# Инструкция

* В разделе с инструкцией в папке "default" находятся некоторые стандартные файлы (на момент написания инструкции). После распаковки архива с фреймворком, можно просто заменить некоторые файлы, файлами из папки "default".

По ходу инструкции будут описаны изменения, которые сделаны в данных файлах.

* Скачать актуальную версию фреймворка с официального сайта: <https://doit-cms.ru/>
* Распаковать файлы из архива в корень сайта
* Создать базу данных для данного сайта, указав "character set = utf8 -- UTF-8 Unicode", "collation: utf8\_general\_ci"
* В файле config.php прописать данный для подключения к БД, а также указать следующие строчки:

$\_ENV['EMAIL\_FROM\_ADDRESS'] = 'noreply@mailer.' . $\_SERVER['HTTP\_HOST'];

$\_ENV['EMAIL\_FROM\_NAME'] = 'Название сайта/компании';

$\_ENV['SITE\_MAIN\_DOMAIN'] = $\_SERVER['HTTP\_HOST'];

* В файле app/admin.ini указываются доступы в админку (под логинами admin и developer).

Пароль указывается в виде md5 значения.

По умолчанию пароль от обоих учеток: 1

Также внизу указываются пункты, которые будут доступны под админом сайта в меню "Редактировать".

Можно указывать чисто название таблицы БД и название для данного пункта меню:

pages "Текстовые страницы"

в этом случае ссылка будет вести на интерфейс админа, в котором можно управлять содержимом указанной таблицы БД.

Можно просто указывать произвольные ссылки:

/admin/test\_url "Тест"

Ссылки типа "/admin/..." доступы только админу сайта.

* Вот минимальные требуемые ссылки в файле app/admin.ini:

1. pages "Текстовые страницы" - содержимое таблицы pages
2. /admin/edit/options/all "Опции" - строка из таблицы options, в которой указываем все основные данные для сайта (email для писем, телефон в шапке, текст в подвале, ссылки на соц.сети и т.д.)
3. seoparams "Страницы SEO" - данная таблица используется для задания SEO данных страниц
4. /admin/edit/robots/all "Robots.txt" - строка из таблицы robots, дает возможность через админку сайта задавать содержимое, выводимое по ссылке: http://site/robots.txt - требуется сеошнику

В разделе с инструкцией в папке "default" можно найти данный файл со стандартным паролем и необходимыми ссылками.

* Минимальными необходимыми таблицами в БД являются:

pages

seoparams (с обязательным полем: page\_url)

options

robots

Все таблицы в БД создавать только через учетку developer в меню Скаффолдинг - Создать

Там просто ввести название новой таблицы и нажать "Создать".

Файл dump.sql с указанными таблицами присутствует в папку "default"

* Все названия таблицы БД должны быть в виде английского слова маленькими буквами во множественном числе
* Нюансы по названиям полей:

Для дат используются названия по типу: \*\_at - к примеру date\_at. Дата хранится в sql формате: 2015-12-19 16:33:35

Для полей типа checkbox используется название: is\_\* - к примеру is\_mainmenu. По умолчанию в этом поле будет хранится 0 у всех, при установленной галочке 1. В php можно проверять if (d()->this->is\_mainmenu) { ... }

Для связей между таблицами используются по типу \*\_id - к примеру catalog\_id (id строчки из таблицы catalogs)

* Для того чтобы авторизоваться под администратором надо перейти по ссылки вида:

http://site/admin

Ввести логин: developer

И пароль (по умолчанию: 1).

* Все основные файлы функционала находятся в папке app/
* Файлы стилей, картинок верстки, скриптов размещаются в images/
* Файл app/main.html - это основной шаблон сайта. Там размещается код шапки, подвала.

Часть, которая меняется в зависимости от страницы, в этом файле обозначается как {{{content}}}

Вместо этого будет подставляться код из других файлов.

В данном файле необходимо провести следующие действия:

Строчку <meta charset="UTF-8"> поднять выше, чтобы она шла до вывода тега <title>

В теге <title> добавить вывод заголовка из Seo (специальный класс, работает с таблицей seoparams), а также добавить необходимые мета-теги:

<title>{Seo.title}</title>

<meta name="description" content="{Seo.description}" />

<meta name="keywords" content="{Seo.keywords}" />

<meta name="viewport" content="width=device-width,minimum-scale=1,maximum-scale=1,initial-scale=1,user-scalable=no">

{Option.meta}

К подключению style.css прописать гет параметр, основанный на времени последнего изменения файла, чтобы не возникала проблема, когда в файл внесены изменения, а у пользователей отображается закэшированная у них версия данного файла без новых изменение.

Перед </body> вставить:

{Option.metric}

Стандартный файл с данным изменениями можно взять в папке "default"

Также необходимо заменить подключаемую версию jquery на актуальную (в файле это еще не сделано).

* В файле .htaccess необходимо убрать подстроку "|^robots\.txt"

Файл с данным изменением можно взять в папке "default"

* В папке app/fields находятся файлы, в которых описываются, какие поля (и их типы), даются для заполнения при добавлении новых записей в соответствующую таблицу БД

Данные файлы можно редактировать как обычные текстовые файлы из данной папки, либо в момент, когда на сайте перешли на страницу добавления или редактирования элемента из таблицы (в режиме developer), то в меню будет доступен пункт: Скаффолдинг - Текущий шаблон

Лучше пользоваться вариантом редактирования через сайт.

Если для текущей таблицы БД шаблон еще не создавался, то в данном интерфейсе будет предложено: "Шаблон пустой. Использовать заготовку?". Нажимать "Да".

Теперь можно указывать поля, которые вам необходимы.

Поля задаются по шаблону:

тип\_поля название\_поля\_в\_таблице "Название\_на\_русском"

Иногда требуется 4й параметр

Есть следующие основные типы полей:

small - обычное текстовое поле: <input type = "text">

rich - большое текстовое поле, с возможностями выделять жирным, курсивом, вставлять картинки и т.д. Каждый абзац в данном поле автоматически заключается в тег <p>

rich\_non\_p - аналогично типу rich, но абзацы не заключаются в тег <p>

big - обычное большое текстовое поле: <textarea>

date - текстовое поле, с привязанным к нему календарем, чтобы можно было удобно выбирать дату. Название полю давать по типу \*\_at

image - возможность указать ссылку на картинку, либо загрузить ее с компьютера

file - возможность указать ссылка на файл, либо загрузить его с компьютера

select - поле типа выпадающий список. Возможные варианты задаются после названия поля на русском в двойных кавычках, указывая перед закрывающей кавычкой в скобках число, которое будет являться сохраняемым значением, пример:

select razdel "Раздел" "Оборудование(1)" "Обучение(2)"

checkbox - поле типа галочка. Названия поля давать по типу is\_\* - пример

checkbox is\_active "Активный?"

select\_table - поле типа выпадающий список, в качестве значение выводятся поля из таблицы, указанной 4ым параметром, к примеру:

select\_table catalog\_id "Каталог" catalogs

legend - создает в форме подзаголовок, для визуального выделения части полей. Для корректного вывода русского заголовка требуется в конце заголовка поставить пробел, пример:

legend "Подвал "

container\_begin - используется в паре с container\_end. Используется, что показывать некоторые поля к заполнению, только в случае выполнения условия, к примеру если конкретная галочку стоит или не стоит, пример:

checkbox is\_show\_dops "Показать доп поля"

container\_begin is\_show\_dops 1

small dop1 "Доп поле"

container\_end

**Дополнительные типы:**

select\_table\_null - поле типа выпадающий список, в качестве значение выводятся поля из таблицы, указанной 4ым параметром. Есть возможность указать вариант "Нет", в случае чего будет сохранено значение "0" к примеру:

select\_table\_null page\_id "Родительская страницы" pages

Чтобы использовать данный тип, необходимо в папку app загрузить файл "admin\_select\_table\_null.html", который есть в папке "default"

onchange\_bl - подключает файл, который считает кол-во введенных символов у пользовательских seo-полей, описанных ниже.

Чтобы использовать данный тип, необходимо в папку app загрузить файл "admin\_small\_seo.html", который есть в папке "default"

small\_seo - обычное текстовое поле, с поддержкой подсчета символов onchange\_bl

Чтобы использовать данный тип, необходимо в папку app загрузить файл "admin\_onchange\_bl.html", который есть в папке "default"

big\_3 - большое текстовое поле с параметром rows="3", с поддержкой подсчета символов onchange\_bl

Чтобы использовать данный тип, необходимо в папку app загрузить файл "admin\_big\_3.html", который есть в папке "default"

big\_7 - большое текстовое поле с параметром rows="7", с поддержкой подсчета символов onchange\_bl

Чтобы использовать данный тип, необходимо в папку app загрузить файл "admin\_big\_7.html", который есть в папке "default"

* Стандартные файлы шаблонов (в папке app/fields/) для указанных обязательных таблиц предоставлены в папке "default"

pages.ini

seoparams.ini

robots.ini

options.ini

* Надо загрузить файл app/admin\_panel.php, предоставленный в папке "default". Это пользовательская замена отображения в режиме администратора верхней меню-панели. В ней дополнительно выводится пункт "Параметры страницы", для удобного заполнения seo-параметров. Если находясь на нужной странице сайта нажать эту кнопку, то появится интерфейс добавления/редактирования seo-параметров именно для данной страницы. Сделано для удобства, чтобы не искать в большом списке всех записей.
* seoparams

Страница Редактировать - Страницы SEO

При добавлении элементов в данную таблицу в поле "URL страницы" необходимо заносить адрес требуемой страницы с ведущим слешем, примеры:

/index

/news/news1

* Для того, чтобы работал вывод robots.txt через таблицу БД, необходимо еще

Залить файл app/robots.func.php, предоставленный в папке "default".

В app/router.func.php необходимо добавить строчку

route('/robots.txt', 'main', 'show\_robots');

Файл предоставлен в папке "default"

* app/router.func.php

Кратко о работе данного файла.

В нем прописываются правила - какие url'ы какой функцией должны обрабатываться.

Пример:

route('/page\_uniq', 'pages#page\_uniq');

Данная строчка сообщает сайту, что на странице http://site/page\_uniq в app/main.html вместо {{{content}}} будет подставлено содержимое метода page\_uniq(), который описан в файле app/mod\_pages/pages.func.php

Данная строчка, является полным аналогом записи вида:

route('/page\_uniq', 'content', 'pages#page\_uniq');

В которой явно указывается, что подставлять содержимое 'pages#page\_uniq' надо именно вместо 'content'.

Предпочтительнее использовать первый вариант, для минимизации кода.

Явное указание, что именно заменять, позволяет написать такое правило:

route('/original\_page', 'main', 'pages#original\_page');

В этом случае, содержимое новой функции будет подставляться вместо всего вывода app/main.html

Т.е. таким образом можно удобно создать уникальную страницу, с другим дизайном, подключив другие файлы стилей и т.д. Это можно было бы добиться указав в основном файле app/main.html кучу условия, когда какой блок выводить, какие стили подключать - но это менее удобно и затратнее по времени выйдет.

Обычно переопределение выводимой функции 'main' используется чтобы выводить всплывающие формы.

Т.е. в коде где-нибудь указывается через использование fancybox ссылка:

<a href = "/ajax/fosv"....

А в роутере указывается правило:

route('/ajax/fosv', 'main', 'pages#fosv');

**Нюанс**

Запись вида:

route('/catalogs/', 'catalogs#show');

Т.е. где определяемый url заканчивается слешем, позволяет определить функцию таким образом:

function show($url) {

d()->this = d()->Catalog->find\_by\_url($url);

…

Т.е. теперь, при открытии ссылок вида:

/catalogs/catalog1

/catalogs/computers

Будет вызываться данная функция, с передачей в качестве параметра значения, идущего после /catalogs/, т.е. для примеров выше:

$url = 'catalog1' и $url = 'computers'

Что позволяет получить из таблицы БД нужную для вывода строку (работа с таблицами БД будет описана ниже).

* После того как определили функцию для вывода контента, к примеру 'pages#page\_uniq' - внутри класса, описанного в данном файле (т.е. app/mod\_pages/pages.func.php) надо объявить функцию с указанным именем, в ней прописать необходимый вам код. После выполнения указанного кода, если не было прописано команды

exit;

будет произведен вывод содержимого файла

app/mod\_pages/\_page\_uniq.html

Т.е. файл вывода определяется как html-файл из текущей папки, с название соответствующим названию функции, но со знаком "\_" вначале.

Иногда файл автоматически не выводится (вероятно зависит от некоторых настроек сервера). В этом случае в конце вашей функции необходимо указать строчку:

print d()->view();

Если функция в php-файле не будет описана, то просто сразу будет произведен вывод html-файла.

Также можно специально указать для вывода файле с именем отличным от названия функции, если это необходимо, пример:

print d()->pages\_page\_uniq2\_tpl();

Здесь вначале указывается контроллер (папка) где искать файл: "pages" говори, что искать в app/mod\_pages

Затем идет название файла: "page\_uniq2"

И обязательная приписка: "\_tpl"

* Главную страницу сайта принято выводить из функции pages#index

Для этого в файл app/mod\_pages/pages.func.php просто надо продублировать полностью функцию show() и переименовать ее в index().

Ну и создать в данной папке файл \_index.html

Измененный php-файл и html-файл предоставлены в папке "default"

* Все остальные страницы, если для них не описано правил в роутере, обрабатываются функцией pages#show
* **Работа с таблицами БД**
* Добавлять новые данные в таблицу БД можно либо перейдя на страницу с интерфейсом по данной таблице, создав соответствующий пункт в app/admin.ini

Либо можно в любом html-файле в необходимом месте указать конструкцию типа:

{{add 'pages'}}

После чего на странице в указанном месте для авторизованного админа сайта появится иконка 

Нажав на нее откроется страница добавления элемента в указанную таблицу.

Можно задать значения, установленные по умолчанию, к примеру:

{{add 'pages', 'is\_mainmenu' => 1}}

В этом случае, если в шаблоне описана галочка с именем is\_mainmenu, то при нажатии на такой плюс, галочка автоматически будет установлена. Таким образом можно указать сколько угодно параметров, просто аналогично дописывать их через запятую

{{add 'pages', 'is\_mainmenu' => 1, 'tip\_str' => 'Главная'}}

* Во всех шаблонах, которые относятся к данным, у которых должна быть возможность вывода на отдельной странице (текстовая страница, каталог, новость, товар, статья и т.д.) обязательно должно быть поле для заполнения:

small url "Адрес страницы (необязательно)"

Данное поле нужно для ЧПУ. Можно его заполнять вручную (обязательно соблюдать уникальность значения для всей таблицы требуется). Либо можно при добавлении элемента оставить это поле пустым, в этом случае оно автоматически заполнится значением: имя\_таблицы\_в\_единственном\_числе + id строки, т.е.: page4

По данному полю будет искаться строчка из таблицы, необходимая для вывода.

Также желательно у всех шаблонов задавать поле:

small title "Название"

Т.к. в различных интерфейсах админки строки выводятся именно данным значением.

* Можно вывести ссылку на сортировку элементов:

{{sort\_icon 'pages'}}

На сайте появится иконка: 

При нажатии на нее откроется интерфейс где будут выведены элементы таблицы (выводится значение поля title), и их можно менять местами прямо перетаскивая их мышкой.

Также можно выводить ссылку с дополнительными параметрами

{{sort\_icon 'pages', 'is\_mainmenu' => 1}}

В этом случае в интерфейсе сортировки будут выведены только те элементы таблицы, у которых в поле 'is\_mainmenu' сохранено значение 1

* Чтобы реализовать на сайте вывод значений из какой-нибудь таблицы, допустим products, необходимо обратиться к ней через d()->, написав ее с большой буквы и в единственном числе (есть некоторые исключения из-за правил английского языка, к примеру, для таблицы новостей news единственное число все равно будет с s на конце: News),к примеру:

d()->Page

d()->Product

d()->Catalog

**Пример работы:**

$products = d()->Product;

После этого в переменной $products будет находится объект со списком значений из таблицы.

**Нюанс**

Если таким образом определить переменную в функции в php-файле, то при выводе шаблона (верстки), т.е. html-файла данная переменная будет не определенной.

Чтобы в html-файле иметь доступ к переменным, определенным в соответствующей функции в php-файле, необходимо именовать переменные через d()->

Т.е. в нашем случае:

d()->products = d()->Product;

Теперь у нас и в php и html файлах будет доступна переменная d()->products

Для вывода списка элементов используется цикл foreach, т.е. в html-файле пишется:

<?php

foreach (d()->products as $v) {

print '<div><a href = "/products/' . $v->url . '">' . $v->title . '</a></div>';

}

?>

Однако данный вид записи не очень удобен, поэтому в html-файле можно реализовать аналогичный код таким образом:

<foreach products>

<div>

<a href = "/products/{.url}">{.title}</a>

</div>

</foreach>

Разберем более подробно.

При таком вызове foreach не надо указывать перед переменной начальное d()->

Первая строка является аналогом строки:

<?php foreach (d()->products as d()->this) { ?>

Т.е. внутри цикла у нас будет определена переменная d()->this, в которой находится текущая строка из таблицы БД.

Запись вида: {.url} является аналогом записи:

<?php print d()->this->url; ?>

Также если была определена переменная, к примеру:

d()->big\_title = 'Крутой заголовок';

То в html-ке ее можно вывести как

{big\_title}

Что будет являться аналогом:

<?php print d()->big\_title; ?>

Обязательно учитывайте, что {.title} и {title} - это разные команды, и использовать надо именно необходимую.

Если переменная d()->this была уже как-то определена до вызова <foreach ...>, то внутри цикла она будет недоступна, т.к. в d()->this будет находится уже строка переменной цикла. Однако после завершения цикла </foreach> в переменной d()->this опять будет храниться ваше значение.

Внутри такого цикла можно у элемента выводить ссылка на его редактирование и удаление:

<a href = "/products/{.url}">{.title}</a>{{edit}}{{delete}}

Учитывайте, что т.к. они являются ссылками при выводе, их нельзя помещать в тег <a>

Такой вызов этих иконок создает ссылка на редактирование/удаление строки находящейся именно в переменной d()->this (на текущий момент).

Также можно не определять заранее переменную для вывода цикла, а писать сразу:

<foreach Product>

После определения (или во время) переменной как объекта таблицы БД можно использовать дополнительные команды, по синтаксису mysql:

d()->products = d()->Product->select('id, title, url, catalog\_id');

d()->products->order\_by('id ASC');

d()->products->group\_by('catalog\_id');

d()->products->limit(100);

Для того, чтобы фильтровать результаты запроса, используется команда ->where внутри которой пишутся условия. Однако вместо значений для проверки условий ставится знак '?', а после описания условий, вне кавычек, через запятую указываются значения, которые должны быть подставлены вместо вопросов, с учетом порядка.

Для условий IN, NOT IN должен передаваться массив:

$catalog\_id = 5;

$brand\_ids = array(1, 3, 6);

d()->products->where('catalog\_id = ? AND brand\_id IN (?)', $catalog\_id, $brand\_ids);

Не обязательно писать каждую команду на отдельной строке. Можно сделать все в одной строке прямо при вызове, или после нее:

d()->products = d()->Product->select('id, title, url, catalog\_id')->where('catalog\_id = ? AND brand\_id IN (?)', $catalog\_id, $brand\_ids)->order\_by('id ASC');

Повторный вызовы команд select, order\_by, group\_by, limit будут переопределять предыдущие вызовы.

Повторный вызов команды where будет добавлять к условиям еще, связывая их через AND

d()->products->where('catalog\_id = ?', $catalog\_id);

d()->products->where('brand\_id IN (?)', $brand\_ids);

По умолчанию, если не указывать, сортировка идет по возрастанию поля sort, в котором при добавлении элементов указывается значение совпадающее с id. Это же поле меняется при использовании функционала сортировки.

Соответственно можно было сразу в html прописать:

<foreach Product>where('catalog\_id = ? AND brand\_id IN (?)', $catalog\_id, $brand\_ids)>

* Иногда, когда слишком много элементов, чтобы вывести их на сайте, желательно использовать разбивку их на страницы - пагинацию.

Для этого после того как получили список, к примеру товаров определенной категории:

d()->products = d()->Product->where('catalog\_id = ?', $catalog\_id);

Необходимо специальными командами создать пагинатор.

Указываем сколько хотим видеть элементов на одной странице

d()->products->paginate(12);

Потом используя специальный класс d()->Paginator сгенерировать разбитие на страницы и занести результат в переменную d()->paginator

d()->paginator = d()->Paginator->generate(d()->products);

**Нюанс:** при использовании пагинатора, вручную указанное значение в команде ->limit() будет автоматически переопределено.

После этого в html можно писать:

<foreach products>

<div>{.title}{{edit}}{{delete}}</div>

</foreach>

<?php if (d()->paginator != '') { ?>

<div class = "paginator">

<?php if (d()->paginator\_left != '') { ?>

<a class = "paginator\_left" href = "{paginator\_left}"><</a>

<?php } ?>

{paginator}

<?php if (d()->paginator\_right != '') { ?>

<a class = "paginator\_right" href = "{paginator\_right}">></a>

<?php } ?>

</div>

<?php } ?>

Кроме определенной d()->paginator, также автоматически определяются переменные d()->paginator\_left и d()->paginator\_right, в которых будет находится ссылка на предыдущую или следующую страницы. Выводить их не обязательно.

Если на странице были определены в ссылке различные GET-параметры, то все они сохраняется на ссылках на страницы.

Для работы пагинатора используется GET-параметр "page". Вторая страница имеет значение $\_GET['page'] = 1. И так по возрастанию

* Для того, чтобы узнать количество полученных элементов из таблицы, надо вызвать метод count:

d()->products = d()->Product->where('catalog\_id = ?', $catalog\_id);

print d()->products->count;

Если был использован функционал пагинатора, то в данном методе будет возвращаться только кол-во элементов именно для текущей страницы. Однако при этом все равно можно легко получить общее кол-во используя метод found\_rows

print d()->products->found\_rows;

* В переменную кроме списка элементов можно получить и только один конкретный элемент и работать, выводить его удобно без использования цикла:

d()->one\_product = d()->Product->where('url = ?', $url);

Теперь в html-файле можно писать:

{one\_product.title}

Что является аналогом:

<?php print d()->one\_product->title; ?>

А также писать:

{{edit d()->one\_product}}{{delete d()->one\_product}}

При определении основного элемента, который является главным для текущей страницы (к примеру выводим страницу конкретной новости), принято определять в d()->this:

d()->this = d()->News->where('url = ?', $url);

В html:

{.title}{{edit}}{{delete}}

Что позволяет уменьшить объем кода и ускорить процесс написания.

Вместо конструкций вида: ->where('url = ?', $url), ->where(id = ?', $id) можно писать такие варианты:

d()->one\_product = d()->Product->find\_by\_url($url);

d()->one\_product = d()->Product->find\_by\_id($id);

* Если требуется проверить, получили ли мы из таблицы данные, или там нет строк по нашему запросу, можно, конечно, использовать сравнение ->count с 0, но есть более удобные варианты записи.

Это проверка, которая выполняется, если результат пустой:

if (d()->products->is\_empty) { … }

Это проверка, которая выполняется, если есть хотя бы 1 строка в результате:

if (d()->products->ne) { … }

* Чтобы самому через php-код добавить новую запись в таблицу, надо использовать следующий код:

$new\_product = d()->Product->new;

$new\_product->title = Крутой товар';

$new\_product->save;

$new\_product\_id = $new\_product->insert\_id;

Т.е. для объявления о том, что создаем новую строку говорит команда ->new, а завершает сам процесс команда ->save;

Если для дальнейшего кода вам потребуется знать id строки, которую вы добавили, ее можно получить через команду ->insert\_id примененную к перменную через которую добавляли, сразу после команды save.

Если в таблице нет каких-либо столбцов, а вы их тут указали

$new\_product->new\_price = 999;

То такой столбец будет автоматически создан. Тип поля будет текстовый.

* **Как определить новый контроллер, модуль, т.е. новую папку mod\_**
* Для создания, к примеру модуля Новостей, необходимо

Создать таблицу: news

В файл app/router.func.php добавить в конец строчки

route('/news/index', 'news#index');

route('/news/', 'news#show');

В app/fields/ определить шаблон для добавления/редактирования новости.

Создать папку app/mod\_имя\_таблицы\_бд

app/mod\_news

В нее скопировать файл app/mod\_pages/pages.func.php и переименовать в

app/mod\_news/news.func.php

Затем в этом файле в определении названии класса заменить Pages на свою таблицу, т.е. News:

class NewsController

после чего создать функции

index() - будет открываться при заходе на ссылку http://site/news/, где будет выводиться заголовок страницы и список всех новостей, разбитых пагинатором.

show() - будет открываться при заходе на ссылки типа http://site/news/news1, где будет выводится информация по конкретной новости.

Пример данной папки app/mod\_news залит в папку other/news/ (вне папки default).

Шаблон редактирования, строки в роутер, таблицу в БД добавлять надо вручную

* Если сайт не посадочная страница, то пункты меню принято реализовывать как пункты таблицы pages, у которых стоит галочка is\_mainmenu "Выводить в меню". Сделано это чтобы не дублировать лишнюю информацию, т.е. в одной таблице будут чисто заголовок и ссылка, а в другой таблице (pages) все равно придется создать строку для данного url. Поэтому оптимальнее сделать как описано.
* В Редактировать - Опции есть поле email, которое используется для определения почты администратора сайта, куда отправлять письма.

Принято заносить в данное поле значения через запятую, чтобы можно было указать несколько почт.

* Т.к. в таблице options у нас обычно только 1 строка, получать оттуда на вывод данные можно просто и удобно:

{Option.email}

if (d()->Option->title1 != '’) { ...

* Ссылки конечных страниц, конкретных товаров, статей, каталогов, должны быть без «/» в конце (/page1, /aticles/article1, /products/product3).

Страницы типа «Все статьи», «Все новости» и т.д. принято делать с «/» в конце (/articles/, /news/). Но не обязательно.

* Для всех элементов выводимых через CMS должны присутствовать элементы редактирования и удаления {{edit}}{{delete}}, а перед выводом списка элементов должны выводится элементы добавления и сортировки этих элементов {{add ‘articles’}}{{sort\_icon ‘articles’}}
* **Работа с формами**
* GET формы для фильтров и т.д. реализовывать как обычно
* POST формы для отправки писем и т.д. необходимо реализовывать специальным образом:

Начало формы задается таким образом:

{{form 'form\_name', 'action' => '/ajax/form1', 'id' => 'form\_name\_id', 'ajax' => true}}

где:

'form\_name' - название формы, латиницей, без пробелов и спец символов

'action' => '/ajax/form1' - указывается ссылка для обработки формы, куда отправятся данные с формы

'id' => 'form\_name\_id' - просто задается атрибут id для данного тега <form>

'ajax' => true - указывает, что форма должна отправляться через ajax

В случае, если необходима возможность прикрепления и отправка файла в форме, опцию ajax придется убрать, и прописать дополнительный параметр:

{{form 'form\_name', 'action' => '/ajax/form1', 'id' => 'form\_name\_id', 'enctype' => "multipart/form-data"}}

Закрывается форма:

{{/form}}

Поля (input, textarea) можно выводить следующим образом:

{{input 'name', 'id' => 'input\_name', 'class' => 'def\_input requir\_field', 'placeholder'=>'Ваше имя'}}

{{textarea 'text', 'id' => 'textarea\_text', 'class' => 'def\_textarea requir\_field', 'placeholder'=>'Комментарий'}}

Можно писать обычным образом, но надо учесть, что имя должно задаваться "data[field\_name]"

<input type = "text" name = "data[name]" id = "input\_name" value = "" placeholder = "Ваше имя">

Поля типа галочек, select и т.д. писать обычным образом, но надо учесть, что имя должно задаваться "data[field\_name]"

На поле для ввода телефона желательно использовать скриптовую маску типа:

+7 (111) 111-1111

Также нами обычно принято в каждой форме делать спрятанное поле data[check], которое обязательно должно быть пустым, для легкой проверки на спам-роботов:

<input type = "text" name = "data[check]" id = "input\_check" value = "" style = "display: none;">

В файле app/validator.init.ini можно указать проверки на поля. Но т.к. данные проверки происходят на стороне php после отправки формы, в случае, если это не ajax форма, желательно продублировать необходимые проверки на стороне js вручную

Каждая проверка в файле задается минимиум 2мя строчками. В одной после ключевого слова validator указывает название формы, которые мы дали при ее объявлении, потом название поля. Во второй строке указывается тип проверки, и текст об ошибке:

[validator.form\_name.name]

required.message=Вы не указали Имя!

Есть следующие стандартные типы проверок:

required - поля не должно быть пустым

valid\_email - поле должно быть обязательно заполнено и проходить валидацию как email

Для удобства можно использовать свои проверки, основные из них можно получить скачав файл app/helper.func.php из папки "default":

valid\_email2 - получаемое значение тримируется, если пустое то проверка считается пройденной. Если не пустое, то обязательно должно проходить валидацию как email

valid\_phone - получаемое значение тримируется, и оно обязательно должно проходить валидацию как телефон по маске типа: +7 (111) 111-1111

Также в данном файле принято определять различные вспомогательные функции, не обязательно для валидации.

Можно сразу несколько проверок указывать на одно поле, к примеру:

[validator.form\_name.phone]

required.message=Ошибка!

valid\_phone.message=Ошибка!

После создания формы со всеми необходимыми полями, необходимо в роутере определить функцию для обработки указанного у формы URL'а. Так как верстка и т.д. нам тут не требуется, необходимо переопределять именно main:

route('/ajax/form1', 'main', 'pages#send\_form1');

Соответственно в app/mod\_pages/pages.func.php описываем функцию для обработки функции по следующим правилам.

Разберем обычный вариант, т.е. ajax форму

Основной код пишется в месте указанном многоточием:

function send\_form1 () {

if (d()->validate('send\_form1')) {

if (isset(d()->params['check']) && d()->params['check'] == ''){

…

}

exit;

}

d()->reload();

}

Эта функцию d()->validate('send\_form1') как раз вызывает на проверку все наши валидации. В качестве параметра передаем название формы. После вызова данной функции, все наши поля доступны в массиве d()->params, причем при обращении не требуется указывать data, т.е. к примеру телефон будет храниться в

d()->params['phone']

В основном коде необходимо сделать 2 основных действия: отправить письмо с данными на почты, указанные для получения писем, а также сохранить в БД в отдельную таблицу эти данные, чтобы хранились все заполненные формы в БД.

Название таблицы не принципиально, обычно использую таблицу: contacts. Ее необходимо вручную создать до отправки первого письма через Скаффолдинг - Создать.

Отправка письма реализуется на почты указанные в d()->Option->email, с учетом того, что там могут быть несколько почт, разделенных запятой. Отправка происходит либо через определенную реальную почту, т.е. через SMTP, либо нет. Какой тип отправки происходит, а также данные для отправки через SMTP задаются в соответствующих полях в Редактировать - Опции. Обычно используем отправку через SMTP почты на яндексе.

Для нее надо ставить галочку ssl и в качестве порта указывать: 465

Почта при регистрации должна быть подтверждена номером телефона.

В качестве тела письма указывается html файл, по аналогии как указывали шаблон для вывода:

d()->pages\_mail\_f1\_tpl()

После отправки писем и сохранения в БД, обычно делаем отправку пользователя на какую-нибудь страницу, к примеру, где выводит текст, что Спасибо, с Вами свяжутся. Либо просто на главную, а там уже можно сделать всплывающее окно, либо что еще захочется.

Это можно сделать, указав через print любой обычный js код, к примеру:

print 'document.location.href="/thankyou"';

Эта команда разумеется должна быть одной из последних, после чего идет команда exit;

В случае, если форма не ajax, то команды d()->reload() быть не должно - поломается отправка.

И писать js-код как указано выше не будет возможность, можно будет делать перенаправление средствами php:

header ('Location: /thankyou');

exit;

Пример обработчика формы (без описания самой формы, валидаций, роутера) залит в папку other/form/mod\_pages/.

Нюанс - в предоставленном файле other/form/mod\_pages/pages.func.php описана только одна функция, show(), index() удалены, чтобы не мешать. В рабочем сайте удалять ничего не надо, просто определять новую функцию по примеру и все.

* **Сортировка изображений**

Т.к. в функционале сортировки используется для отображения пунктов значение из title, а в таблице изображений, к примеру когда делаем фотогалерею - photos, часто бывает только поле для залития изображения - image

Поэтому при сортировке будут просто пустые блоки и не понятно, где какое изображение.

Поэтому можно поступить следующим образом.

В любой модуле, к примеру в app/mod\_pages/, необходимо залить файл other/sort/photo.class.php

И все, при сортировке будут миниатюры изображений выводится. В данном файле мы доопределяем класс d()->Photo, переопределяя у него вывод по запросу поля title, выводя превью картинки по полю image.

Если у вас таблица не photos, а к примеру sliders - то надо изменить название файла (маленькими буквами, в единственном значении): slider.class.php

И внутри файла имя класса переопределить, вместо Photo - Slider

* Часто, после того как сайт уже работает и наполнен, возникает необходимость создать новый таблицы в БД, и новые поля в уже существующих таблицах.

А заливать дамп БД уже нельзя, т.к. перезапишутся рабочие данные.

Для этого, после того как на локальной версии создано все что нужно, можно использовать команду: Скаффолдинг - Миграция. Там нажать кнопку "Сгенерировать и сохранить файл схемы". После этого сгенерируется файл схемы БД (все таблицы, их поля): app/schema.ini

После этого залив данный файл на основной сайт, можно использовать команду: Скаффолдигн - Обработать схему. После этого в БД будут созданы все таблицы и поля указанные в схеме. Данные не затронутся, если в БД есть таблицы или поля, которых нет в схеме - то они также останутся на месте.

* Файл app/error\_404.html выводится на сайте, когда открывается несуществующая страница.

Перенаправление на эту страницу реализуется:

if (d()->this->is\_empty) {

d()->message="Страница не существует".d()->add(array('pages','url'=>$url));

return d()->error('404');

}

Где d()->this - это текущий объект, который должен был вывестись (может быть и другая переменная).

В d()->message - заносится текст, который будет отображаться на странице, и к этому текст также дописывается результат функции d()->add(), т.е. иконка добавления выведется админу сайта, чтобы сразу можно было создать страницу по текущему адресу.

Учитывайте, что название таблицы передаваемое в d()->add, т.е. pages в примере, должна указываться именно необходимая таблица. Т.е. если это функция отображения конкретной новости, то должна быть указана таблица news

**Нюанс**

Во всех функциях типа show, который выводятся конкретную страницу, новость, товар и т.д. обязательно должна быть проверка на пустоту и перенаправление на 404 страницу, иначе просто будет бесконечное количество одинаковых страниц, где в контентной части будет просто пустота

Функции для удобства

* url() - возвращает текущий URL страницы, без ведущего (первого) слеша, без названия домена, без GET-параметров
* h($str) - возвращает строку $str обработанную функцией htmlspecialchars()
* userdate($date\_at) - преобразуется дату вида 2019-03-04 в: 4 марта 2019
* preview - создает из исходной картинки файл картинки с указанными шириной-высотой, чтобы на странице, где много элементов с маленькими картинками выводятся, например, фотогалерея - не грузить большие фотки зря.

Пример вызова в html:

{{preview d()->this->image, 500, 300}}

{{preview d()->this->image, 500, 'auto'}}

Вначале указывается ширина, потом высота.

Можно задать оба значения в виде числа - тогда если пропорции указаны не верно, то картинка будет обрезаться, чтобы получить картинку указанных размеров, без ее растягивания

Если задать, к примеру, только ширину числом, а в высоте указать 'auto' - тогда картинка будет получаться с указанной шириной, а высота будет автоматически определяться с сохранением пропорций.