docs-v5 Documentation

Выпуск 1

Alexandr Zubarev

26 February 2016

Оглавление:

1.1 Установка

1.1.1 Локальная установка ALMA.CMS v5

В данном примере рассматривается установка ALMA.CMS на локальный сервер XAMPP под ОС Window 7. Если на сервере не установлен Parser3, то для установки необходимо скачать Программу установки для Win32 и установить его в папку `C:\xampp\cgi-bin`.

Склонировать ALMA.CMS v5 в папку `mysite.local` можно следующими командами:

```
cd /C/xampp/htdocs/
git clone https://bitbucket.org/alma-com/v5.git mysite.local
cd mysite.local
```

На данный момент актуальная версия CMS хранится в ветке `develop`, чтобы на нее переключиться и сделать её `master`, необходимо выполнить следующии команды:

```
git checkout develop
git branch -D master
git branch -m master
```

После этого необходимо импортировать БД в кодировке UTF-8. Файл для импорта БД называется `v5.sql`

Далее необходимо скачать NConvert для Windows и файл `nconvert.exe` расположить `C:/xampp/htdocs/mysite.local/cgi-bin/`.

Отлично, осталось немного. Находим в корне сайта файл `config.example.p`, его необходимо скопировать и переименовать в `config.p`. В этом файле настраивается подключение к БД и путь к NConvert. Вот что должно у вас получиться:

```
$sql_table[mysite]
$sql_login[root]
$sql_pass[]

$nconvert_path[/cgi-bin/]
$nconvert_name[nconvert.exe]
```

Файл `.htaccess.example` тоже копируем и переименовываем в `.htaccess`, открываем его и вместо второй строки вставляем следующую:

```
Action parsed-html /cgi-bin/parser3.exe
```

1.1.2 Создание репозитория для сайта

После того как сайт был локально развернут, необходимо создать репозиторий и слить в него все данные. Для примера рассматривается создание репозитория в Bitbucket. Находим кнопку Create repository, имя репозитория вводим `mysite.ru` и сохраняем.

Далее в Git bash необходимо выполнить следующие команды:

```
cd /C/xampp/htdocs/
git remote rm origin
git remote add origin https://bitbucket.org/ваш_логин/mysite.ru.git
git push -u origin --all # pushes up the repo and its refs for the first time
git push -u origin --tags # pushes up any tags
```

После этого вы имеете удаленный репозиторий для своего сайта!

1.1.3 Деплой на боевой сервер

Первым делом необходимо сгенерировать открытый SSH-ключ для пользователя, в котором будет располагаться сайт.

Предупреждение: Если у пользователя уже имеется сгенерированный открытый SSH-ключ, то его создавать не нужно.

Для этого подключаемся по SSH к серверу. Если вы зашли под root-ом, то чтобы выполнять команды от имени `mysite`, нужно прописать в консоль:

```
sudo -u mysite bash
```

Далее выполняем команды, (если .ssh не существует - нужно создать: `mkdir ~/.ssh`), при выполнении укажите любое имя для ключа, а поле пароля оставьте пустым.:

```
cd ~/.ssh
ssh-keygen -t rsa
```

Затем запускаем ssh агент и добавляем созданный ключ:

```
ssh-agent /bin/bash
ssh-add ~/.ssh/id_rsa
```

После этого в папке `~/.ssh`, можно увидеть два файла id_rsa и id_rsa.pub (вместо id_rsa имя ключа, которое вы ввели). Скопируйте себе .pub файл, откройте в редакторе и всё его содержимое добавьте в настройки вашего сайта на Bitbucket.

Найти этот разделы можно так: `Страница проекта -> Settings -> Deployment keys`.

После этого необходимо склонировать с удаленного репозитория проект на боевой сервер:

```
cd ~/путь_до_проекта/
git init
git remote add origin git@bitbucket.org:ваш_логин/mysite.ru.git
git pull origin master
```

Далее необходимо настроить БД. Файл“config.p“ должен быть похож на это:

```
$sql_table[mysite]
$sql_login[root]
$sql_pass[password]
```

В файле “.htaccess“менять ничего не нужно.

И последнее, в папке `cgi-bin` необходимо выставить права доступа 755 следующим файлам `parser.cgi` и `NConvert`.

Предупреждение: Если выдается ошибка Internal Server Error, то скорее всего побились бинарные файлы parser.cgi и NConvert, необходимо скачать файлы под нужную ОС.

1.1.4 Автоматизация деплоя

Для того чтобы не залезать каждый раз на сервер и не выполнять команду `git pull`, необходимо:

1. В папке `cgi-bin` создать файл `deploy.sh` и вставить следующий код:

```
#!/bin/bash
echo Content-type: text/plain
echo

cd ..
echo "-----------------------------------------------"
echo "|git commit on server:"
echo "-----------------------------------------------"
git add -A
git commit -m "---server commit---"

echo
echo "-----------------------------------------------"
echo "|git pull:"
echo "-----------------------------------------------"
git pull -X theirs origin master

echo
echo "-----------------------------------------------"
echo "|git status:"
echo "-----------------------------------------------"
git status
```

2. Файлу `deploy.sh` дать права доступа 755.
3. в Bitbacket-е `Страница проекта -> Settings -> Webhooks` необходимо добавить URL `http://mysite.ru/deploy.sh`.

Теперь при каждом PUSH-е в удаленный репозиторий будет вызываться хук, который цепляет скрипт на сервере.

1.2 Ядро доработок

Ядро доработок (kernel) в приложении лежит в основе «первоначальной загрузки» всего сайта. Находиться данный провайдер `/app/kernel.p`.

Данный файл нужен для подключение и использования классов помощников, которые должны отрабатывать перед загрузкой всего сайта:

```
^use[/app/helpers/alma.p]
^alma:num_decline[102;сообщение;сообщения;сообщений]
```

Также для определения переменных которые подключаются во все шаблоны:

```
^view:share[var;value]
```

Для сохранение переменных, объектов, позорного кода в Реестр, чтобы потом из Реестра в нужном вам месте вытянуть значения.

```
^registry:set[shame;2*2=4]
```

1.2.1 Помощники

Помощники - это пользовательские классы, которые расширяют функционал сайта. Пример помощника можно найти `/app/helpers/alma.p`. Данный класс содержит один метод `num_decline`, который позволяет склонять имена существительных после числительных.

1.3 Представления

1.3.1 Основы использования

Представления (views) содержат HTML-код, передаваемый вашим приложением. Это удобный способ разделения бизнес-логики и логики отображения информации. Представления находятся в каталоге `class/block/nameYouBlock/views`.

Простое представление выглядит примерно так:

```
<!-- Представление class/block/bnews/views/detail.pt -->

<div class="bnews">
    <h1>$name</h1>
    <div class="date">$date</div>
    <a class="photo fancybox_bgal" href="/images/bnews/b_${path}">
        <img src="/images/bnews/m_$path" alt="">
    </a>
    }
    ^untaint{$full_text}
</div>
```

Подключить представления можно так:

```
$data[
        $.name[Наименование новости]
        $.date[12 Сентября 2016]
        $.path[/bank/image.jpg]
        $.full_text[Полный полный текст]
]
^view:render[bnews/views/detail.pt;$data]
```

1.3.2 Вывод представления

Функция `^view:render[path;data]` - выводить представления, где:

- `path` - путь к шаблону. Если путь относительный, то поиск относительно `/class/block/`.
- `data` - hash с переменными.

1.3.3 Передача данных во все представления

Иногда вам нужно передать данные во все представления вашего приложения. Для этого необходимо в ядро доработок `/app/kernel.p` написать следующее:

```
^view:share[nameSite;Альма]
```

После этого в любом шаблоне вы можете использовать переменную `nameSite`.

1.4 Eloquent ORM

Система объектно-реляционного отображения (ORM) Eloquent — красивая и простая реализация шаблона ActiveRecord для работы с базами данных. Каждая таблица имеет соответствующий класс-модель, который используется для работы с этой таблицей.

1.4.1 Основы использования

Для начала создадим модель Eloquent. Модели располагаются в папке `app\models`. Все модели Eloquent наследуют базовый класс `eloquent` и инициализируют переменную `$_table` с названием таблицы.

Пример создание модели:

```
@CLASS
role

@BASE
eloquent

@auto[]
$_table[roles]
```

Когда модель определена, у вас всё готово для того, чтобы можно было выбирать и создавать записи.

1.4.2 Методы Eloquent

`@all[]`
Получение всех записей модели.

```
    $roles[^role:all[]]
```

`@find[string $id]`
Получение записи по любому уникальному полю.

```
    $roles[^role:find[1]]
```

`@new[]`
Создание нового объекта модели.

```
    $roles[^role:new[]]
```

`@empty[]`
: Содержит пустоту, обычно используется при обновлении/добавлении записи.

```
$roles.name[^role:empty[]]
```

`@sql[]`
: Выполнение произвольного sql запроса. `%TABLE%` - название таблицы, `%KEY%` - ключ таблицы.

```
$roles[^role:sql[SELECT %KEY% FROM %TABLE% LIMIT 2]]
```

`@save[]`
: Сохранение данных в БД (только динамический вызов). Работает только для одной записи.

```
$roles[^role:find[admin]]
$roles[^role.save[]]
```

`@delete[]`
: Удаление данных из БД (только динамический вызов). Работает как для одной записи, так и для многих сразу.

```
$roles[^role:all[]]
$roles[^role.delete[]]
```

`@get_table[]`
: Получение название таблицы.

```
$name[^role:get_table[]]
```

`@get_key[]`
: Получение название ключа таблицы.

```
$key[^role:get_key[]]
```

`@get_id[]`
: Получение id записи, работает если только одна запись в моделе.

```
$roles[^role:find[admin]]
$id[^role:get_id[]]
```

`@get_unique_fields[]`
: Получение списка уникальных столбцов.

```
$fields[^role:get_unique_fields[]]
```

Пример сохранения новой записи

```
$roles[^role:new[]]
$roles.name[Администратор]
$roles[^role.save[]]
```

Пример обновления записи

```
$roles[^role:find[1]]
$roles.name[Терминатор]
$roles[^role.save[]]
```

1.4.3 Модель любой таблицы

Класс Eloquent позволяет обращаться к любой таблице без создания дополнительных классов.

```
^eloquent:connect[roles]
$roles[^eloquent:all[]]
^eloquent:close[]
```

Метод `connect` устанавливает соединение с таблицей, с которой Eloquent должен работать. В конце необходимо вызвать метод `clear`, для того чтобы удалить соединение с таблицей.

1.5 Авторизация

Класс авторизации находится `app\auth.p`. Данный класс можно расширять собственными методами, а также переопределять встроенные методы.

По умолчанию логином является E-mail.

1.5.1 Методы класса

`@login[id_user]`
: Ручной вход в систему по id или логину пользователя.

```
^auth:login[test@gmail.com]
```

`@logout[]`
: Выход из системы текущего авторизованного пользователя.

```
^auth:logout[]
```

`@logout_all[id_user]`
: Выход пользователя из системы со всех устройств. Где `$id_user` - id или логин пользователя.

```
^auth:logout_all[test@gmail.com]
```

`@attempt[login;password;remember]`
: Попытка авторизации пользователя. Возвращает `true` или `false`. Для того чтобы запомнить пользователя, необходимо передать `$remember`.

```
^if(^auth:attempt[test@gmail.com;MyPass;1]){
  Вы авторизованны!
}{
  Неудачная попытка
}
```

`@user[]`
: Получение текущего авторизованного пользователя. Возвращает модель Eloquent.

```
$user[^auth:user[]]
$user.email - вывод e-mail пользователя
```

`@check[]`
: Определение, авторизован ли текущий пользователь. Возвращает `true` или `false`.

```
^if(^auth:check[]){
  Вы авторизованны!
}{
  Пожалуйста авторизуйтесь!
}
```

`@get_url_login[]`
Получение URL авторизации.

`@get_action_login[]`
Получение URL отправки формы при авторизации.

`@get_url_forgot[]`
Получение URL забыли пароль.

`@get_action_forgot[]`
Получение URL отправки формы забыли пароль.

`@get_url_registration[]`
Получение URL регистрации.

`@get_action_registration[]`
Получение URL отправки формы при регистрации пользователя.

`@get_url_personal[]`
Получение URL личного кабинета.

1.5.2 Вывод стандартных форм

`@show_form_login[]`
Форма авторизации.

`@login_request[]`
Выполнение запроса на авторизацию пользователя.

`@show_form_registration[]`
Форма регистрации.

`@registration_request[]`
Выполнение запроса на регистрацию пользователя.

`@show_form_forgot[]`
Форма восстановления пароля.

`@forgot_request[]`
Выполнение запроса на восстановление пароля.

`@show_form_forgot_password[]`
Форма изменения пароля при восстановлении пароля.

`@forgot_password_request[]`
Выполнение запроса на сброс пароля.

1.5.3 Пример создание своей формы авторизации

Для того чтобы переопределить стандартную форму авторизации, необходимо в `app\auth.p`, создать метод `@show_form_login[]` со своей формой.

```
@CLASS
auth

...

@show_form_login[]
<form method="POST" action="^auth:get_action_login[]" class="js-ajax-form">
<span class="js-alert"></span>
<table>
  <tr>
    <td>E-mail:</td>
    <td><input type="email" name="email" value=""></td>
  </tr>
  <tr>
    <td>Пароль:</td>
    <td><input type="password" name="password" value=""></td>
  </tr>
  <tr>
    <td colspan="2">
      <label for="remember">
        <input type="checkbox" id="remember" name="remember" value="1">
        Запомнить меня на этом компьютере
      </label>
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <input type="submit" name="submit" value="Войти">
      <a href="^auth:get_url_forgot[]">Забыли пароль?</a>
    </td>
  </tr>
</table>
</form>
```

1.6 Проверка ввода

Для проверки ввода и получения сообщений об ошибках разработан класс validator.

1.6.1 Пример использования

Рассмотрим пример регистрации пользователя. Необходимо проверить что:

1. Пользователь заполнил поле E-mail, пароль и подтверждение пароля.
2. Введенный E-mail является действительно E-mail.
3. Введенный E-mail не занят в нашей базе данных.
4. Длина пароля должна быть более 6 символов и менее 255.
5. Введенный пароль совпадает с подтверждением пароля.

Даа, получается очень много проверок нужно сделать перед регистрацией пользователя. Но при помощи класса validator, все эти проверки можно уместить в 6 строк кода. Вот смотрите:

```
$hash[
  $.email[required|email|unique:users]
  $.password[required|min:6|max:255]
  $.password_confirmation[same:password]
]
$hash_result[^validator:make[$hash]]
```

`$.email` - это название поля.

`required` - это условие что поле обязательно для заполнения.

`email` - это условие что поле должно быть в формате E-mail.

И т.д. В конце вызывается метод `make`, который возвращает hash следующего вида:

```
$hash[
  $.status[error] - результат проверки success или error
  $.message[
    $.field1[Поле "field1" обязательно для заполнения.]
    $.field2[Поле "field2" имеет неправильный формат.]
  ]
]
```

Если ваша форма использует класс Ajax форму, то вы можете сразу информировать пользователя о том, что поля неправильно заполнены. Для этого необходимо в формате JSON вернуть hash `$hash_result`. Вот полный пример:

```
$hash[
  $.email[required|email|unique:users]
  $.password[required|min:6|max:255]
  $.password_confirmation[same:password]
]
$hash_result[^validator:make[$hash]]

^if($hash_result.status ne 'error'){
  ...
  $hash_result.url[^auth:get_url_login[]]
}

$result[^json:string[$hash_result]]
```

1.6.2 Методы класса

`@make[hash;extra_names]`
: Валидация ввода. `$extra_names` - дополнительный hash с красивыми именами

```
^validator:make[$.square[required];$.square[Площадь]]
```

`@beatiful[key]`
: Получение красивого названия поля

```
^validator:beatiful[square]
```

`@is_email[email]`
: Проверка формата E-mail. Возвращает `true` или `false`.

```
^if(^validator:is_email[test@gmail.com]){
  Правильный формат
}
```

1.6.3 Доступные правила проверки

between:min,max Поле должно быть числом в диапазоне от `min` до `max`.

email Поле должно быть корректным адресом E-mail.

exist:table Поле должно существовать в заданной таблице `table`.

max:value Значение поля должно быть меньше или равно `value`.

min:value Значение поля должно быть более или равно `value`.

required Проверяемое поле должно иметь непустое значение.

same:field Поле должно иметь то же значение, что и поле `field`.

unique:table,exceptId Значение поля должно быть уникальным в заданной таблице `table`. Если задан `exceptId`, то из проверки исключается данная запись (Обычно это нужно при редактировании записи, чтобы исключить самого себя из проверки).

1.7 Класс url

Данный класс помогает при работе с url.

1.7.1 Методы класса

`@to[]`
Редирект к URL. Может некорректно работать с относительными путями.

```
^url:to[/my/page/please/now]
```

`@now[]`
Возвращает текущий URL без имени домена.

```
^url:now[]
```

`@root[]`
Возвращает главный URL сайта без имени домена.

```
^url:root[]
```

`@root_to[]`
Редирект на главный URL сайта.

```
^url:root_to[]
```

`@back[url_default]`
Возвращает предыдущий URL без имени домена. Если нет предыдущего URL, то возвращает `$url_default`, по умолчанию главный URL сайта.

```
^url:back[]
```

`@back_to[url_default]`
Редирект в предыдущий URL. Если нет предыдущего URL, то возвращает `$url_default`, по умолчанию главный URL сайта.

```
^url:back_to[]
```

1.8 Ajax форма

Рассмотрим пример работы Ajax формы:

```
<form method="POST" action="reg.html" class="js-ajax-form">
<span class="js-alert"></span>
<table>
  <tr>
    <td>Имя:</td>
    <td><input type="text" name="name" value=""></td>
    <td><span class="js-has-error" data-field="name"></span></td>
  </tr>
  <tr>
    <td>E-mail:</td>
    <td><input type="email" name="email" value=""></td>
    <td><span class="js-has-error" data-field="email"></span></td>
  </tr>
  <tr>
    <td>Пароль:</td>
    <td><input type="password" name="password" value=""></td>
    <td><span class="js-has-error" data-field="password"></span></td>
  </tr>
  <tr>
    <td>Подтверждение пароля:</td>
    <td><input type="password" name="password_confirmation" value=""></td>
    <td><span class="js-has-error" data-field="password_confirmation"></span></td>
  </tr>
  <tr>
    <td colspan="2"><input type="submit" name="submit" value="Регистрация"></td>
  </tr>
</table>
</form>
```

- `js-ajax-form` - Класс для инициализации Ajax формы.
- `js-alert` - Место куда будут выводиться сообщения.
- `js-has-error[data-field="name_field"]` - Место куда будут выводиться сообщения относящиеся к полю `name_field`.
- `has-error` - класс добавляемый к полям, у которых есть сообщение.

Для того чтобы Ajax форма автоматически понимала, что ей делать, необходимо на обрабатывающей форму странице сформировать JSON следующего вида:

```
{
  "status": "error", //Статус провреки success или error
  "message": {
    "field1" : "Поле field1 обязательно для заполнения.
  },
  "url" : "/my/url" //Редирект к URL. Является необязательным.
}
```

Данный JSON быстро сформировать поможет класс validator. Рассмотрим пример обрабатывающей форму страницы:

```
$hash[
  $.email[required|email|unique:users]
  $.password[required|min:6|max:255]
  $.password_confirmation[same:password]
```

```
]
$hash_result[^validator:make[$hash]]

^if($hash_result.status ne 'error'){
  ...
  $hash_result.url[^auth:get_url_login[]]
}

$result[^json:string[$hash_result]]
```