

Создаются два отдельных объекта $News и $NewsLanguage.

С которыми в дальнейшем можно работать как с единым целым, так и по отдельности.

$News использовать для функционала (какие либо вычисления) требующего точной идентификации объекта как целого.

$NewsLanguage использовать в информационных целях и дальнейших переводах как отдельный объект конкретного языка.

Такой подход упрощает реализацию и понимание приложения, а также добавляет ей гибкости.

#### Зависимости

Zero\_App, Zero\_Logs

### Логирование (Zero\_Logs)

#### Назначение и Описание

Используемый класс Zero\_Logs

Реализует функционал профилированный мониторинг и снятие статистических данных о работе приложения в частности и в целом. Профилирование работы приложения.

#### Реализация

Инициализация компонента происходит во время инициализации приложения.

Фиксирует затраченное время на выполнение приложение в целом и по его составным частям.

Затраченной оперативной памяти.

Фиксирует критические ошибки в работе приложения и запросов к БД.

Фиксирует предупреждения.

Действия пользователя и их результат.

Фиксирует работу фоновых задач приложения.

Выводит отладочную информацию в браузер.

Формирование статистики происходит при завершении работы приложения (происходит перехват).

Является обработчиком исключений и ошибок программирования

Обрабатывает в исключениях ошибки 403, 404, 500.

Использование таймеров

Zero\_Logs::Start('#{APP.Full}');

…

Zero\_Logs::Stop('#{APP.Full}');

Использование сообщений (ошибки, предупреждения, отладочная информация)

Zero\_Logs::Set\_Message('#{ERROR\_LOAD\_PROP} load prop {$prop}, 'warning');

Использование исключений

throw new Exception('metod not found: ' . get\_class($this) . ' -> ' . $method);

throw new Exception('Not Found', 404);

#### Зависимости

Zero\_App, Zero\_View, Zero\_Session

### Модель (Zero\_Model)

#### Назначение

Используемый класс Zero\_Model.

Реализует основную работу с моделями. Являясь базовым абстрактным классом для каждой из них.

Конкретные модели реализуют в себе частные методы (уникальные, сложные):

Валидаторы (VL\_PropName)

Фильтры (FL\_PropName)

Запросы к БД (DB\_QueryName)

#### Реализация

Модель создается и инициализируется через фабрику и фабричный метод.

1. Создание с сохранением в сессию (Factory). И вызовом фабричного метода (Init)
2. Создание с сохранением в пределах процесса (Instance) . И вызовом фабричного метода (Init)
3. Создание (Make)

Фабрика производит поиск и создание модели соответствующей переданному источнику.

Поиск ведется в конфигурации приложения. Если соответствие не было найдено создается модель одноименная источнику.

Данный алгоритм позволяет переопределять и наследовать модель не изменяя базовую, причем для каждого приложения в отдельности.

Реализует абстрактную работу со свойствами моделей через магические методы и их конфигурации.

Реализует механизм динамического сохранения моделей в сессии

Работа со свойствами:

Реализован механизм хранения состояния свойств модели во время работы.

Что позволяет оптимизировать работу с БД. Производить обращение к ней, только если свойства действительно были изменены. Соответственно есть в связи с этим определенные правила работы:

Изменение свойств производится как обычно:

$News->Description = $value;

Изменение свойств со стороны источника (то есть загрузка) производится через метод Set\_Props.

$News->Set\_Props($properties)

Реализован также метод для получения свойств с определенным статусом.

$News->Get\_Props($flag)

$flag = -1 получить только измененные свойства и их значения

$flag = 0 получить все свойства (по умолчанию) и их значения

$flag = 1 получить только не измененные свойства и их значения

Реализован механизм независимых конфигураций свойств под разные задачи. Как то:

фильтры, гриды, формы, БД

Использование необходимых компонентов через модель:

Используется алгоритм ленивой загрузки компонентов.

Компоненты инициализируются при обращении к специализированным свойствам, сохраняющих их.

В момент обращение к компоненту через свойство ему делегируется текущая модель.

И далее с ним производится необходимая работа.

Компоненты с которыми можно работать через модель:

Zero\_DB Работа с источником хранящим данные моделей

Zero\_Validator Проверка свойств модели

Zero\_Cache Кеширование данных

Примеры использования:

$Model->DB->Select(“\*”);

$Model->VL->Validate($\_REQUEST[‘data’]);

$Model->FL->Add\_Filter\_Select($prop, $comment, $is\_visible, $load, $source\_name);

Альтернатива и синонимы:

$Model->DB->…

$Model->Get\_DB()->…

$Model->VL->…

$Model->Get\_VL()->…

$Model->Cache->…

$Model->Get\_Cache()->…

*Примечание:*

Используется паттерн композиция с делегированием при работе с компонентами.

#### Зависимости

Zero\_App, Zero\_Session, Zero\_I18n, Zero\_DB, Zero\_Validator, Zero\_Cache, Zero\_Logs

### Сессия (Zero\_Session)

#### Назначение

Используемый класс Zero\_Session.

Наследуется от библиотечного класса ArrayObject

Централизованное сохранение необходимых данных (прежде всего объектов) в сессию.

Для доступа к ним между процессами (запросами).

#### Алгоритм работы

Инициализация компонента происходит во время инициализации приложения

Создает единственный экземпляр (объект) самого себя внутри своего статичного свойства.

И в дальнейшем вся работа идет через него.

Таким образом, реализует сохранение любых данных в виде индексированного массива внутри себя.

#### Зависимости

Zero\_Logs

### Валидатор (Zero\_Validator)

#### Назначение

Используемый класс Zero\_Validator

Проверка данных пришедших извне на правильность заполнения. Формализация их до нужного вида.

#### Реализация

Инициализация компонента происходит при обращении (использовании) через свойство конкретной модели.

Проверка может быть как по форме представления свойства, так и по его названию. Возможно использование сценариев проверки. Реализует механизм сообщений с детализацией по каждому проверяемому свойству.  
Методы проверки свойств (валидаторы) могут быть расширены моделью, свойства которой проверяются.

Работает только через модель (его свойство).

Запуск и выполнение валидации (проверки):

$Model->Validator->Validate($\_REQUEST['Prop']);

Порядок поиска:

1. Поиск по имени свойства в самой модели. В качестве аргументов таким методам передается проверяемое значение и сценарий.
2. Поиск по форме представления в валидаторе. В качестве аргументов таким методам передается проверяемое значение и целевое свойство.

Если ничего не найдено значение присваивается свойству, безусловно.

Непрошедшие проверку свойства и их причины заносятся в массив:

$Validator->Errors;

И доступны через метод

$Validator->Get\_Errors();

Конфигурация для первичной валидации и поиска валидаторов по форме представления заложена в статичном методе модели.

Config\_Form()

Вызывается конфигурация через динамический метод базовой модели.

Get\_Config\_Form()

*Важно:*

Персональные валидаторы свойств реализуются в модели свойства которой они валидируют.

Имена таких методов должны начинаться с префикса “VL\_”

*Примечание:*

Также реализован поиск начального и завершающего метода валидации, с передачей им всего массива пришедших данных.

$Validator->Validate\_Before ($data, $scenario);

$Validator->Validate\_After ($data, $scenario);

#### Зависимости

Zero\_Model, Zero\_Helper, Zero\_Logs, Zero\_I18n

### Представление (Zero\_View)

#### Назначение

Используемый класс Zero\_View

Представление решает задачу разделения представления информации от ее формирования.

Позволяет разделить работу программиста и верстальщика.

Тем самым ускоряет процесс верстки шаблонов и интерфейсов представления информации.

А также позволяет программисту сконцентрироваться на бизнес логике приложения, не отвлекаясь на верстку. Благодаря параллельной работе ускоряется процесс разработки приложения.

Представление реализовано шаблонизатором.

Специальной инициализации компонент не требует.

#### Файловая структура

/application

/modname

/view

/TplFolderName

TemplateName1.html

TemplateName2.html

TemplateName3.html

Layout1.html

Layout2.html

Layout3.html

/phpzero

… аналагогично /application

#### Реализация

Синтаксис шаблонизатора:

##### Комментарии

Различные технические комментарии. Удаляться при компиляции шаблона.

{# comment\_name #}

##### Переменные

Вывод значение переменной ( к примеру название товара )

{$name}

Присвоение нового значения переменной без вывода ( к примеру счетчик )

{set $parice + 5}

##### Функции

Перевод каретки (\n) меняет на <br>\n

{nl2br($name)}

Перевод html символов в символы их дословного отображения

{htmlspecialchars($name)}

Вывод структуры и данных указанной переменной

{pre($name)}

Вывод списка всех переданных переменных (данных) в шаблон

{pre($\_)}

Вывод всех переданных переменных (данных) в шаблон (их значений)

{pre($\_\_)}

*Примечание:*

Собственно можно использовать любую встроенную или пользовательскую php функцию

##### Логика

Вывод, обработка содержимого блока при истинности условия в выражении

{if ( $count > 20 )} ... {/if}

Вывод, обработка содержимого одного из 2 блоков в зависимости от истинности условия в выражении

{if ( $count > 20 )} ... {else} ... {/if}

Вывод, обработка содержимого одного из блоков в зависимости от истинности стоящего перед ним условия в выражении

{if ( $count > 20 )} ... {else if ( $count > 10 )} ... {else} ... {/if}

Логический переключатель. Целесообразно использовать, когда вариантов ветвления более 5

{switch ( $expression )}

{case $value:} <span>popcorn</span> {break}{/case}

{case $value:} <span>octopus</span> {break}{/case}

{default :} <span>octopus</span> {/default}

{/switch}

*Важно:*

Первый case пишется сразу же после указания switch: {switch ( $expression )}{case $value:} …

##### Циклы

Формирование списка, таблицы из массива элементов ( к примеру товары, группы, статус чего либо ) если он не пустой ( хотя бы один элемент )

{foreach ($wares as $id=>$ware)}

... <a href="index.php?id={$id}">{$ware['name']}</a> ... {$ware['description']} ...

Или

... <a href="index.php?id={$id}">{$ware}</a> ...

{/foreach}

{foreach ($status\_list as $value)}

... {$value}<br> ... Или ... {$value[‘status’]}<br> ...

{/foreach }

{for ($i=0; $i < count($wares); $i++)}

номер {$i} <a href="index.php?id={$wares[$i]['id']}">{$wares[$i]['name']}</a>

{/ for }

Выполнение блока при соблюдении условия в выражении

{while ( $count > $count\_iteration )}

... {$name} ...

{set $count\_iteration+=1}

{/while }

*Примечание:*

{/if}, {/foreach }, {/for}, {/while}, {/switch}, {/case}, {/default}

Вместо них можно использовать один для всех элементов {/}

Для управления ходом выполнения циклов и переключателя есть возможность использовать следующие конструкции: {break}, {continue}

##### Директивы

**Перевод:** {**translation** “Zero\_Users” ”key”} или {**lang** “Zero\_Users” ”key”}

Перевод ключевого слова на текущий язык. Производится во время компиляции шаблона.

На выходе формируется языковой шаблон с нужным переводом.

Алгоритм поиска и перевода описан в компоненте Zero\_I18n

**Плагин:** {**plugin** ”plugin\_name” [property\_name1=value1 property\_name2=value2]}

Подключение и выполнение плагина в шаблоне.

*plugin\_name* – имя плагина в формате имени контроллера

*property\_name1, property\_name2* – имена свойств плагина

*value1, value2* – соответствующие значения для свойств. Значение может быть любое.

Алгоритм поиска плагина:

“Zero\_Helper\_Pager”

application/zero/class/Helper/Pager.php

phpzero/zero/class/Helper/Pager.php

“Zero\_Section\_NavigationAccordion”

application/zero/class/Section/NavigationAccordion.php

phpzero/zero/class/Section/NavigationAccordion.php

**Включение:** {**include** “Zero/Include/Headermeta”} или {**inc** “Zero/Include/Headermeta”}

Включение шаблонов *path\_to\_filename* в текущий шаблон как есть без компиляции.

Алгоритм поиска шаблона аналогичен поиску всех остальных шаблонов:

“ Zero/Include/Headermeta”

application/zero/view/Include/Headermeta.html

phpzero/zero/view/Include/Headermeta.html

##### Пути

{HTTP} Ссылка на главную страницу ( http://www.domain.com)

{HTTPH} Полная ссылка на страницу с которой пришли ( http://www.domain.com/page/page/.... )

{HTTPA} http ссылка на статичные данные (images, css, js) ( http://www.domain.com/assets)

{HTTPD} http ссылка на бинарные данные объектов и пользователя (http://www.domain.com/upload/data)

{URL} относительная ссылка на раздел в системе ( /page/page/page ) с учетом языка

##### Примеры

Пример абстрактного использования:

<input type="text" id="{$prop}" name="Prop[{$prop}]" value="{htmlspecialchars($\_REQUEST['Prop'][$prop])}">

<input type="text" id="{$prop}" name="Prop[{$prop}]" value="{htmlspecialchars($Client->$prop)}">

Пример элемента формы:

<input type="text" id="Name" name="Prop[Name]" value="{htmlspecialchars($\_REQUEST ['Prop']['Name'])}">

<input type="text" id="Name" name="Prop[Name]" value="{htmlspecialchars($Client->Name)}">

Правило формирование элемента формы:

name="Prop[Name]"

Служит для централизованной обработки формы системой.

Prop – группирующий индекс для свойств конкретного объекта.

Name – Имя (свойства, поля, переменной).

value="{htmlspecialchars($\_REQUEST ['Prop']['Name'])}"

Содержит ранее введенные данные в текущее поле. (Для добавления новых элементов – регистрация клиента, новое сообщение…)

value="{htmlspecialchars($Client->Name)}"

Содержит данные редактируемого объекта. (Для изменения существующих элементов – профиль клиента, сделанные заказы…)

Там где это необходимо (в контроллерах) создается объект представления.

При создании в конструктор передается конкретный шаблон.

$View = new Zero\_View(‘template\_name’);

Шаблоны можно добавить позднее в стек.

$View->Template\_Add(‘template\_name’);

Или удалить из стека.

$View->Template\_Rem(‘template\_name’);

Далее в него передаются данные для вывода.

$View->Assign('variable', 'value');

После чего происходит вызов на выполнение шаблона и получение готового результата.

$View->Fetch();

*Примечание:*

$View->Fetch(); вызывается системно самим приложением. Без особой необходимости этого делать не надо.

Все бинарные данные хранятся в каталоге /upload.

#### Зависимости

Zero\_App, Zero\_Logs, Zero\_Helper

# Справочник (Reference)

## Приложение

Работа программного комплекса в целом над конкретным проектом. А также компонент Zero\_App в частности, который осуществляет управление его работой.

## Система, ядро

Совокупность программного обеспечения необходимая для работоспособности программного комплекса в целом (/zero).

## Модуль

Совокупность программного обеспечения (контроллеры, модели, представления …) решающие какую либо конкретную прикладную задачу.

## Плагин

Это контроллер, не содержащий и не выполняющий в себе никакие методы действия.

## Хелпер

Набор однотипных и общих для большинства задач вспомогательных функций, объединенных в класс.

Все методы такого класса статичные. Вызываются по мере необходимости решения конкретных задач.

## Компонент, подсистема

Это логически законченный системный функционал решающий определенный круг связанных задач.

Компоненты реализованы классами.

## Макет (layout)

Это главный шаблон представления, в который вставляются все сформированные ранее данные.

## Представления

Это частные шаблоны конкретного контроллера либо шаблоны макетов всей страницы.

## Чанки (Chunk)

Это логически и функционально завершенные блоки кода контроллеров заключенные в отдельные методы.

(Чанк инициализации, чанк формирования вывода, чанк выполнения действий к.)

## Двоечник(и)

Это таблица связей многие ко многим. Или кросс таблица

## Свойство

Применяется в широком смысле слова: свойство объекта, полей в таблице БД, поле в форме представления.

## Компиляция шаблона

Анализ и замена в html шаблонах php псевдо кода на синтаксически правильный php код.

## Источник

Под источником понимается абстрактное хранилище конкретных данных.

Это может быть таблица в реляционной БД, или коллекция в Mongo DB, просто файл.

На данный момент это только MySQL

## Группа

Под группой в данной системе понимается не только объединение определенных пользователей.

Но также их права, как вертикальные, так и горизонтальные.

## УП

Условия пользователя накладываемые в процессе работы контроллеров.

Реализуют горизонтальные права доступа к рабочей информации.

## УС, УРС, УКС

Условие связи. Определяет понятие связи одного объекта с другим и его тип.

УС - условие связи один ко многим и общий термин обозначающий связь (зависит от контекста)

УРС – условие рекурсивной связи или связи на себя. Каталогизированные объекты.

УКС – условие связи многие ко многим

# API

Запросы могут иметь вид:

[GET|PUT|POST|DELETE|OPTIONS] : http://domain.com/api/v1/page/page/page?param=value& param1=value1…

[GET|PUT|POST|DELETE|OPTIONS] : http://api.domain.com/v1/page/page/page? param=value& param1=value1…

Данные запроса передаются в формате json

Ответ приходит также в формате json и иметт следующий вид

{

"Code": 0, код сообщения (0 любой правильный ответ, -1 любой не правильный ответ)

"Message": "", текст сообщения (либо параметры get запроса для options)

"ErrorStatus": false, признак ошибки

"Token": “”,, токен сессии

"Content": … Тело ответа

}

## "Token"

Параметр "Token" обязателен для авторизованных разделов

Он может передаваться:

Через заголовок ‘HTTP\_X\_ACCESS\_TOKEN’.

Через параметр ‘Token’ GET или POST запроса.

Через куку браузера (по умолчанию)

## OPTIONS

Метод описывает реализованные методы запроса.

Запрос без параметров возвращает реализованные методы (в поле Message)

При запросе конкретного метода возвращается информация о параметрах этого метода запроса.

Параметры POST и PUT запросов описываются в поле Content

Параметры GET запросов описываются в поле Message

(одиночный знак ?, означает что get параметров нет)