**Функционал нашей CMS**

Первым делом надо точно определиться, что будет делать наша CMS. Итак, вот список функций:

Клиентская часть:

* Главная страница, на которой выводиться список последних 5 статей
* Страница со списком всех статей
* Страница просмотра отдельной статьи

Серверная часть:

* Вход/выход для администратора
* Список всех статей
* Добавление новой статьи
* Редактирование существующей статьи
* Удаление существующей статьи

Каждая статья имеет собственный заголовок, резюме и дату публикации.

**Планирование работ**

Для создания нашей CMS нужно сделать следующие шаги

1. Создать базу данных
2. Создать таблицу articles
3. Сделать файл конфигурации
4. Построить класс Article
5. Написать скрипт клиентской части index.php
6. Написать скрипт серверной части admin.php
7. Создать шаблон клиентской части
8. Создать шаблон серверной части
9. Создать таблицу стилей и логотип системы

**Примечание:** на страницах уроков приводится код для нашей CMS готовый к копированию в ваши текстовые файлы. Если у вас нет желания создавать файлы самостоятельно, то можно скачать архив с исходниками, в котором содержатся все нужные папки и файлы.

## Шаг 1. Создаем базу данных

На первом шаге нужно создать базу данных MySQL для хранения содержания. Можно сделать так:

1. **Запускаем программу клиент  mysql** Открываем окно терминала и набираем команду

mysql -u username -p

После запроса введите пароль для доступа к MySQL.

username - имя пользователя, который имеет полномочия для создания баз данных. В случае работы на локальном компьютере можно использовать root, хотя для безопасности всегда следует создавать пользователя с другим именем для решения задач администрирования.

1. **Создаем базу данных** После метки mysql> вводим:

create database cms;

И нажимаем Enter.

1. **Выходим из программы клиента mysql** После метки mysql> вводим:

exit

И нажимаем Enter.

Теперь у нас есть пустая база данных, в которой можно размещать таблицы и содержание.

Для решения такой задачи также можно воспользоваться инструментами для администрирования баз данных, таким как phpMyAdmin, cPanel или Plesk (если они установлены на вашем сервере). В некоторых случаях использование подобных инструментов является единственным доступным для пользователя инструментом для работы с базами данных (ситуация зависит от правил, установленных на вашем хостинге).

## Шаг 2. Создаем таблицу articles

Наша простая CMS имеет единственную таблицу в базе данных: articles. В ней содержатся все статьи в нашей системе.

Создадим схему таблицы. Схема таблицы описывает типы данных, которые могут содержаться в таблице и информацию о ней.

Создаем текстовой файл tables.sql на жестком диске и добавляем в него следующий код:

|  |  |  |
| --- | --- | --- |
| 01 | DROP TABLE IF EXISTS articles; | |
| 02 | CREATE TABLE articles |

|  |  |
| --- | --- |
| 03 | ( |
| 04 | id              smallint unsigned NOT NULL auto\_increment, | |

|  |  |
| --- | --- |
| 05 | publicationDate date NOT NULL,                              # Когда статья опубликована |
| 06 | title           varchar(255) NOT NULL,                      # Полный заголовок статьи |

|  |  |
| --- | --- |
| 07 | summary         text NOT NULL,                              # Резюме статьи |
| 08 | content         mediumtext NOT NULL,                        # HTML содержание статьи |

|  |  |
| --- | --- |
| 09 |  |
| 10 | PRIMARY KEY     (id) | |

|  |  |
| --- | --- |
| 11 | ); |

Выше приведенный код определяет схему таблицы articles. Он написан на SQL, языке для создания и манипулирования базами данных в MySQL (и во многих других системах).

Разберем выше приведенный код

1. **Создаем таблицу articles** Выражение DROP TABLE IF EXISTS articles удаляет любую существующую таблицу articles  (вместе с данным - осторожно!). Мы выполняем данную операцию чтобы в базе не было двух таблиц с одинаковыми именами. Выражение CREATE TABLE articles ( ) создает новую таблицу articles. Код, размещенный в скобках, определяет структуру данных в таблице...
2. **Определяем для каждой статьи уникальный ID** Теперь можно определять структуру таблицы. Таблица состоит из набора полей (также их называют столбцами). Каждое поле содержит опредленный тип информации о статье. Сначала мы создаем поле id. Оно имеет тип smallint unsigned (без знаковое маленькое целое), то есть число от 0 до 65,535. Таким образом, наша CMS может содержать до 65,535 статей. Также для него определяется атрибут NOT NULL, который означает, что поле не может быть пустым (null). Данное свойство существенно облегчает труд разработчика. Добавляем атрибут auto\_increment, который указывает MySQL назначать новое, уникальное значение для поля id при создании записи. Итак, первая статья будет иметь id  1, вторая - id  2, и так далее. Мы будем использовать уникальные значения как указатели на статью при выводе и редактировании в CMS.
3. **Добавляем поле publicationDate** Следующая строка создает поле publicationDate, которое хранит дату публикации каждой статьи. Данное поле имеет тип date, соответствующий значениям дат.
4. **Добавляем поле title** Теперь создаем поле title, в котором размещается заголовок. Оно имеет тип varchar(255), то есть может хранить строку длиной до 255 символов.
5. **Добавляем поля summary и content** Последние два поля 2, summary и content, содержат резюме статьи (краткое описание материала) и HTML содержание соответственно. Резюме имеет тип text (то есть, может состоять из 65,535). А поле content имеет тип mediumtext (то есть может содержать до 16,777,215).
6. **Добавляем основной ключ** Последняя строка в выражении CREATE TABLE определяет ключ для таблицы. Ключ также называют индексом, и он служит для быстрого поиска данных в таблице за счет некоторого увеличения требующегося пространства для хранения. Мы определяем поле id как PRIMARY KEY. Каждая таблица может содержать единственный PRIMARY KEY, так как данный ключ уникально определяет каждую запись в таблице. Кроме того, с помощью данного ключа MySQL очень быстро находит нужную запись.

Теперь у нас есть схема таблицы и ее нужно загрузить в MySQL для создания структуры. Самый простой способ - открыть окно терминала, перейти к папке с файлом tables.sql и запустить следующую команду:

mysql -u username -p cms < tables.sql

...где username - имя пользователя MySQL, а cms - имя базы данных, которую мы создали на шаге 1.

Вводите пароль пользователя после запроса, и MySQL загрузит и выполнит код из файла tables.sql, создав таблицу articles в базе данных cms.

Также можно воспользоваться инструментами для администрирования баз данных, таким как phpMyAdmin, cPanel или Plesk (если они установлены на вашем сервере).

## Шаг 3. Создаем файл конфигурации

Теперь  у нас есть база данных и мы готовы разрабатывать код PHP. Начнем с создания файла конфигурации для хранения различных установок для нашей CMS. Данный файл будет использоваться остальными скриптами нашей системы.

Первым делом создаем папку cms в папке веб сервера. Она будет содержать все файлы нашей CMS.

В папке cms создаем файл config.php и копируем в него следующий код:

|  |  |
| --- | --- |
| 01 | <?php |
| 02 | ini\_set( "display\_errors", true ); | |

|  |  |  |
| --- | --- | --- |
| 03 | date\_default\_timezone\_set( "Australia/Sydney" );  // <http://www.php.net/manual/en/timezones.php> | |
| 04 | define( "DB\_DSN", "mysql:host=localhost;dbname=cms" ); |

|  |  |
| --- | --- |
| 05 | define( "DB\_USERNAME", "username" ); |
| 06 | define( "DB\_PASSWORD", "password" ); |

|  |  |
| --- | --- |
| 07 | define( "CLASS\_PATH", "classes" ); |
| 08 | define( "TEMPLATE\_PATH", "templates" ); | |

|  |  |  |
| --- | --- | --- |
| 09 | define( "HOMEPAGE\_NUM\_ARTICLES", 5 ); | |
| 10 | define( "ADMIN\_USERNAME", "admin" ); |

|  |  |
| --- | --- |
| 11 | define( "ADMIN\_PASSWORD", "mypass" ); |
| 12 | require( CLASS\_PATH . "/Article.php" ); | |

|  |  |
| --- | --- |
| 13 |  |
| 14 | function handleException( $exception ) { | |

|  |  |  |
| --- | --- | --- |
| 15 | echo "Sorry, a problem occurred. Please try later."; | |
| 16 | error\_log( $exception->getMessage() ); |

|  |  |  |
| --- | --- | --- |
| 17 | } | |
| 18 |  |

|  |  |  |
| --- | --- | --- |
| 19 | set\_exception\_handler( 'handleException' ); | |
| 20 | ?> |

Разберем код подробно:

1. **Выводим ошибки в браузере** Строка ini\_set() устанавливает режим вывода сообщений об ошибках в браузере. Отличная опция для отладки кода, но на готовом проекте данную опцию надо отключить ( установить значение false) для безопасности ресурса.
2. **Устанавливаем временную зону** Так как наша CMS будет использовать функцию PHP date(), нужно указать временную зону сервера для PHP (иначе PHP будет генерировать предупреждение). В примере установлена зона "Australia/Sydney" — [поменяйте на свою](http://www.php.net/manual/ru/timezones.php).
3. **Устанавливаем детали доступа к базе данных** Затем определяем константу DB\_DSN, которая указывает PHP, где искать базу данных MySQL. Параметр dbname должен соответствовать имени базы данных нашей CMS (cms). Также мы будем хранить имя пользователя MySQL и пароль, которые используются для доступа к базе данных CMS в константах DB\_USERNAME и DB\_PASSWORD. Установите правильные значения в данных константах, которые соответствуют вашим настройкам.
4. **Устанавливаем пути** Мы устанавливаем 2 пути в нашем файле конфигураций: CLASS\_PATH, который указывает на место хранения файлов классов, и TEMPLATE\_PATH, который указывает на место хранения шаблонов  HTML. Оба пути указываются относительно верхнего каталога cms.
5. **Устанавливаем количество статей, выводимых на главной странице** HOMEPAGE\_NUM\_ARTICLES управляет максимальным количеством заголовков статей, которые выводятся на главной странице. Мы установили 5, но можно легко увеличить или уменьшить значение.
6. **Устанавливаем имя и пароль администратора** Константы ADMIN\_USERNAME и ADMIN\_PASSWORD содержат данные регистрации для администратора нашей CMS.
7. **Включаем класс Article** Так как файл класса Article (мы его создадим позже) требуется во всех скриптах нашего приложения, добавим его здесь.
8. **Создаем обработчик исключительных ситуаций** В завершение определяем handleException() - простую функцию для обработки [исключений PHP](http://www.php.net/manual/ru/language.exceptions.php), которые могут генерироваться при выполнении кода. Данная функция выводит общее сообщение об ошибке и записывает данные об ошибке в журнал веб сервера. Такая функция способствует улучшению безопасности системы за счет обработки исключений PDO, которые могут выводить имя пользователя и пароль на странице. После определения функции handleException(), мы устанавливаем ее как обработчик исключений PHP, вызывая функцию set\_exception\_handler().

Такой обработчик исключений сделан для упрощения материалов урока. "Правильный" способ для обработки исключений для перехвата всех вызовов PDO в  Article.php заключается в использовании [блоков try ... catch](http://www.php.net/manual/ru/language.exceptions.php).

### Замечание о безопасности

В реальных проектах лучше помещать config.php где-нибудь за пределами корневого каталога веб сайта, так как в файле содержатся имена и пароли. Обычно код PHP невозможно просмотреть в браузере, но иногда из-за неправильной конфигурации веб сервера код становится доступным. Вы можете использовать функцию [hash()](http://www.php.net/manual/ru/function.hash.php) для хэширования паролей и хранить в config.php хэши, вместо текстовых паролей. Затем при регистрации пользователя можно опять воспользоваться функцией hash() для кодирования введенного пароля и сравнения результата с сохраненным в  config.php хэшем.

[В следующем уроке](http://ruseller.com/lessons.php?rub=37&id=1583) мы построим основной класс нашего приложения - Article.

[В предыдущем уроке серии](http://ruseller.com/lessons.php?rub=37&id=1582) мы создали базу данных и конфигурационный файл для нашей простой CMS. Теперь создадим основной класс нашего приложения - Article.

В нашей CMS  Article будет единственным классом PHP. Он будет обслуживать задачи сохранения статьи в базе данных и получения материалов для вывода на страницах проекта. Как только мы построим данный класс будет действительно легко создать другие скрипты для создания, обновления, вывода и удаления статей.

В нашей папке cms создаем каталог classes. В папке classes создаем новый файл с именем Article.php и копируем в него следующий код:

<?php

/\*\*

\* Класс для обработки статей

\*/

class Article

{

// Свойства

/\*\*

\* @var int ID статей из базы данных

\*/

public $id = null;

/\*\*

\* @var int Дата первой публикации статьи

\*/

public $publicationDate = null;

/\*\*

\* @var string Полное название статьи

\*/

public $title = null;

/\*\*

\* @var string Краткое описание статьи

\*/

public $summary = null;

/\*\*

\* @var string HTML содержание статьи

\*/

public $content = null;

/\*\*

\* Устанавливаем свойства с помощью значений в заданном массиве

\*

\* @param assoc Значения свойств

\*/

public function \_\_construct( $data=array() ) {

if ( isset( $data['id'] ) ) $this->id = (int) $data['id'];

if ( isset( $data['publicationDate'] ) ) $this->publicationDate = (int) $data['publicationDate'];

if ( isset( $data['title'] ) ) $this->title = preg\_replace ( "/[^\.\,\-\\_\'\"\@\?\!\:\$ a-zA-Z0-9()]/", "", $data['title'] );

if ( isset( $data['summary'] ) ) $this->summary = preg\_replace ( "/[^\.\,\-\\_\'\"\@\?\!\:\$ a-zA-Z0-9()]/", "", $data['summary'] );

if ( isset( $data['content'] ) ) $this->content = $data['content'];

}

/\*\*

\* Устанавливаем свойств с помощью значений формы редактирования записи в заданном массиве

\*

\* @param assoc Значения записи формы

\*/

public function storeFormValues ( $params ) {

// Сохраняем все параметры

$this->\_\_construct( $params );

// Разбираем и сохраняем дату публикации

if ( isset($params['publicationDate']) ) {

$publicationDate = explode ( '-', $params['publicationDate'] );

if ( count($publicationDate) == 3 ) {

list ( $y, $m, $d ) = $publicationDate;

$this->publicationDate = mktime ( 0, 0, 0, $m, $d, $y );

}

}

}

/\*\*

\* Возвращаем объект статьи соответствующий заданному ID статьи

\*

\* @param int ID статьи

\* @return Article|false Объект статьи или false, если запись не найдена или возникли проблемы

\*/

public static function getById( $id ) {

$conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD );

$sql = "SELECT \*, UNIX\_TIMESTAMP(publicationDate) AS publicationDate FROM articles WHERE id = :id";

$st = $conn->prepare( $sql );

$st->bindValue( ":id", $id, PDO::PARAM\_INT );

$st->execute();

$row = $st->fetch();

$conn = null;

if ( $row ) return new Article( $row );

}

/\*\*

\* Возвращает все (или диапазон) объектов статей в базе данных

\*

\* @param int Optional Количество строк (по умолчанию все)

\* @param string Optional Столбец по которому производится сортировка статей (по умолчанию "publicationDate DESC")

\* @return Array|false Двух элементный массив: results => массив, список объектов статей; totalRows => общее количество статей

\*/

public static function getList( $numRows=1000000, $order="publicationDate DESC" ) {

$conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD );

$sql = "SELECT SQL\_CALC\_FOUND\_ROWS \*, UNIX\_TIMESTAMP(publicationDate) AS publicationDate FROM articles

ORDER BY " . mysql\_escape\_string($order) . " LIMIT :numRows";

$st = $conn->prepare( $sql );

$st->bindValue( ":numRows", $numRows, PDO::PARAM\_INT );

$st->execute();

$list = array();

while ( $row = $st->fetch() ) {

$article = new Article( $row );

$list[] = $article;

}

// Получаем общее количество статей, которые соответствуют критерию

$sql = "SELECT FOUND\_ROWS() AS totalRows";

$totalRows = $conn->query( $sql )->fetch();

$conn = null;

return ( array ( "results" => $list, "totalRows" => $totalRows[0] ) );

}

/\*\*

\* Вставляем текущий объект статьи в базу данных, устанавливаем его свойства.

\*/

public function insert() {

// Есть у объекта статьи ID?

if ( !is\_null( $this->id ) ) trigger\_error ( "Article::insert(): Attempt to insert an Article object that already has its ID property set (to $this->id).", E\_USER\_ERROR );

// Вставляем статью

$conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD );

$sql = "INSERT INTO articles ( publicationDate, title, summary, content ) VALUES ( FROM\_UNIXTIME(:publicationDate), :title, :summary, :content )";

$st = $conn->prepare ( $sql );

$st->bindValue( ":publicationDate", $this->publicationDate, PDO::PARAM\_INT );

$st->bindValue( ":title", $this->title, PDO::PARAM\_STR );

$st->bindValue( ":summary", $this->summary, PDO::PARAM\_STR );

$st->bindValue( ":content", $this->content, PDO::PARAM\_STR );

$st->execute();

$this->id = $conn->lastInsertId();

$conn = null;

}

/\*\*

\* Обновляем текущий объект статьи в базе данных

\*/

public function update() {

// Есть ли у объекта статьи ID?

if ( is\_null( $this->id ) ) trigger\_error ( "Article::update(): Attempt to update an Article object that does not have its ID property set.", E\_USER\_ERROR );

// Обновляем статью

$conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD );

$sql = "UPDATE articles SET publicationDate=FROM\_UNIXTIME(:publicationDate), title=:title, summary=:summary, content=:content WHERE id = :id";

$st = $conn->prepare ( $sql );

$st->bindValue( ":publicationDate", $this->publicationDate, PDO::PARAM\_INT );

$st->bindValue( ":title", $this->title, PDO::PARAM\_STR );

$st->bindValue( ":summary", $this->summary, PDO::PARAM\_STR );

$st->bindValue( ":content", $this->content, PDO::PARAM\_STR );

$st->bindValue( ":id", $this->id, PDO::PARAM\_INT );

$st->execute();

$conn = null;

}

/\*\*

\* Удаляем текущий объект статьи из базы данных

\*/

public function delete() {

// Есть ли у объекта статьи ID?

if ( is\_null( $this->id ) ) trigger\_error ( "Article::delete(): Attempt to delete an Article object that does not have its ID property set.", E\_USER\_ERROR );

// Удаляем статью

$conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD );

$st = $conn->prepare ( "DELETE FROM articles WHERE id = :id LIMIT 1" );

$st->bindValue( ":id", $this->id, PDO::PARAM\_INT );

$st->execute();

$conn = null;

}

}

?>

Файл получается достаточно длинным, но код очень простой. Разберем его подробно:

**1. Определение класса и его свойства**

Сначала определим класс Article:

class Article

{

Все, что следует за данным строками до закрывающей фигурной скобки в конце файла содержит код нашего класса  Article.

После определения класса мы объявляем свойства класса : $id, $publicationDate и так далее. Каждый объект Article, который мы создаем, будет хранить данные в объявленных свойствах. Обратите внимание, что имена свойств соответствуют именам полей в таблице articles.

Технически, такой тип класса, который содержит свойства соответствующие непосредственно полям базы данных и методы для хранения и получения записей, соответствует шаблону объектно-ориентированного проектирования, известному как  [active record](http://ru.wikipedia.org/wiki/ActiveRecord).

**2. Конструктор**

Затем мы создаем методы класса. Это функции, которые привязаны к классу и к объекту, создаваемому из класса. Наш основной код вызывает методы для манипулирования данными в объекте Article.

Первый метод, \_\_construct(), является конструктором. Это специальный метод, который автоматически вызывается системой PHP каждый раз, когда создается новый объект Article. Наш конструктор получает необязательный массив $data, в котором содержатся данные для свойств нового объекта. Затем мы присваиваем данные свойствам в теле конструктора. Таким образом, получается удобный способ для создания и инициализации объекта в одно действие.

$this->propertyName означает: "Свойство объекта this с именем "*$propertyName*".

Обратите внимание, что метод фильтрует данные, прежде чем присвоить их свойствам. Свойства id и publicationDate приводятся к типу int с помощью (int), так данные значения должны быть типа int. Свойства title и summary фильтруются с помощью регулярных выражений, так как в них допускает наличие символов из определенного набора. С точки зрения безопасности фильтрация данных ввода - отличная практика. Пропускаем только допустимые значения и символы.

Однако, мы не фильтруем свойство content. Почему? Вероятно, администратор захочет использовать более широкий диапазон символов в содержании статьи - например, разметку HTML. Если мы ограничим диапазон доступных символов в содержании, то снизим полезность нашей системы для администратора.

Обычно, такие места могут оказаться дырой в системе безопасности, так как пользователь может вставить вредный код JavaScript или материалы с ошибками в содержание статьи. Однако, так как мы полагаем. что единственной персоной, которой доступно редактирование содержание, будет администратор системы, располагающий безграничным доверием, то вопрос с уязвимостью содержания остается за рамками нашего внимания. Если вы имеете дело с генерируем пользователями содержанием, например, комментариями или записями форума, то следует быть более осторожным и допускать только "безопасный" код HTML к использованию. Отличным инструментом для решения таких задач является  [HTML Purifier](http://htmlpurifier.org/), который анализирует ввод кода HTML и удаляет все потенциальные угрозы.

Безопасность кода PHP выходит за рамки наших уроков. Вам следует посвятить определенное время для [изучения данного вопроса](http://php.net/manual/ru/security.php).

**3. storeFormValues()**

Следующий метод storeFormValues()похож на конструктор в том, что он сохраняет полученный массив данных в свойствах объекта. Основное отличие заключается в том, что storeFormValues() может обрабатывать данные в формате, который используется в формах New Article (Новая статья) и Edit Article (Редактировать статью) (мы создадим их позже). В частности, он может обрабатывать дату публикации в формате  YYYY-MM-DD, конвертировать ее в формат времени UNIX, который отлично подходит для хранения в объекте.

Формат времени UNIX представляет собой целое значение, которое содержит количество секунд от полуночи 1 января 1970 до искомой даты. Датой в таком формате легко оперировать, и ее удобно хранить.

Назначение данного метода - облегчить реализацию скрипта для хранения дат, вводимых в формах.

Все члены (то есть свойства и методы) нашего класса Article имеют ключевое слово public перед определением, что означает доступность кода вне класса. Также можно создавать частные члены (директива private) (их можно использовать только в классе) и защищенные члены (директива protected) (которые можно использовать в классе и его подклассах).

**4. getById()**

Теперь перейдем к методам, реализующим доступ к базе данных MySQL. Первый из них - getById(). Он принимает в качестве аргумента ID статьи  ($id) и возвращает запись с указанным ID из таблицы articles, сохраняя данные в новом объекте Article.

Обычно, когда вы вызываете метод, сначала нужно создать объект, а затем вызвать метод, принадлежащий объекту. Но, так как  getById() возвращает новый объект  Article, будет полезно вызывать его напрямую, а не через существующий объект. Иначе придется создавать новый объект-заглушку каждый раз, когда нужно вызвать метод и получить статью.

Для разрешения вызова метода без объекта  мы добавляем декларацию [static](http://www.php.net/manual/ru/language.oop5.static.php) к определению метода. Таким образом разрешается вызов метода непосредственно без определения объекта.

public static function getById( $id ) {

Метод использует PDO для соединения с базой данных, получает запись статьи с помощью запроса SQL SELECT и сохраняет данные в новом объекте Article, который возвращается в вызывающий код. [PDO](http://www.php.net/manual/ru/book.pdo.php) — PHP Data Objects —объектно-ориентированная библиотека, встроенная в PHP, которая облегчает связь скриптов PHP с базами данных.

Разберем метод подробнее:

1. **Соединение с базой данных**
2. $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD );

Здесь выполняется соединение с базой данных MySQL с помощью данных из файла config.php. Дескриптор соединения сохраняется в переменной $conn. Данный дескриптор используется в остальном коде для обмена данных с базой.

1. **Получаем запись статьи**
2. $sql = "SELECT \*, UNIX\_TIMESTAMP(publicationDate) AS publicationDate FROM articles WHERE id = :id";
3. $st = $conn->prepare( $sql );
4. $st->bindValue( ":id", $id, PDO::PARAM\_INT );
5. $st->execute();
6. $row = $st->fetch();

Выражение SELECT возвращает все поля (\*) из записи в таблице articles, которые соответствуют заданному полю id. Значение поля publicationDate возвращается в формате времени UNIX, вместо формата для дат MySQL, что упрощает процесс сохранения в нашем объекте.

Вместо того, чтобы помещать наш параметр $id непосредственно в строку SELECT, что увеличивает риск нарушения системы безопасности, мы используем :id. Такой параметр известен как *placeholder (указатель места размещения)*. Далее мы вызываем метод PDO для привязывания значение $id к указателю места размещения.

Сразу после сохранения выражения SELECT в строке, мы подготавливаем его с помощью функции $conn->prepare(), сохраняя полученный дескриптор в переменной  $st.

Подготовка выражения используется для работы со многими базами данных. Она позволяет выполнять запросы быстрее и безопаснее.

Затем мы привязываем значение переменной $id ( ID нужной статьи) к указателю места размещения :id с помощью вызова метода bindValue().

И вызываем метод execute() для выполнения запроса. После чего используем метод fetch() для перемещения полученной записи в ассоциированный массив с именами полей и соответствующими значениями, который хранится в переменной $row.

1. **Закрываем соединение**
2. $conn = null;

Так как нам больше не нужно соединение, мы закрываем его, присваивая значение null переменной $conn. Закрывать соединение с базой данных как можно быстрее является хорошей практикой для освобождения памяти на сервере.

1. **Возвращаем объект Article**
2. if ( $row ) return new Article( $row );
3. }

Последним действием в нашем методе является создание объекта  Article, который будет содержать запись из базы данных, и возвращение его вызывающему коду. Сначала проверяем наличие данных в переменной $row после вызова метода fetch(). Если данные есть, создаем новый объект Article передавая переменную ему $row. Будет вызван конструктор класса, который наполнит объект данными из массива $row. Затем возвращаем готовый объект и работа метода завершена.

**5. getList()**

Следующий метод getList() во многом похож на метод getById(). Основное отличие заключается в том, что метод getList() возвращает несколько статей сразу. Его используют, когда нужно вывести список статей для пользователя или администратора.

getList() принимает 2 аргумента:

$numRows

Максимальное количество получаемых статей. По умолчанию установлено значение 1,000,000 (то есть, практически все статьи). Данный параметр позволяет нам получать только первые 5 статей для главной страницы.

$order

Порядок сортировки получаемых статей. По умолчанию используется параметр "publicationDate DESC", который означает "сортировка по дате публикации, новые статьи первые".

Большая часть кода метода похожа на код метода getById(). Посмотрим на несколько строк:

$sql = "SELECT SQL\_CALC\_FOUND\_ROWS \*, UNIX\_TIMESTAMP(publicationDate) AS publicationDate FROM articles

ORDER BY " . mysql\_escape\_string($order) . " LIMIT :numRows";

Здесь запрос немного сложнее. Обратите внимание, что здесь не используется выражение WHERE, так как мы хотим получить все статьи, а не одну с заданным ID.

Также добавлено выражение ORDER BY для сортировки возвращаемых записей в определенном порядке. Используется выражение LIMIT с параметром $numRows  (как указатель места размещения) для ограничения количества получаемых записей.

Специальное значение MySQL SQL\_CALC\_FOUND\_ROWS указывает базе данных, что нужно вернуть действительное количество возвращаемых записей. Такая информация полезна для информирования пользователя и организации других функций, например, постраничного вывода списка.

Вместо передачи значения переменной $order в запрос через указатель места размещения, мы передаем его прямо в строку запроса, вызывая функцию  mysql\_escape\_string(), чтобы отбросить любые специальные символы (для безопасности). Если использовать указатель места размещения, то PDO поместит кавычки (') вокруг строки (например, ORDER BY 'publicationDate DESC'), что является ошибкой синтаксиса.

$list = array();

while ( $row = $st->fetch() ) {

$article = new Article( $row );

$list[] = $article;

}

Так как мы возвращаем несколько строк, нужно создать массив $list для размещения соответствующих объектов Article. Затем используем цикл while для получения следующей строки через fetch(), создаем новый объект Article, сохраняем строку в объекте и добавляем объект к массиву $list. Когда строк не останется, метод fetch() вернет false, и цикл остановится.

// Теперь получаем общее число статей, которые соответствуют критерию

$sql = "SELECT FOUND\_ROWS() AS totalRows";

$totalRows = $conn->query( $sql )->fetch();

$conn = null;

return ( array ( "results" => $list, "totalRows" => $totalRows[0] ) );

В завершении мы выполняем запрос, который использует функцию MySQL FOUND\_ROWS() для получения количества возвращаемых строк, вычисленного в предыдущей команде SQL\_CALC\_FOUND\_ROWS. В этот раз используем метод PDO query(), который позволяет быстро выполнить запрос, если нет указателей места замещения. Мы вызываем метод fetch() для получения результата. Затем возвращаем оба значения - список объектов Article ($list) и общее количество строк - как ассоциированный массив.

**6. insert()**

Оставшиеся методы в нашем классе Article работают с добавлением, изменением и удалением записей статей в базе данных.

insert() добавляет новую статью в таблицу articles, используя значения из текущего объекта Article:

* Сначала метод проверяет, что объект не имеет установленного свойства $id. Если у объекта есть ID, то, вероятно, статья уже имеется в базе данных и ее добавлять не нужно.
* Затем метод выполняет запрос SQL INSERT для вставки записи в таблицу articles, используя указатели места замещения для передачи значений свойств в базу данных. Обратите внимание, что мы используем функцию MySQL FROM\_UNIXTIME() для конвертации даты публикации в формат MySQL.
* После выполнения запроса, метод возвращает ID новой статьи с помощью функции PDO lastInsertId() и сохраняет значение в свойстве $id. Мы установили в таблице articles для поля id свойство auto\_increment, поэтому MySQL генерирует уникальное значение ID для каждой новой записи.

Обратите внимание, что мы используем PDO::PARAM\_INT при привязке целых значений к указателям места замещения, и PDO::PARAM\_STR при привязке строк. Таким образом, PDO может правильно обрабатывать значения.

**7. update ()**

Данный метод похож на метод insert(), за исключением того, что здесь происходит обновление записи в базе данных вместо создания новой записи.

Сначала проверяем наличие ID у объекта, так как обновить можно только запись с известным ID. Затем используем выражение SQL UPDATE для обновления полей записи. Обратите внимание на передачу ID объекта в выражение UPDATE, так как мы знаем, какую запись надо обновить.

**8. delete ()**

Метод delete() использует выражение SQL DELETE для удаления из таблицы articles статьи, которая хранится в объекте. Для идентификации записи задействуем свойство $id объекта. Для безопасности мы добавили LIMIT 1 к запросу, чтобы ограничиться удалением только одной записи.

[В следующем уроке](http://ruseller.com/lessons.php?rub=37&id=1585) мы создадим скрипты для клиентской и серверной части нашего приложения.

## 1. Клиентская часть

Сначала создадим файл index.php, который будет содержать скрипт клиентской части, то есть выводить страницы в браузере пользователя. Копируем ниже приведенный код и сохраняем файл в корневом каталоге нашей CMS cms.

|  |  |  |
| --- | --- | --- |
| 01 | <?php | |
| 02 |  |

|  |  |
| --- | --- |
| 03 | require( "config.php" ); |
| 04 | $action = isset( $\_GET['action'] ) ? $\_GET['action'] : ""; | |

|  |  |
| --- | --- |
| 05 |  |
| 06 | switch ( $action ) { | |

|  |  |  |
| --- | --- | --- |
| 07 | case 'archive': | |
| 08 | archive(); |

|  |  |
| --- | --- |
| 09 | break; |
| 10 | case 'viewArticle': | |

|  |  |  |
| --- | --- | --- |
| 11 | viewArticle(); | |
| 12 | break; |

|  |  |
| --- | --- |
| 13 | default: |
| 14 | homepage(); | |

|  |  |
| --- | --- |
| 15 | } |
| 16 |  | |

|  |  |
| --- | --- |
| 17 |  |
| 18 | function archive() { | |

|  |  |
| --- | --- |
| 19 | $results = array(); |
| 20 | $data = Article::getList(); | |

|  |  |
| --- | --- |
| 21 | $results['articles'] = $data['results']; |
| 22 | $results['totalRows'] = $data['totalRows']; | |

|  |  |  |
| --- | --- | --- |
| 23 | $results['pageTitle'] = "Article Archive | Widget News"; | |
| 24 | require( TEMPLATE\_PATH . "/archive.php" ); |

|  |  |
| --- | --- |
| 25 | } |
| 26 |  | |

|  |  |
| --- | --- |
| 27 | function viewArticle() { |
| 28 | if ( !isset($\_GET["articleId"]) || !$\_GET["articleId"] ) { | |

|  |  |  |
| --- | --- | --- |
| 29 | homepage(); | |
| 30 | return; |

|  |  |  |
| --- | --- | --- |
| 31 | } | |
| 32 |  |

|  |  |
| --- | --- |
| 33 | $results = array(); |
| 34 | $results['article'] = Article::getById( (int)$\_GET["articleId"] ); | |

|  |  |  |
| --- | --- | --- |
| 35 | $results['pageTitle'] = $results['article']->title . " | Widget News"; | |
| 36 | require( TEMPLATE\_PATH . "/viewArticle.php" ); |

|  |  |
| --- | --- |
| 37 | } |
| 38 |  | |

|  |  |
| --- | --- |
| 39 | function homepage() { |
| 40 | $results = array(); |

|  |  |  |
| --- | --- | --- |
| 41 | $data = Article::getList( HOMEPAGE\_NUM\_ARTICLES ); | |
| 42 | $results['articles'] = $data['results']; |

|  |  |  |
| --- | --- | --- |
| 43 | $results['totalRows'] = $data['totalRows']; | |
| 44 | $results['pageTitle'] = "Widget News"; |

|  |  |  |
| --- | --- | --- |
| 45 | require( TEMPLATE\_PATH . "/homepage.php" ); | |
| 46 | } |

|  |  |
| --- | --- |
| 47 |  |
| 48 | ?> | |

Разберем код скрипта подробно.

### ****Включаем файл конфигурации****

Первая строка кода включает файл config.php, и все установки приложения становятся доступны в скрипте. Мы используем функцию require(), а не  include(), так как  require() генерирует ошибку в случае отсутствия подключаемого файла.

### ****Получаем параметр****action

Сохраняем параметр $\_GET['action'] в переменной $action, чтобы можно было использовать значение далее в скрипте. Но прежде проверяем наличие значения $\_GET['action'] с помощью функции isset(). Если его нет, устанавливаем для переменной $action пустую строку ("").

Проверять значения, поставляемые пользователем ( такие как строки, значения из форм, куки), на наличие перед использованием - хорошая практика программирования.  Так не только ограничивается количество дырок в системе безопасности, но и предотвращает вывод предупреждения "undefined index" при выполнении скрипта.

### ****Определяем действие для выполнения****

Блок switch проверяет параметр action в URL для определения нужного действия (вывести архив, просмотреть статью). Если параметр action отсутствует в URL, то скрипт выводит главную страницу.

### archive()

Данная функция выводит список всех статей в базе данных. Для этого используется метод getList() класса Article . Функция сохраняет результат и заголовок страницы в ассоциированном массиве $results, и шаблон может вывести его на странице. В завершении включается шаблон для вывода страницы. (Мы рассмотрим шаблоны в другом уроке нашей серии.)

### viewArticle()

Данная функция выводит страницу одной статьи. Она получает ID статьи для вывода из параметра URL articleId, затем вызывает метод класса Article  getById() для получения объекта статьи, который сохраняется в массиве $results для использования в шаблоне. (Если нет параметра articleId или статья не может быть найдена, то функция просто выводит главную страницу.)

Обратите внимание, что мы используем преобразование типов (int) для явного приведения параметра запроса articleId к целому типу. Такое действие предотвращает передачу отличных от чисел значений в наш код.

### homepage()

Последняя функция homepage()выводит главную страницу сайта, на которой содержится список из нескольких статей, количество которых указано в параметре конфигурации HOMEPAGE\_NUM\_ARTICLES  (по умолчанию 5 ). Функция очень похожа на функцию archive()  за исключением передачи параметра HOMEPAGE\_NUM\_ARTICLES методу getList() для ограничения количества возвращаемых статей.

## 2. Серверная часть

Скрипт серверной части несколько сложнее, чем index.php, так как в нем реализованы функции администрирования для нашей CMS. Хотя основная структура похожа на index.php.

Создаем файл admin.php в том же каталоге, что и файл index.php. И копируем в него код:

|  |  |  |
| --- | --- | --- |
| 001 | <?php | |
| 002 |  |

|  |  |  |
| --- | --- | --- |
| 003 | require( "config.php" ); | |
| 004 | session\_start(); |

|  |  |
| --- | --- |
| 005 | $action = isset( $\_GET['action'] ) ? $\_GET['action'] : ""; |
| 006 | $username = isset( $\_SESSION['username'] ) ? $\_SESSION['username'] : ""; | |

|  |  |
| --- | --- |
| 007 |  |
| 008 | if ( $action != "login" && $action != "logout" && !$username ) { | |

|  |  |  |
| --- | --- | --- |
| 009 | login(); | |
| 010 | exit; |

|  |  |  |
| --- | --- | --- |
| 011 | } | |
| 012 |  |

|  |  |  |
| --- | --- | --- |
| 013 | switch ( $action ) { | |
| 014 | case 'login': |

|  |  |  |
| --- | --- | --- |
| 015 | login(); | |
| 016 | break; |

|  |  |  |
| --- | --- | --- |
| 017 | case 'logout': | |
| 018 | logout(); |

|  |  |
| --- | --- |
| 019 | break; |
| 020 | case 'newArticle': | |

|  |  |  |
| --- | --- | --- |
| 021 | newArticle(); | |
| 022 | break; |

|  |  |  |
| --- | --- | --- |
| 023 | case 'editArticle': | |
| 024 | editArticle(); |

|  |  |
| --- | --- |
| 025 | break; |
| 026 | case 'deleteArticle': | |

|  |  |  |
| --- | --- | --- |
| 027 | deleteArticle(); | |
| 028 | break; |

|  |  |
| --- | --- |
| 029 | default: |
| 030 | listArticles(); | |

|  |  |  |
| --- | --- | --- |
| 031 | } | |
| 032 |  |

|  |  |
| --- | --- |
| 033 |  |
| 034 | function login() { | |

|  |  |
| --- | --- |
| 035 |  |
| 036 | $results = array(); | |

|  |  |  |
| --- | --- | --- |
| 037 | $results['pageTitle'] = "Admin Login | Widget News"; | |
| 038 |  |

|  |  |  |
| --- | --- | --- |
| 039 | if ( isset( $\_POST['login'] ) ) { | |
| 040 |  |

|  |  |  |
| --- | --- | --- |
| 041 | // Пользователь получает форму входа: попытка авторизировать пользователя | |
| 042 |  |

|  |  |  |
| --- | --- | --- |
| 043 | if ( $\_POST['username'] == ADMIN\_USERNAME && $\_POST['password'] == ADMIN\_PASSWORD ) { | |
| 044 |  |

|  |  |  |
| --- | --- | --- |
| 045 | // Вход прошел успешно: создаем сессию и перенаправляем на страницу администратора | |
| 046 | $\_SESSION['username'] = ADMIN\_USERNAME; |

|  |  |  |
| --- | --- | --- |
| 047 | header( "Location: admin.php" ); | |
| 048 |  |

|  |  |  |
| --- | --- | --- |
| 049 | } else { | |
| 050 |  |

|  |  |
| --- | --- |
| 051 | // Ошибка входа: выводим сообщение об ошибке для пользователя |
| 052 | $results['errorMessage'] = "Incorrect username or password. Please try again."; | |

|  |  |  |
| --- | --- | --- |
| 053 | require( TEMPLATE\_PATH . "/admin/loginForm.php" ); | |
| 054 | } |

|  |  |
| --- | --- |
| 055 |  |
| 056 | } else { | |

|  |  |
| --- | --- |
| 057 |  |
| 058 | // Пользователь еще не получил форму: выводим форму | |

|  |  |  |
| --- | --- | --- |
| 059 | require( TEMPLATE\_PATH . "/admin/loginForm.php" ); | |
| 060 | } |

|  |  |
| --- | --- |
| 061 |  |
| 062 | } | |

|  |  |
| --- | --- |
| 063 |  |
| 064 |  |

|  |  |
| --- | --- |
| 065 | function logout() { |
| 066 | unset( $\_SESSION['username'] ); | |

|  |  |  |
| --- | --- | --- |
| 067 | header( "Location: admin.php" ); | |
| 068 | } |

|  |  |
| --- | --- |
| 069 |  |
| 070 |  |

|  |  |  |
| --- | --- | --- |
| 071 | function newArticle() { | |
| 072 |  |

|  |  |
| --- | --- |
| 073 | $results = array(); |
| 074 | $results['pageTitle'] = "New Article"; | |

|  |  |  |
| --- | --- | --- |
| 075 | $results['formAction'] = "newArticle"; | |
| 076 |  |

|  |  |  |
| --- | --- | --- |
| 077 | if ( isset( $\_POST['saveChanges'] ) ) { | |
| 078 |  |

|  |  |  |
| --- | --- | --- |
| 079 | // Пользователь получает форму редактирования статьи: сохраняем новую статью | |
| 080 | $article = new Article; |

|  |  |  |
| --- | --- | --- |
| 081 | $article->storeFormValues( $\_POST ); | |
| 082 | $article->insert(); |

|  |  |  |
| --- | --- | --- |
| 083 | header( "Location: admin.php?status=changesSaved" ); | |
| 084 |  |

|  |  |  |
| --- | --- | --- |
| 085 | } elseif ( isset( $\_POST['cancel'] ) ) { | |
| 086 |  |

|  |  |  |
| --- | --- | --- |
| 087 | // Пользователь сбросил результаты редактирования: возвращаемся к списку статей | |
| 088 | header( "Location: admin.php" ); |

|  |  |  |
| --- | --- | --- |
| 089 | } else { | |
| 090 |  |

|  |  |  |
| --- | --- | --- |
| 091 | // Пользователь еще не получил форму редактирования: выводим форму | |
| 092 | $results['article'] = new Article; |

|  |  |  |
| --- | --- | --- |
| 093 | require( TEMPLATE\_PATH . "/admin/editArticle.php" ); | |
| 094 | } |

|  |  |
| --- | --- |
| 095 |  |
| 096 | } | |

|  |  |
| --- | --- |
| 097 |  |
| 098 |  |

|  |  |  |
| --- | --- | --- |
| 099 | function editArticle() { | |
| 100 |  |

|  |  |
| --- | --- |
| 101 | $results = array(); |
| 102 | $results['pageTitle'] = "Edit Article"; | |

|  |  |  |
| --- | --- | --- |
| 103 | $results['formAction'] = "editArticle"; | |
| 104 |  |

|  |  |  |
| --- | --- | --- |
| 105 | if ( isset( $\_POST['saveChanges'] ) ) { | |
| 106 |  |

|  |  |  |
| --- | --- | --- |
| 107 | // Пользователь получил форму редактирования статьи: сохраняем изменения | |
| 108 |  |

|  |  |  |
| --- | --- | --- |
| 109 | if ( !$article = Article::getById( (int)$\_POST['articleId'] ) ) { | |
| 110 | header( "Location: admin.php?error=articleNotFound" ); |

|  |  |  |
| --- | --- | --- |
| 111 | return; | |
| 112 | } |

|  |  |
| --- | --- |
| 113 |  |
| 114 | $article->storeFormValues( $\_POST ); | |

|  |  |
| --- | --- |
| 115 | $article->update(); |
| 116 | header( "Location: admin.php?status=changesSaved" ); | |

|  |  |
| --- | --- |
| 117 |  |
| 118 | } elseif ( isset( $\_POST['cancel'] ) ) { | |

|  |  |
| --- | --- |
| 119 |  |
| 120 | // Пользователь отказался от результатов редактирования: возвращаемся к списку статей | |

|  |  |  |
| --- | --- | --- |
| 121 | header( "Location: admin.php" ); | |
| 122 | } else { |

|  |  |
| --- | --- |
| 123 |  |
| 124 | // Пользвоатель еще не получил форму редактирования: выводим форму | |

|  |  |  |
| --- | --- | --- |
| 125 | $results['article'] = Article::getById( (int)$\_GET['articleId'] ); | |
| 126 | require( TEMPLATE\_PATH . "/admin/editArticle.php" ); |

|  |  |  |
| --- | --- | --- |
| 127 | } | |
| 128 |  |

|  |  |  |
| --- | --- | --- |
| 129 | } | |
| 130 |  |

|  |  |
| --- | --- |
| 131 |  |
| 132 | function deleteArticle() { | |

|  |  |
| --- | --- |
| 133 |  |
| 134 | if ( !$article = Article::getById( (int)$\_GET['articleId'] ) ) { | |

|  |  |  |
| --- | --- | --- |
| 135 | header( "Location: admin.php?error=articleNotFound" ); | |
| 136 | return; |

|  |  |  |
| --- | --- | --- |
| 137 | } | |
| 138 |  |

|  |  |
| --- | --- |
| 139 | $article->delete(); |
| 140 | header( "Location: admin.php?status=articleDeleted" ); | |

|  |  |  |
| --- | --- | --- |
| 141 | } | |
| 142 |  |

|  |  |
| --- | --- |
| 143 |  |
| 144 | function listArticles() { | |

|  |  |
| --- | --- |
| 145 | $results = array(); |
| 146 | $data = Article::getList(); | |

|  |  |
| --- | --- |
| 147 | $results['articles'] = $data['results']; |
| 148 | $results['totalRows'] = $data['totalRows']; | |

|  |  |  |
| --- | --- | --- |
| 149 | $results['pageTitle'] = "All Articles"; | |
| 150 |  |

|  |  |
| --- | --- |
| 151 | if ( isset( $\_GET['error'] ) ) { |
| 152 | if ( $\_GET['error'] == "articleNotFound" ) $results['errorMessage'] = "Error: Article not found."; | |

|  |  |  |
| --- | --- | --- |
| 153 | } | |
| 154 |  |

|  |  |
| --- | --- |
| 155 | if ( isset( $\_GET['status'] ) ) { |
| 156 | if ( $\_GET['status'] == "changesSaved" ) $results['statusMessage'] = "Your changes have been saved."; | |

|  |  |  |
| --- | --- | --- |
| 157 | if ( $\_GET['status'] == "articleDeleted" ) $results['statusMessage'] = "Article deleted."; | |
| 158 | } |

|  |  |
| --- | --- |
| 159 |  |
| 160 | require( TEMPLATE\_PATH . "/admin/listArticles.php" ); | |

|  |  |  |
| --- | --- | --- |
| 161 | } | |
| 162 |  |

|  |  |
| --- | --- |
| 163 | ?> |

Рассмотрим интересные места нашего скрипта:

### ****Старт сессии пользователя****

В начале скрипта мы вызываем функцию session\_start(). Данная функция PHP запускает новую сессию пользователя, которая позволяет нам контролировать регистрацию пользователя в системе.  Если сессия для пользователя уже имеется, то PHP автоматически возобновит ее и будет использовать.

Так как для сессии требуется работа с куки, и куки пересылаются в браузер перед содержанием, то следует вызывать session\_start() в начале скрипта, до отправки любого содержания.

### ****Получаем параметр**** action ****и переменную сессии**** username

Затем мы сохраняем параметр $\_GET['action'] в переменной $action, а переменную сессии $\_SESSION['username'] в $username, чтобы затем использовать данные значения в скрипте. Перед сохранение осуществляем проверку с помощью функции isset(). Если значение не существует, устанавливаем в соответствующей переменной пустую строку ("").

### ****Проверяем, зарегистрирован ли пользователь****

Нельзя позволять пользователю совершать какие-либо операции, пока он не зарегистрируется как администратор. Поэтому нужно проверить $username на наличие в сессии значения для ключа username, который будет использоваться для подтверждения регистрации пользователя. Если значение $username пустое, то показываем страницу регистрации и выходим немедленно.

### ****Определяем действие для выполнения****

Блок switch работает так же как и в index.php: производится вызов соответствующей функции на основании значения параметра URL action. По умолчанию выводится список статей.

### login()

Функция вызывается, когда нужно произвести регистрацию пользователя в системе.

Если пользователь отправил форму регистрации (что проверяется по параметру формы login ), то функция сопоставляет введенные имя и пароль со значениями параметров конфигурации ADMIN\_USERNAME и ADMIN\_PASSWORD. Если они соответствуют, то мы устанавливаем ключ сессии username в значение имени администратора, а затем перенаправляем обратно скрипту admin.php, который выводит список статей. Если пароль и имя пользователя неверные, то будет выведена форма регистрации с сообщением об ошибке.

Если пользователь еще не отправил форму регистрации, то просто выводим ее.

### logout()

Данная функция вызывается, когда пользователь выходит из системы. Здесь просто удаляется ключ сессии username и производится перенаправление обратно в admin.php.

### newArticle()

Данная функция позволяет пользователю создавать новую статью. Если пользователь уже отправил форму для новой статьи, то функция создает новый объект Article, сохраняет данные формы в объекте с помощью вызова функции storeFormValues(), вставляет статью в базу данных с помощью функции insert() и перенаправляет обратно на список статей, выводя сообщение об успешном завершении операции.

Если пользователь не отправил форму для новой статьи, то функция создает новый пустой объект  Article без значений, затем использует шаблон editArticle.php для вывода формы редактирования статьи для только что созданного объекта.

### editArticle()

Данная функция похожа на newArticle(), за исключением того, что пользователю предоставляется возможность редактировать статью. Когда пользователь сохраняет изменения, функция получает существующую статью с помощью getById(), записывает новые значения в объекте Article, затем сохраняет измененный объект с помощью функции update(). (Если статья не найдена в базе данных, функция выведет сообщение об ошибке.)

Когда выводится форма редактирования статьи, функция использует метод getById() для загрузки текущих значений в поля формы.

Скрипт использует один и тот же шаблон (editArticle.php) и для создания новой статьи и для редактирования имеющейся. То есть нужно создавать только одну форму HTML. Параметр formAction используется для определения операции со статьей.

### deleteArticle()

Когда пользователь удаляет статью, данная функция сначала получает ее (если статьи нет в базе данных, то выводится сообщение об ошибке), а затем вызывает метод delete() для удаления данных из базы. После завершения операции функция перенаправляет пользователя на страницу со списком статей и выводит сообщение о удалении.

### listArticles()

Последняя функция функция в скрипте admin.php выводит список статей в CMS. Используется метод getList() класса Article для получения всех статей. Затем применяем шаблон listArticles.php для вывода списка. Кроме того, функция проверяет параметры запроса URL error и status, чтобы вывести сообщения об ошибке и состоянии системы, если нужно.

[В следующем уроке](http://ruseller.com/lessons.php?rub=37&id=1590) мы создадим шаблоны для клиентской и серверной части.

## 1. Клиентская часть

### Файлы включения

Создаем папку с именем templates в папке cms нашего проекта. Затем создаем папку с именем include в папке templates. В последней папке мы поместим разметку для заголовка и нижнего колонтитула, которые будут одинаковы для всех страниц сайта. Поэтому их можно включать в каждый шаблон нашего проекта.

Создаем файл с именем header.php в папке include и копируем в него следующий код:

<!DOCTYPE html>

<html lang="en">

<head>

<title><?php echo htmlspecialchars( $results['pageTitle'] )?></title>

<link rel="stylesheet" type="text/css" href="style.css" />

</head>

<body>

<div id="container">

<a href="."><img id="logo" src="images/logo.jpg" alt="Widget News" /></a>

Данный код просто выводит разметку начала страницы HTML. Здесь используется переменная $results['pageTitle'], передаваемая из основного скрипта (index.php или admin.php), для установки элемента title, а также присоединяется таблица стилей style.css (мы создадим ее чуть позже).

Обратите внимание, что значение переменной $results['pageTitle'] передается через функцию [htmlspecialchars()](http://php.net/manual/ru/function.htmlspecialchars.php). Данная функция кодирует любые специальные символы HTML (такие как <, > или &) в эквиваленты (&lt;, &gt; и &amp; соответственно). Вместе с фильтрацией ввода, которую мы разбирали в предыдущих уроках серии при создании конструктора класса Article кодирование вывода является хорошей привычкой для повышения безопасности сайта. Мы будем кодировать практически все данные в нашем шаблоне подобным образом.

Теперь создаем файл с именем footer.php в той же папке:

|  |  |
| --- | --- |
| 1 | <div id="footer"> |
| 2 | Widget News &copy; 2011. All rights reserved. <a href="admin.php">Site Admin</a> | |

|  |  |  |
| --- | --- | --- |
| 3 | </div> | |
| 4 |  |

|  |  |  |
| --- | --- | --- |
| 5 | </div> | |
| 6 | </body> |

|  |  |
| --- | --- |
| 7 | </html> |

Данная разметка завершает каждую HTML страницу в нашей системе.

### homepage.php

Вернемся в папку templates и создадим в ней файл homepage.php, в который копируем следующий код:

|  |  |  |
| --- | --- | --- |
| 01 | <?php include "templates/include/header.php" ?> | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | <ul id="headlines"> | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | <?php foreach ( $results['articles'] as $article ) { ?> | |
| 06 |  |

|  |  |
| --- | --- |
| 07 | <li> |
| 08 | <h2> | |

|  |  |  |
| --- | --- | --- |
| 09 | <span class="pubDate"><?php echo date('j F', $article->publicationDate)?></span><a href=".?action=viewArticle&amp;articleId=<?php echo $article->id?>"><?php echo htmlspecialchars( $article->title )?></a> | |
| 10 | </h2> |

|  |  |  |
| --- | --- | --- |
| 11 | <p class="summary"><?php echo htmlspecialchars( $article->summary )?></p> | |
| 12 | </li> |

|  |  |
| --- | --- |
| 13 |  |
| 14 | <?php } ?> | |

|  |  |
| --- | --- |
| 15 |  |
| 16 | </ul> | |

|  |  |
| --- | --- |
| 17 |  |
| 18 | <p><a href="./?action=archive">Article Archive</a></p> | |

|  |  |
| --- | --- |
| 19 |  |
| 20 | <?php include "templates/include/footer.php" ?> | |

Данный шаблон выводит заголовки статей на главной странице в виде неупорядоченного списка. Скрипт проходит циклом по массиву объектов Article, который хранится в $results['articles'], и выводит для каждой статьи дату публикации, название и резюме. Название ссылается на '.' (index.php), передавая параметр  action=viewArticle вместе с ID статьи в URL. Таким образом, пользователь может прочитать статью, нажав на ссылку названия.

Также шаблон включает ссылку на архив статей ("./?action=archive").

Обратите внимание, что данный шаблон, как и последующие, использует директиву PHP include для включения начала страницы и нижнего колонтитула.

### archive.php

Теперь создаем файл archive.php в каталоге templates и помещаем в него код:

|  |  |  |
| --- | --- | --- |
| 01 | <?php include "templates/include/header.php" ?> | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | <h1>Article Archive</h1> | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | <ul id="headlines" class="archive"> | |
| 06 |  |

|  |  |  |
| --- | --- | --- |
| 07 | <?php foreach ( $results['articles'] as $article ) { ?> | |
| 08 |  |

|  |  |
| --- | --- |
| 09 | <li> |
| 10 | <h2> | |

|  |  |  |
| --- | --- | --- |
| 11 | <span class="pubDate"><?php echo date('j F Y', $article->publicationDate)?></span><a href=".?action=viewArticle&amp;articleId=<?php echo $article->id?>"><?php echo htmlspecialchars( $article->title )?></a> | |
| 12 | </h2> |

|  |  |  |
| --- | --- | --- |
| 13 | <p class="summary"><?php echo htmlspecialchars( $article->summary )?></p> | |
| 14 | </li> |

|  |  |
| --- | --- |
| 15 |  |
| 16 | <?php } ?> | |

|  |  |
| --- | --- |
| 17 |  |
| 18 | </ul> | |

|  |  |
| --- | --- |
| 19 |  |
| 20 | <p><?php echo $results['totalRows']?> article<?php echo ( $results['totalRows'] != 1 ) ? 's' : '' ?> in total.</p> | |

|  |  |
| --- | --- |
| 21 |  |
| 22 | <p><a href="./">Return to Homepage</a></p> | |

|  |  |
| --- | --- |
| 23 |  |
| 24 | <?php include "templates/include/footer.php" ?> | |

Данный шаблон выводит архив всех статей, которые имеются в нашей CMS. Код практически идентичен homepage.php. Здесь добавляется класс CSS archive для неупорядоченного списка, чтобы мы могли формировать пункты списка отличными от главной страницы. Также добавляем год в дату публикации, так как архив может содержать данные за несколько лет.

На странице также выводится общее число статей в базе данных, которое мы получаем с помощью $results['totalRows']. А вместо ссылки на архив выводится ссылка на главную страницу "Return to Homepage".

### viewArticle.php

Последний шаблон для клиентской части выводится статью для пользователя. Создаем файл с именем viewArticle.php в папке templates folder и копируем в него следующий код:

|  |  |  |
| --- | --- | --- |
| 01 | <?php include "templates/include/header.php" ?> | |
| 02 |  |

|  |  |
| --- | --- |
| 03 | <h1 style="width: 75%;"><?php echo htmlspecialchars( $results['article']->title )?></h1> |
| 04 | <div style="width: 75%; font-style: italic;"><?php echo htmlspecialchars( $results['article']->summary )?></div> |

|  |  |
| --- | --- |
| 05 | <div style="width: 75%;"><?php echo $results['article']->content?></div> |
| 06 | <p class="pubDate">Published on <?php echo date('j F Y', $results['article']->publicationDate)?></p> |

|  |  |
| --- | --- |
| 07 |  |
| 08 | <p><a href="./">Return to Homepage</a></p> | |

|  |  |
| --- | --- |
| 09 |  |
| 10 | <?php include "templates/include/footer.php" ?> | |

Данный шаблон имеет достаточно простой код. Здесь выводится для выбранной статьи название, резюме, содержание и дата публикации. Кроме того на странице размещается ссылка для возвращения на главную.

Шаблон для клиентской части готов. [В следующем уроке](http://ruseller.com/lessons.php?rub=37&id=1591) мы сделаем шаблон для серверной части нашей CMS.

### loginForm.php

Создаем папку с именем admin в каталоге нашего проекта  templates. В папке admin создаем файл loginForm.php:

|  |  |  |
| --- | --- | --- |
| 01 | <?php include "templates/include/header.php" ?> | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | <form action="admin.php?action=login" method="post" style="width: 50%;"> | |
| 04 | <input type="hidden" name="login" value="true" /> |

|  |  |
| --- | --- |
| 05 |  |
| 06 | <?php if ( isset( $results['errorMessage'] ) ) { ?> | |

|  |  |  |
| --- | --- | --- |
| 07 | <div class="errorMessage"><?php echo $results['errorMessage'] ?></div> | |
| 08 | <?php } ?> |

|  |  |
| --- | --- |
| 09 |  |
| 10 | <ul> | |

|  |  |
| --- | --- |
| 11 |  |
| 12 | <li> | |

|  |  |
| --- | --- |
| 13 | <label for="username">Username</label> |
| 14 | <input type="text" name="username" id="username" placeholder="Your admin username" required autofocus maxlength="20" /> | |

|  |  |  |
| --- | --- | --- |
| 15 | </li> | |
| 16 |  |

|  |  |
| --- | --- |
| 17 | <li> |
| 18 | <label for="password">Password</label> | |

|  |  |  |
| --- | --- | --- |
| 19 | <input type="password" name="password" id="password" placeholder="Your admin password" required maxlength="20" /> | |
| 20 | </li> |

|  |  |
| --- | --- |
| 21 |  |
| 22 | </ul> | |

|  |  |
| --- | --- |
| 23 |  |
| 24 | <div class="buttons"> | |

|  |  |  |
| --- | --- | --- |
| 25 | <input type="submit" name="login" value="Login" /> | |
| 26 | </div> |

|  |  |
| --- | --- |
| 27 |  |
| 28 | </form> | |

|  |  |
| --- | --- |
| 29 |  |
| 30 | <?php include "templates/include/footer.php" ?> | |

Эта страница содержит форму регистрации администратора, которая отправлет к admin.php?action=login. Форма содержит скрытое поле login, которое используетcя в функции login() для проверки факта отправки формы. Также здесь имеется область для любых сообщений об ошибках (таких, как неправильное имя пользователя или неправильный пароль) и поля для имени пользователя и пароля, и кнопка отправки формы.

Мы использовали некоторые атрибуты HTML5, такие как placeholder, required, autofocus и date для полей в наших формах. Такие атрибуты делают формы более удобными для использования, а также  сохраняют возможность для проверки необходимых значений в нашем коде PHP. Так как не все браузеры сейчас поддерживают данные атрибуты HTML5, то вы,       возможно, будете применять  JavaScript и/или PHP функции для проверки необходимых значений в серверной части.

### listArticles.php

Сейчас создадим второй администраторский шаблон в вашей папке admin. Он называется listArticles.php:

|  |  |  |
| --- | --- | --- |
| 01 | <?php include "templates/include/header.php" ?> | |
| 02 |  |

|  |  |
| --- | --- |
| 03 | <div id="adminHeader"> |
| 04 | <h2>Widget News Admin</h2> | |

|  |  |  |
| --- | --- | --- |
| 05 | <p>You are logged in as <b><?php echo htmlspecialchars( $\_SESSION['username']) ?></b>. <a href="admin.php?action=logout"?>Log out</a></p> | |
| 06 | </div> |

|  |  |
| --- | --- |
| 07 |  |
| 08 | <h1>All Articles</h1> | |

|  |  |
| --- | --- |
| 09 |  |
| 10 | <?php if ( isset( $results['errorMessage'] ) ) { ?> | |

|  |  |  |
| --- | --- | --- |
| 11 | <div class="errorMessage"><?php echo $results['errorMessage'] ?></div> | |
| 12 | <?php } ?> |

|  |  |
| --- | --- |
| 13 |  |
| 14 |  |

|  |  |
| --- | --- |
| 15 | <?php if ( isset( $results['statusMessage'] ) ) { ?> |
| 16 | <div class="statusMessage"><?php echo $results['statusMessage'] ?></div> | |

|  |  |  |
| --- | --- | --- |
| 17 | <?php } ?> | |
| 18 |  |

|  |  |  |
| --- | --- | --- |
| 19 | <table> | |
| 20 | <tr> |

|  |  |  |
| --- | --- | --- |
| 21 | <th>Publication Date</th> | |
| 22 | <th>Article</th> |

|  |  |  |
| --- | --- | --- |
| 23 | </tr> | |
| 24 |  |

|  |  |  |
| --- | --- | --- |
| 25 | <?php foreach ( $results['articles'] as $article ) { ?> | |
| 26 |  |

|  |  |  |
| --- | --- | --- |
| 27 | <tr onclick="location='admin.php?action=editArticle&amp;articleId=<?php echo $article->id?>'"> | |
| 28 | <td><?php echo date('j M Y', $article->publicationDate)?></td> |

|  |  |
| --- | --- |
| 29 | <td> |
| 30 | <?php echo $article->title?> | |

|  |  |  |
| --- | --- | --- |
| 31 | </td> | |
| 32 | </tr> |

|  |  |
| --- | --- |
| 33 |  |
| 34 | <?php } ?> | |

|  |  |
| --- | --- |
| 35 |  |
| 36 | </table> | |

|  |  |
| --- | --- |
| 37 |  |
| 38 | <p><?php echo $results['totalRows']?> article<?php echo ( $results['totalRows'] != 1 ) ? 's' : '' ?> in total.</p> | |

|  |  |
| --- | --- |
| 39 |  |
| 40 | <p><a href="admin.php?action=newArticle">Add a New Article</a></p> | |

|  |  |
| --- | --- |
| 41 |  |
| 42 | <?php include "templates/include/footer.php" ?> | |

Этот шаблон выводит список статей для редактирования администратором. После отображения любых сообщений об ошибке или статусе мы проходим циклом по массиву объектов Article, который содержится в $results['articles'], и выводим на экран дату и название для каждой статьи в таблицу. К каждой строке таблицы добавляем событие JavaScript onclick, чтобы администратор мог щелчком мыши открыть статью для редактирования.

Шаблон также выводит общее количество всех статей и ссылку для добавления нового материала.

### editArticle.php

Теперь создадим последний шаблон для администраторской части editArticle.php в папке нашего проекта admin:

|  |  |  |
| --- | --- | --- |
| 01 | <?php include "templates/include/header.php" ?> | |
| 02 |  |

|  |  |
| --- | --- |
| 03 | <div id="adminHeader"> |
| 04 | <h2>Widget News Admin</h2> | |

|  |  |  |
| --- | --- | --- |
| 05 | <p>You are logged in as <b><?php echo htmlspecialchars( $\_SESSION['username']) ?></b>. <a href="admin.php?action=logout"?>Log out</a></p> | |
| 06 | </div> |

|  |  |
| --- | --- |
| 07 |  |
| 08 | <h1><?php echo $results['pageTitle']?></h1> | |

|  |  |
| --- | --- |
| 09 |  |
| 10 | <form action="admin.php?action=<?php echo $results['formAction']?>" method="post"> | |

|  |  |  |
| --- | --- | --- |
| 11 | <input type="hidden" name="articleId" value="<?php echo $results['article']->id ?>"/> | |
| 12 |  |

|  |  |
| --- | --- |
| 13 | <?php if ( isset( $results['errorMessage'] ) ) { ?> |
| 14 | <div class="errorMessage"><?php echo $results['errorMessage'] ?></div> | |

|  |  |  |
| --- | --- | --- |
| 15 | <?php } ?> | |
| 16 |  |

|  |  |  |
| --- | --- | --- |
| 17 | <ul> | |
| 18 |  |

|  |  |
| --- | --- |
| 19 | <li> |
| 20 | <label for="title">Article Title</label> | |

|  |  |  |
| --- | --- | --- |
| 21 | <input type="text" name="title" id="title" placeholder="Name of the article" required autofocus maxlength="255" value="<?php echo htmlspecialchars( $results['article']->title )?>" /> | |
| 22 | </li> |

|  |  |
| --- | --- |
| 23 |  |
| 24 | <li> | |

|  |  |
| --- | --- |
| 25 | <label for="summary">Article Summary</label> |
| 26 | <textarea name="summary" id="summary" placeholder="Brief description of the article" required maxlength="1000" style="height: 5em;"><?php echo htmlspecialchars( $results['article']->summary )?></textarea> | |

|  |  |  |
| --- | --- | --- |
| 27 | </li> | |
| 28 |  |

|  |  |
| --- | --- |
| 29 | <li> |
| 30 | <label for="content">Article Content</label> | |

|  |  |  |
| --- | --- | --- |
| 31 | <textarea name="content" id="content" placeholder="The HTML content of the article" required maxlength="100000" style="height: 30em;"><?php echo htmlspecialchars( $results['article']->content )?></textarea> | |
| 32 | </li> |

|  |  |
| --- | --- |
| 33 |  |
| 34 | <li> | |

|  |  |
| --- | --- |
| 35 | <label for="publicationDate">Publication Date</label> |
| 36 | <input type="date" name="publicationDate" id="publicationDate" placeholder="YYYY-MM-DD" required maxlength="10" value="<?php echo $results['article']->publicationDate ? date( "Y-m-d", $results['article']->publicationDate ) : "" ?>" /> | |

|  |  |  |
| --- | --- | --- |
| 37 | </li> | |
| 38 |  |

|  |  |
| --- | --- |
| 39 |  |
| 40 | </ul> | |

|  |  |
| --- | --- |
| 41 |  |
| 42 | <div class="buttons"> | |

|  |  |
| --- | --- |
| 43 | <input type="submit" name="saveChanges" value="Save Changes" /> |
| 44 | <input type="submit" formnovalidate name="cancel" value="Cancel" /> | |

|  |  |  |
| --- | --- | --- |
| 45 | </div> | |
| 46 |  |

|  |  |  |
| --- | --- | --- |
| 47 | </form> | |
| 48 |  |

|  |  |
| --- | --- |
| 49 | <?php if ( $results['article']->id ) { ?> |
| 50 | <p><a href="admin.php?action=deleteArticle&amp;articleId=<?php echo $results['article']->id ?>" onclick="return confirm('Delete This Article?')">Delete This Article</a></p> | |

|  |  |  |
| --- | --- | --- |
| 51 | <?php } ?> | |
| 52 |  |

|  |  |
| --- | --- |
| 53 | <?php include "templates/include/footer.php" ?> |

Здесь выводится форма редактирования, которая используется как для создания новых статей, так и для редактирования существующих. Она отправляет к admin.php?action=newArticle, или к admin.php?action=editArticle, в зависимости от значения переменной $results['formAction']. Шаблон содержит также скрытое поле articleId, для отслеживания ID редактируемой статьи (если он есть).

Форма также имеет область для сообщений об ошибках, а также поля для названия статьи, резюме, содержания и даты публикации. Здесь также имеются 2 кнопки для сохранения и удаления изменений и ссылка, позволяющая администратору удалить только что отредактированную статью.

Все данные передаются через функцию htmlspecialchars() до их вывода в разметке. Это не только хорошая привычка в плане безопасности, но это также гарантирует, чтобы наши значения в полях формы имели правильный формат. Например, когда значение поля title, содержащего двойные кавычки ("), не проходит такой обработки при размещении в коде, оно может быть обрезано, так как двойные кавычки используются для разделения значения поля и разметки.

Обратите внимание, что мы используем атрибут HTML5 formnovalidate для кнопки "Cancel". Данный атрибут указывает браузеру не проводить проверку формы, если пользователь нажал кнопку "Cancel".

## Стили и логотип

Наша CMS практически готова, но для лучшего внешнего вида мы создаем набор правил CSS и сохраняем их в файле style.css в папке cms:

|  |  |  |
| --- | --- | --- |
| 001 | /\* Стили для body and внешнего контейнера \*/ | |
| 002 |  |

|  |  |
| --- | --- |
| 003 | body { |
| 004 | margin: 0; | |

|  |  |
| --- | --- |
| 005 | color: #333; |
| 006 | background-color: #00a0b0; | |

|  |  |  |
| --- | --- | --- |
| 007 | font-family: "Trebuchet MS", Arial, Helvetica, sans-serif; | |
| 008 | line-height: 1.5em; |

|  |  |  |
| --- | --- | --- |
| 009 | } | |
| 010 |  |

|  |  |
| --- | --- |
| 011 | #container { |
| 012 | width: 960px; | |

|  |  |
| --- | --- |
| 013 | background: #fff; |
| 014 | margin: 20px auto; | |

|  |  |
| --- | --- |
| 015 | padding: 20px; |
| 016 | -moz-border-radius: 5px; | |

|  |  |  |
| --- | --- | --- |
| 017 | -webkit-border-radius: 5px; | |
| 018 | border-radius: 5px; |

|  |  |  |
| --- | --- | --- |
| 019 | } | |
| 020 |  |

|  |  |
| --- | --- |
| 021 |  |
| 022 | /\* Логотип и нижний колонтитул \*/ | |

|  |  |
| --- | --- |
| 023 |  |
| 024 | #logo { | |

|  |  |  |
| --- | --- | --- |
| 025 | display: block; | |
| 026 | width: 300px; |

|  |  |  |
| --- | --- | --- |
| 027 | padding: 0 660px 20px 0; | |
| 028 | border: none; |

|  |  |  |
| --- | --- | --- |
| 029 | border-bottom: 1px solid #00a0b0; | |
| 030 | margin-bottom: 40px; |

|  |  |  |
| --- | --- | --- |
| 031 | } | |
| 032 |  |

|  |  |
| --- | --- |
| 033 | #footer { |
| 034 | border-top: 1px solid #00a0b0; | |

|  |  |
| --- | --- |
| 035 | margin-top: 40px; |
| 036 | padding: 20px 0 0 0; | |

|  |  |  |
| --- | --- | --- |
| 037 | font-size: .8em; | |
| 038 | } |

|  |  |
| --- | --- |
| 039 |  |
| 040 |  |

|  |  |  |
| --- | --- | --- |
| 041 | /\* Заголвки \*/ | |
| 042 |  |

|  |  |
| --- | --- |
| 043 | h1 { |
| 044 | color: #eb6841; | |

|  |  |  |
| --- | --- | --- |
| 045 | margin-bottom: 30px; | |
| 046 | line-height: 1.2em; |

|  |  |  |
| --- | --- | --- |
| 047 | } | |
| 048 |  |

|  |  |
| --- | --- |
| 049 | h2, h2 a { |
| 050 | color: #edc951; | |

|  |  |  |
| --- | --- | --- |
| 051 | } | |
| 052 |  |

|  |  |
| --- | --- |
| 053 | h2 a { |
| 054 | text-decoration: none; | |

|  |  |  |
| --- | --- | --- |
| 055 | } | |
| 056 |  |

|  |  |
| --- | --- |
| 057 |  |
| 058 | /\* Заголовки статей \*/ | |

|  |  |
| --- | --- |
| 059 |  |
| 060 | #headlines { | |

|  |  |  |
| --- | --- | --- |
| 061 | list-style: none; | |
| 062 | padding-left: 0; |

|  |  |  |
| --- | --- | --- |
| 063 | width: 75%; | |
| 064 | } |

|  |  |
| --- | --- |
| 065 |  |
| 066 | #headlines li { | |

|  |  |  |
| --- | --- | --- |
| 067 | margin-bottom: 2em; | |
| 068 | } |

|  |  |
| --- | --- |
| 069 |  |
| 070 | .pubDate { | |

|  |  |  |
| --- | --- | --- |
| 071 | font-size: .8em; | |
| 072 | color: #eb6841; |

|  |  |  |
| --- | --- | --- |
| 073 | text-transform: uppercase; | |
| 074 | } |

|  |  |
| --- | --- |
| 075 |  |
| 076 | #headlines .pubDate { | |

|  |  |  |
| --- | --- | --- |
| 077 | display: inline-block; | |
| 078 | width: 100px; |

|  |  |
| --- | --- |
| 079 | font-size: .5em; |
| 080 | vertical-align: middle; | |

|  |  |  |
| --- | --- | --- |
| 081 | } | |
| 082 |  |

|  |  |  |
| --- | --- | --- |
| 083 | #headlines.archive .pubDate { | |
| 084 | width: 130px; |

|  |  |  |
| --- | --- | --- |
| 085 | } | |
| 086 |  |

|  |  |
| --- | --- |
| 087 | .summary { |
| 088 | padding-left: 100px; | |

|  |  |  |
| --- | --- | --- |
| 089 | } | |
| 090 |  |

|  |  |  |
| --- | --- | --- |
| 091 | #headlines.archive .summary { | |
| 092 | padding-left: 130px; |

|  |  |  |
| --- | --- | --- |
| 093 | } | |
| 094 |  |

|  |  |
| --- | --- |
| 095 |  |
| 096 | /\* Заголовок для страницы администратора \*/ | |

|  |  |
| --- | --- |
| 097 |  |
| 098 | #adminHeader { | |

|  |  |
| --- | --- |
| 099 | width: 940px; |
| 100 | padding: 0 10px; | |

|  |  |  |
| --- | --- | --- |
| 101 | border-bottom: 1px solid #00a0b0; | |
| 102 | margin: -30px 0 40px 0; |

|  |  |  |
| --- | --- | --- |
| 103 | font-size: 0.8em; | |
| 104 | } |

|  |  |
| --- | --- |
| 105 |  |
| 106 |  |

|  |  |  |
| --- | --- | --- |
| 107 | /\* Стили для формы с цветным фоном, скругленными углами и тенью \*/ | |
| 108 |  |

|  |  |
| --- | --- |
| 109 | form { |
| 110 | margin: 20px auto; | |

|  |  |  |
| --- | --- | --- |
| 111 | padding: 40px 20px; | |
| 112 | overflow: auto; |

|  |  |
| --- | --- |
| 113 | background: #fff4cf; |
| 114 | border: 1px solid #666; | |

|  |  |
| --- | --- |
| 115 | -moz-border-radius: 5px; |
| 116 | -webkit-border-radius: 5px; | |

|  |  |
| --- | --- |
| 117 | border-radius: 5px; |
| 118 | -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |

|  |  |  |
| --- | --- | --- |
| 119 | -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |
| 120 | box-shadow: 0 0 .5em rgba(0, 0, 0, .8); |

|  |  |  |
| --- | --- | --- |
| 121 | } | |
| 122 |  |

|  |  |
| --- | --- |
| 123 |  |
| 124 | /\* Задаем для элементов формы согласованные поля, отступы, и высоту строки \*/ | |

|  |  |
| --- | --- |
| 125 |  |
| 126 | form ul { | |

|  |  |  |
| --- | --- | --- |
| 127 | list-style: none; | |
| 128 | margin: 0; |

|  |  |  |
| --- | --- | --- |
| 129 | padding: 0; | |
| 130 | } |

|  |  |
| --- | --- |
| 131 |  |
| 132 | form ul li { | |

|  |  |  |
| --- | --- | --- |
| 133 | margin: .9em 0 0 0; | |
| 134 | padding: 0; |

|  |  |  |
| --- | --- | --- |
| 135 | } | |
| 136 |  |

|  |  |
| --- | --- |
| 137 | form \* { |
| 138 | line-height: 1em; | |

|  |  |  |
| --- | --- | --- |
| 139 | } | |
| 140 |  |

|  |  |
| --- | --- |
| 141 |  |
| 142 | /\* Метки полей \*/ | |

|  |  |
| --- | --- |
| 143 |  |
| 144 | label { | |

|  |  |  |
| --- | --- | --- |
| 145 | display: block; | |
| 146 | float: left; |

|  |  |
| --- | --- |
| 147 | clear: left; |
| 148 | text-align: right; | |

|  |  |
| --- | --- |
| 149 | width: 15%; |
| 150 | padding: .4em 0 0 0; | |

|  |  |  |
| --- | --- | --- |
| 151 | margin: .15em .5em 0 0; | |
| 152 | } |

|  |  |
| --- | --- |
| 153 |  |
| 154 |  |

|  |  |  |
| --- | --- | --- |
| 155 | /\* Поля \*/ | |
| 156 |  |

|  |  |  |
| --- | --- | --- |
| 157 | input, select, textarea { | |
| 158 | display: block; |

|  |  |
| --- | --- |
| 159 | margin: 0; |
| 160 | padding: .4em; | |

|  |  |  |
| --- | --- | --- |
| 161 | width: 80%; | |
| 162 | } |

|  |  |
| --- | --- |
| 163 |  |
| 164 | input, textarea, .date { | |

|  |  |
| --- | --- |
| 165 | border: 2px solid #666; |
| 166 | -moz-border-radius: 5px; | |

|  |  |  |
| --- | --- | --- |
| 167 | -webkit-border-radius: 5px; | |
| 168 | border-radius: 5px; |

|  |  |  |
| --- | --- | --- |
| 169 | background: #fff; | |
| 170 | } |

|  |  |
| --- | --- |
| 171 |  |
| 172 | input { | |

|  |  |  |
| --- | --- | --- |
| 173 | font-size: .9em; | |
| 174 | } |

|  |  |
| --- | --- |
| 175 |  |
| 176 | select { | |

|  |  |
| --- | --- |
| 177 | padding: 0; |
| 178 | margin-bottom: 2.5em; | |

|  |  |  |
| --- | --- | --- |
| 179 | position: relative; | |
| 180 | top: .7em; |

|  |  |  |
| --- | --- | --- |
| 181 | } | |
| 182 |  |

|  |  |
| --- | --- |
| 183 | textarea { |
| 184 | font-family: "Trebuchet MS", Arial, Helvetica, sans-serif; | |

|  |  |  |
| --- | --- | --- |
| 185 | font-size: .9em; | |
| 186 | height: 5em; |

|  |  |  |
| --- | --- | --- |
| 187 | line-height: 1.5em; | |
| 188 | } |

|  |  |
| --- | --- |
| 189 |  |
| 190 | textarea#content { | |

|  |  |  |
| --- | --- | --- |
| 191 | font-family: "Courier New", courier, fixed; | |
| 192 | } |

|  |  |  |
| --- | --- | --- |
| 193 |  | |
| 194 |  |

|  |  |  |
| --- | --- | --- |
| 195 | /\* Рамка вокруг поля с фокусом ввода \*/ | |
| 196 |  |

|  |  |
| --- | --- |
| 197 | form \*:focus { |
| 198 | border: 2px solid #7c412b; | |

|  |  |  |
| --- | --- | --- |
| 199 | outline: none; | |
| 200 | } |

|  |  |
| --- | --- |
| 201 |  |
| 202 |  |

|  |  |  |
| --- | --- | --- |
| 203 | /\* Правильно заполненное поле имеет зеленый фон \*/ | |
| 204 |  |

|  |  |  |
| --- | --- | --- |
| 205 | input:valid, textarea:valid { | |
| 206 | background: #efe; |

|  |  |  |
| --- | --- | --- |
| 207 | } | |
| 208 |  |

|  |  |
| --- | --- |
| 209 |  |
| 210 | /\* Кнопки отправки \*/ | |

|  |  |
| --- | --- |
| 211 |  |
| 212 | .buttons { | |

|  |  |  |
| --- | --- | --- |
| 213 | text-align: center; | |
| 214 | margin: 40px 0 0 0; |

|  |  |  |
| --- | --- | --- |
| 215 | } | |
| 216 |  |

|  |  |  |
| --- | --- | --- |
| 217 | input[type="submit"] { | |
| 218 | display: inline; |

|  |  |  |
| --- | --- | --- |
| 219 | margin: 0 20px; | |
| 220 | width: 12em; |

|  |  |
| --- | --- |
| 221 | padding: 10px; |
| 222 | border: 2px solid #7c412b; | |

|  |  |
| --- | --- |
| 223 | -moz-border-radius: 5px; |
| 224 | -webkit-border-radius: 5px; | |

|  |  |
| --- | --- |
| 225 | border-radius: 5px; |
| 226 | -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |

|  |  |  |
| --- | --- | --- |
| 227 | -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |
| 228 | box-shadow: 0 0 .5em rgba(0, 0, 0, .8); |

|  |  |
| --- | --- |
| 229 | color: #fff; |
| 230 | background: #ef7d50; | |

|  |  |
| --- | --- |
| 231 | font-weight: bold; |
| 232 | -webkit-appearance: none; | |

|  |  |  |
| --- | --- | --- |
| 233 | } | |
| 234 |  |

|  |  |  |
| --- | --- | --- |
| 235 | input[type="submit"]:hover, input[type="submit"]:active { | |
| 236 | cursor: pointer; |

|  |  |  |
| --- | --- | --- |
| 237 | background: #fff; | |
| 238 | color: #ef7d50; |

|  |  |  |
| --- | --- | --- |
| 239 | } | |
| 240 |  |

|  |  |  |
| --- | --- | --- |
| 241 | input[type="submit"]:active { | |
| 242 | background: #eee; |

|  |  |
| --- | --- |
| 243 | -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8) inset; |
| 244 | -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8) inset; | |

|  |  |  |
| --- | --- | --- |
| 245 | box-shadow: 0 0 .5em rgba(0, 0, 0, .8) inset; | |
| 246 | } |

|  |  |
| --- | --- |
| 247 |  |
| 248 |  |

|  |  |  |
| --- | --- | --- |
| 249 | /\* Таблицы \*/ | |
| 250 |  |

|  |  |
| --- | --- |
| 251 | table { |
| 252 | width: 100%; | |

|  |  |  |
| --- | --- | --- |
| 253 | border-collapse: collapse; | |
| 254 | } |

|  |  |
| --- | --- |
| 255 |  |
| 256 | tr, th, td { | |

|  |  |  |
| --- | --- | --- |
| 257 | padding: 10px; | |
| 258 | margin: 0; |

|  |  |  |
| --- | --- | --- |
| 259 | text-align: left; | |
| 260 | } |

|  |  |
| --- | --- |
| 261 |  |
| 262 | table, th { | |

|  |  |  |
| --- | --- | --- |
| 263 | border: 1px solid #00a0b0; | |
| 264 | } |

|  |  |
| --- | --- |
| 265 |  |
| 266 | th { | |

|  |  |
| --- | --- |
| 267 | border-left: none; |
| 268 | border-right: none; | |

|  |  |  |
| --- | --- | --- |
| 269 | background: #ef7d50; | |
| 270 | color: #fff; |

|  |  |  |
| --- | --- | --- |
| 271 | cursor: default; | |
| 272 | } |

|  |  |
| --- | --- |
| 273 |  |
| 274 | tr:nth-child(odd) { | |

|  |  |  |
| --- | --- | --- |
| 275 | background: #fff4cf; | |
| 276 | } |

|  |  |
| --- | --- |
| 277 |  |
| 278 | tr:nth-child(even) { | |

|  |  |  |
| --- | --- | --- |
| 279 | background: #fff; | |
| 280 | } |

|  |  |
| --- | --- |
| 281 |  |
| 282 | tr:hover { | |

|  |  |  |
| --- | --- | --- |
| 283 | background: #ddd; | |
| 284 | cursor: pointer; |

|  |  |  |
| --- | --- | --- |
| 285 | } | |
| 286 |  |

|  |  |
| --- | --- |
| 287 |  |
| 288 | /\* Окна для вывода статуса и сообщений об ошибках \*/ | |

|  |  |
| --- | --- |
| 289 |  |
| 290 | .statusMessage, .errorMessage { | |

|  |  |  |
| --- | --- | --- |
| 291 | font-size: .8em; | |
| 292 | padding: .5em; |

|  |  |
| --- | --- |
| 293 | margin: 2em 0; |
| 294 | -moz-border-radius: 5px; | |

|  |  |  |
| --- | --- | --- |
| 295 | -webkit-border-radius: 5px; | |
| 296 | border-radius: 5px; |

|  |  |
| --- | --- |
| 297 | -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); |
| 298 | -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |

|  |  |  |
| --- | --- | --- |
| 299 | -box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |
| 300 | } |

|  |  |
| --- | --- |
| 301 |  |
| 302 | .statusMessage { | |

|  |  |  |
| --- | --- | --- |
| 303 | background-color: #2b2; | |
| 304 | border: 1px solid #080; |

|  |  |  |
| --- | --- | --- |
| 305 | color: #fff; | |
| 306 | } |

|  |  |
| --- | --- |
| 307 |  |
| 308 | .errorMessage { | |

|  |  |  |
| --- | --- | --- |
| 309 | background-color: #f22; | |
| 310 | border: 1px solid #800; |

|  |  |  |
| --- | --- | --- |
| 311 | color: #fff; | |
| 312 | } |

Детального разбора кода CSS  мы делать не будем, так как наши уроки посвящены PHP и MySQL.

А в папке images в каталоге нашего проекта cms размещаем файл изображения logo.jpg :

[В предыдущих уроках](http://ruseller.com/lessons.php?rub=37&id=1582) серии мы построили простую CMS, которая имеет клиентскую и серверную части и позволяет осуществлять примитивное управление содержанием сайта без редактирования кода. Теперь расширим функционал нашего проекта. Добавим использование категорий.

Категории добавляют гибкости сайту - кроме перечисления всех статей в списке можно создавать отдельные страницы для категории, на которых будут выводиться только статьи, объединенные по какому-либо признаку.

Например, наша CMS выводит все типы статей вместе на главной и в архиве. Создав различные категории для новостей, обзоров и интервью можно будет использовать разные страницы для соответствующих разделов на нашем сайте, что существенно улучшит уровень удобства пользования ресурсом.

В новой демонстрации можно просмотреть, как функционируют категории. Каждый заголовок статьи сопровождается названием категории (он расположено ниже названия)  - Interviews, Reviews или News. Нажав на название категории можно открыть страницу архива, на которой перечислены все статьи категории и ее описание вверху.

## План действий

Возьмем за исходную точку нашу готовую CMS и будем модифицировать ее для добавления функционала категорий. Нужно сделать следующие шаги:

1. Модифицировать базу данных
2. Построить класс Category
3. Модифицировать класс Article для управления категориями
4. Модифицировать index.php для вывода категорий
5. Модифицировать admin.php для перечисления, добавления, редактирования, удаления и назначения категорий
6. Модифицировать шаблоны клиентской части и таблицы стилей для вывода категорий
7. Модифицировать шаблоны серверной части управления перечислениями, добавлением, редактированием, удалением и назначением категорий.

## Модифицируем базу данных

Сначала нужно расширить возможности базы данных MySQL для поддержки категорий. Нужно создать новую таблицу categories и модифицировать таблицу articles для включения ID категории, которая связана с каждой статьей.

Открываем файл tables.sql оригинальной CMS и вносим изменения:

|  |  |  |
| --- | --- | --- |
| 01 | DROP TABLE IF EXISTS categories; | |
| 02 | CREATE TABLE categories |

|  |  |
| --- | --- |
| 03 | ( |
| 04 | id              smallint unsigned NOT NULL auto\_increment, | |

|  |  |
| --- | --- |
| 05 | name            varchar(255) NOT NULL,                      # Название категории |
| 06 | description     text NOT NULL,                              # Короткое описание категории |

|  |  |
| --- | --- |
| 07 |  |
| 08 | PRIMARY KEY     (id) | |

|  |  |  |
| --- | --- | --- |
| 09 | ); | |
| 10 |  |

|  |  |  |
| --- | --- | --- |
| 11 | DROP TABLE IF EXISTS articles; | |
| 12 | CREATE TABLE articles |

|  |  |
| --- | --- |
| 13 | ( |
| 14 | id              smallint unsigned NOT NULL auto\_increment, | |

|  |  |
| --- | --- |
| 15 | publicationDate date NOT NULL,                              # Дата публикации категории |
| 16 | categoryId      smallint unsigned NOT NULL,                 # Идентификатор категории статьи |

|  |  |
| --- | --- |
| 17 | title           varchar(255) NOT NULL,                      # Полное название статьи |
| 18 | summary         text NOT NULL,                              # Короткое резюме |

|  |  |  |
| --- | --- | --- |
| 19 | content         mediumtext NOT NULL,                        # Содержание HTML статьи | |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | PRIMARY KEY     (id) | |
| 22 | ); |

Мы добавили новую таблицу categories для хранения категорий. Каждая категория имеет уникальное поле для идентификации (id), название категории (name) и краткое описание для вывода на архивной странице (description).

Также мы изменяем таблицу articles для добавления поля categoryId, которое используется для связывания каждой статьи с соответствующей категорией.

Поле, которое связывает одну таблицу с другой называется как внешний ключ (foreign key).

### А что делать, если у вас уже есть статьи в базе данных?

Если использовать запрос из файла tables.sql в MySQL, то он удалит существующую таблицу articles в вашей базе cms и построит ее с нуля. То есть все статьи, которые уже есть в CMS удалятся.

В подобной ситуации нужно модифицировать таблицу articles с сохранением всех данных. MySQL позволяет выполнять такие задачи легко, благодаря выражению ALTER TABLE. Упрощенно синтаксис ALTER TABLE выглядит так:

ALTER TABLE имя\_таблицы ADD имя\_нового\_поля определение\_нового\_поля AFTER существующее\_поле

Если у вас есть статьи в CMS, то нужно создать новую таблицу categories и модифицировать имеющуюся таблицу articles. Чтобы выполнить задачу, запрос в файле tables.sql должен выглядеть так:

|  |  |  |
| --- | --- | --- |
| 01 | DROP TABLE IF EXISTS categories; | |
| 02 | CREATE TABLE categories |

|  |  |
| --- | --- |
| 03 | ( |
| 04 | id              smallint unsigned NOT NULL auto\_increment, | |

|  |  |
| --- | --- |
| 05 | name            varchar(255) NOT NULL,                      # Name of the category |
| 06 | description     text NOT NULL,                              # A short description of the category |

|  |  |
| --- | --- |
| 07 |  |
| 08 | PRIMARY KEY     (id) | |

|  |  |  |
| --- | --- | --- |
| 09 | ); | |
| 10 |  |

|  |  |
| --- | --- |
| 11 | ALTER TABLE articles ADD categoryId smallint unsigned NOT NULL AFTER publicationDate; |

## Применяем изменения

Теперь нужно применить изменения к нашей базе MySQL.

Если у вас уже есть база данных cms, которая содержит статьи, то необходимо сделать [резервную копию](http://ruseller.com/adds.php?rub_id=12&id=1662) перед внесением изменений!

Если у вас нет базу данных cms, то нужно ее создать, как было описано [в предыдущих уроках серии](http://ruseller.com/lessons.php?rub=37&id=1582):

|  |  |
| --- | --- |
| 1 | mysql -u username -p |
| 2 | create database cms; |

|  |  |
| --- | --- |
| 3 | exit |

Теперь можно загрузить файл tables.sql в MySQL для внесения изменений в таблицу данных:

|  |  |
| --- | --- |
| 1 | mysql -u username -p cms < tables.sql |

Вводим пароль, когда система его запросит и нажимаем Enter. MySQL читает файл tables.sql и выполняет команды из него, создавая и/или модифицируя таблицу в базе данных cms.

Для проверки результатов нужно войти в MySQL:

|  |  |
| --- | --- |
| 1 | mysql -u username -p cms |

Затем используем команды SHOW TABLES и EXPLAIN для проверки схем таблиц в MySQL:

|  |  |  |
| --- | --- | --- |
| 01 | mysql> show tables; | |
| 02 | +---------------+ |

|  |  |
| --- | --- |
| 03 | | Tables\_in\_cms | |
| 04 | +---------------+ |

|  |  |
| --- | --- |
| 05 | | articles      | |
| 06 | | categories    | |

|  |  |
| --- | --- |
| 07 | +---------------+ |
| 08 | 2 rows in set (0.00 sec) | |

|  |  |
| --- | --- |
| 09 |  |
| 10 | mysql> explain articles; | |

|  |  |
| --- | --- |
| 11 | +-----------------+----------------------+------+-----+---------+----------------+ |
| 12 | | Field           | Type                 | Null | Key | Default | Extra          | |

|  |  |
| --- | --- |
| 13 | +-----------------+----------------------+------+-----+---------+----------------+ |
| 14 | | id              | smallint(5) unsigned | NO   | PRI | NULL    | auto\_increment | |

|  |  |
| --- | --- |
| 15 | | publicationDate | date                 | NO   |     | NULL    |                | |
| 16 | | categoryId      | smallint(5) unsigned | NO   |     | NULL    |                | | |

|  |  |
| --- | --- |
| 17 | | title           | varchar(255)         | NO   |     | NULL    |                | |
| 18 | | summary         | text                 | NO   |     | NULL    |                | |

|  |  |
| --- | --- |
| 19 | | content         | mediumtext           | NO   |     | NULL    |                | |
| 20 | +-----------------+----------------------+------+-----+---------+----------------+ |

|  |  |  |
| --- | --- | --- |
| 21 | 6 rows in set (0.00 sec) | |
| 22 |  |

|  |  |
| --- | --- |
| 23 | mysql> explain categories; |
| 24 | +-------------+----------------------+------+-----+---------+----------------+ | |

|  |  |
| --- | --- |
| 25 | | Field       | Type                 | Null | Key | Default | Extra          | |
| 26 | +-------------+----------------------+------+-----+---------+----------------+ |

|  |  |
| --- | --- |
| 27 | | id          | smallint(5) unsigned | NO   | PRI | NULL    | auto\_increment | |
| 28 | | name        | varchar(255)         | NO   |     | NULL    |                | |

|  |  |
| --- | --- |
| 29 | | description | text                 | NO   |     | NULL    |                | |
| 30 | +-------------+----------------------+------+-----+---------+----------------+ |

|  |  |  |
| --- | --- | --- |
| 31 | 3 rows in set (0.00 sec) | |
| 32 |  |

|  |  |
| --- | --- |
| 33 | mysql> |

Обратите внимание на новое поле categoryId в таблице articles и новую таблицу  categories.

Теперь база данных нашей CMS для работы с категориями.

[В следующих уроках](http://ruseller.com/lessons.php?rub=37&id=1596) мы реализуем остальные пункты плана по модификации CMS.

**Создание класса Category**

Также как ранее [мы создавали класс Article](http://ruseller.com/lessons.php?rub=37&id=1583) для управления хранением и выводом статей, нам нужно построить класс Category для решения таких же задач с категориями.

В папке cms имеется каталог classes. В каталоге classes создаем новый файл Category.php и копируем в него следующий код:

|  |  |  |
| --- | --- | --- |
| 001 | <?php | |
| 002 |  |

|  |  |
| --- | --- |
| 003 | /\*\* |
| 004 | \* Класс для обработки категорий статей | |

|  |  |  |
| --- | --- | --- |
| 005 | \*/ | |
| 006 |  |

|  |  |  |
| --- | --- | --- |
| 007 | class Category | |
| 008 | { |

|  |  |  |
| --- | --- | --- |
| 009 | // Свойства | |
| 010 |  |

|  |  |
| --- | --- |
| 011 | /\*\* |
| 012 | \* @var int ID категории из базы данных | |

|  |  |
| --- | --- |
| 013 | \*/ |
| 014 | public $id = null; | |

|  |  |
| --- | --- |
| 015 |  |
| 016 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 017 | \* @var string Название категории | |
| 018 | \*/ |

|  |  |  |
| --- | --- | --- |
| 019 | public $name = null; | |
| 020 |  |

|  |  |
| --- | --- |
| 021 | /\*\* |
| 022 | \* @var string Короткое описание категории | |

|  |  |
| --- | --- |
| 023 | \*/ |
| 024 | public $description = null; | |

|  |  |
| --- | --- |
| 025 |  |
| 026 |  |

|  |  |
| --- | --- |
| 027 | /\*\* |
| 028 | \* Устанавливаем свойства объекта с использованием значений в передаваемом массиве | |

|  |  |
| --- | --- |
| 029 | \* |
| 030 | \* @param assoc Значения свойств | |

|  |  |  |
| --- | --- | --- |
| 031 | \*/ | |
| 032 |  |

|  |  |
| --- | --- |
| 033 | public function \_\_construct( $data=array() ) { |
| 034 | if ( isset( $data['id'] ) ) $this->id = (int) $data['id']; | |

|  |  |
| --- | --- |
| 035 | if ( isset( $data['name'] ) ) $this->name = preg\_replace ( "/[^\.\,\-\\_\'\"\@\?\!\:\$ a-zA-Z0-9()]/", "", $data['name'] ); |
| 036 | if ( isset( $data['description'] ) ) $this->description = preg\_replace ( "/[^\.\,\-\\_\'\"\@\?\!\:\$ a-zA-Z0-9()]/", "", $data['description'] ); |

|  |  |  |
| --- | --- | --- |
| 037 | } | |
| 038 |  |

|  |  |
| --- | --- |
| 039 |  |
| 040 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 041 | \* Устанавливаем свойства объекта с использованием значений из формы редактирования | |
| 042 | \* |

|  |  |  |
| --- | --- | --- |
| 043 | \* @param assoc Значения из формы редактирования | |
| 044 | \*/ |

|  |  |
| --- | --- |
| 045 |  |
| 046 | public function storeFormValues ( $params ) { | |

|  |  |
| --- | --- |
| 047 |  |
| 048 | // Store all the parameters | |

|  |  |  |
| --- | --- | --- |
| 049 | $this->\_\_construct( $params ); | |
| 050 | } |

|  |  |
| --- | --- |
| 051 |  |
| 052 |  |

|  |  |
| --- | --- |
| 053 | /\*\* |
| 054 | \* Возвращаем объект Category, соответствующий заданному ID | |

|  |  |
| --- | --- |
| 055 | \* |
| 056 | \* @param int ID категории | |

|  |  |  |
| --- | --- | --- |
| 057 | \* @return Category|false Объект Category object или false, если запись не была найдена или в случае другой ошибки | |
| 058 | \*/ |

|  |  |
| --- | --- |
| 059 |  |
| 060 | public static function getById( $id ) { | |

|  |  |  |
| --- | --- | --- |
| 061 | $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD ); | |
| 062 | $sql = "SELECT \* FROM categories WHERE id = :id"; |

|  |  |
| --- | --- |
| 063 | $st = $conn->prepare( $sql ); |
| 064 | $st->bindValue( ":id", $id, PDO::PARAM\_INT ); | |

|  |  |
| --- | --- |
| 065 | $st->execute(); |
| 066 | $row = $st->fetch(); | |

|  |  |
| --- | --- |
| 067 | $conn = null; |
| 068 | if ( $row ) return new Category( $row ); | |

|  |  |  |
| --- | --- | --- |
| 069 | } | |
| 070 |  |

|  |  |
| --- | --- |
| 071 |  |
| 072 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 073 | \* Возвращаем все (или диапазон) объектов Category из базы данных | |
| 074 | \* |

|  |  |
| --- | --- |
| 075 | \* @param int Optional Количество возвращаемых строк (по умолчаниюt = all) |
| 076 | \* @param string Optional Столбец, по которому сортируются категории(по умолчанию = "name ASC") |

|  |  |  |
| --- | --- | --- |
| 077 | \* @return Array|false Двух элементный массив: results => массив с объектами Category; totalRows => общее количество категорий | |
| 078 | \*/ |

|  |  |
| --- | --- |
| 079 |  |
| 080 | public static function getList( $numRows=1000000, $order="name ASC" ) { | |

|  |  |
| --- | --- |
| 081 | $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD ); |
| 082 | $sql = "SELECT SQL\_CALC\_FOUND\_ROWS \* FROM categories | |

|  |  |  |
| --- | --- | --- |
| 083 | ORDER BY " . mysql\_escape\_string($order) . " LIMIT :numRows"; | |
| 084 |  |

|  |  |
| --- | --- |
| 085 | $st = $conn->prepare( $sql ); |
| 086 | $st->bindValue( ":numRows", $numRows, PDO::PARAM\_INT ); | |

|  |  |
| --- | --- |
| 087 | $st->execute(); |
| 088 | $list = array(); | |

|  |  |
| --- | --- |
| 089 |  |
| 090 | while ( $row = $st->fetch() ) { | |

|  |  |  |
| --- | --- | --- |
| 091 | $category = new Category( $row ); | |
| 092 | $list[] = $category; |

|  |  |  |
| --- | --- | --- |
| 093 | } | |
| 094 |  |

|  |  |  |
| --- | --- | --- |
| 095 | // Получаем общее количество категорий, которые соответствуют критериям | |
| 096 | $sql = "SELECT FOUND\_ROWS() AS totalRows"; |

|  |  |  |
| --- | --- | --- |
| 097 | $totalRows = $conn->query( $sql )->fetch(); | |
| 098 | $conn = null; |

|  |  |  |
| --- | --- | --- |
| 099 | return ( array ( "results" => $list, "totalRows" => $totalRows[0] ) ); | |
| 100 | } |

|  |  |
| --- | --- |
| 101 |  |
| 102 |  |

|  |  |
| --- | --- |
| 103 | /\*\* |
| 104 | \* Вставляем текущий объект Category в базу данных и устанавливаем его свойство ID. | |

|  |  |  |
| --- | --- | --- |
| 105 | \*/ | |
| 106 |  |

|  |  |  |
| --- | --- | --- |
| 107 | public function insert() { | |
| 108 |  |

|  |  |
| --- | --- |
| 109 | // У объекта Category уже есть ID? |
| 110 | if ( !is\_null( $this->id ) ) trigger\_error ( "Category::insert(): Attempt to insert a Category object that already has its ID property set (to $this->id).", E\_USER\_ERROR ); | |

|  |  |
| --- | --- |
| 111 |  |
| 112 | // Вставляем категорию | |

|  |  |
| --- | --- |
| 113 | $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD ); |
| 114 | $sql = "INSERT INTO categories ( name, description ) VALUES ( :name, :description )"; | |

|  |  |
| --- | --- |
| 115 | $st = $conn->prepare ( $sql ); |
| 116 | $st->bindValue( ":name", $this->name, PDO::PARAM\_STR ); | |

|  |  |  |
| --- | --- | --- |
| 117 | $st->bindValue( ":description", $this->description, PDO::PARAM\_STR ); | |
| 118 | $st->execute(); |

|  |  |  |
| --- | --- | --- |
| 119 | $this->id = $conn->lastInsertId(); | |
| 120 | $conn = null; |

|  |  |  |
| --- | --- | --- |
| 121 | } | |
| 122 |  |

|  |  |
| --- | --- |
| 123 |  |
| 124 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 125 | \* Обновляем текущий объект Category в базе данных. | |
| 126 | \*/ |

|  |  |
| --- | --- |
| 127 |  |
| 128 | public function update() { | |

|  |  |
| --- | --- |
| 129 |  |
| 130 | // У объекта Category  есть ID? | |

|  |  |  |
| --- | --- | --- |
| 131 | if ( is\_null( $this->id ) ) trigger\_error ( "Category::update(): Attempt to update a Category object that does not have its ID property set.", E\_USER\_ERROR ); | |
| 132 |  |

|  |  |
| --- | --- |
| 133 | // Обновляем категорию |
| 134 | $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD ); | |

|  |  |  |
| --- | --- | --- |
| 135 | $sql = "UPDATE categories SET name=:name, description=:description WHERE id = :id"; | |
| 136 | $st = $conn->prepare ( $sql ); |

|  |  |
| --- | --- |
| 137 | $st->bindValue( ":name", $this->name, PDO::PARAM\_STR ); |
| 138 | $st->bindValue( ":description", $this->description, PDO::PARAM\_STR ); | |

|  |  |  |
| --- | --- | --- |
| 139 | $st->bindValue( ":id", $this->id, PDO::PARAM\_INT ); | |
| 140 | $st->execute(); |

|  |  |  |
| --- | --- | --- |
| 141 | $conn = null; | |
| 142 | } |

|  |  |
| --- | --- |
| 143 |  |
| 144 |  |

|  |  |
| --- | --- |
| 145 | /\*\* |
| 146 | \* Удаляем текущий объект Category из базы данных. | |

|  |  |  |
| --- | --- | --- |
| 147 | \*/ | |
| 148 |  |

|  |  |  |
| --- | --- | --- |
| 149 | public function delete() { | |
| 150 |  |

|  |  |
| --- | --- |
| 151 | // У объекта Category  есть ID? |
| 152 | if ( is\_null( $this->id ) ) trigger\_error ( "Category::delete(): Attempt to delete a Category object that does not have its ID property set.", E\_USER\_ERROR ); | |

|  |  |
| --- | --- |
| 153 |  |
| 154 | // Удаляем категорию | |

|  |  |
| --- | --- |
| 155 | $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD ); |
| 156 | $st = $conn->prepare ( "DELETE FROM categories WHERE id = :id LIMIT 1" ); | |

|  |  |  |
| --- | --- | --- |
| 157 | $st->bindValue( ":id", $this->id, PDO::PARAM\_INT ); | |
| 158 | $st->execute(); |

|  |  |  |
| --- | --- | --- |
| 159 | $conn = null; | |
| 160 | } |

|  |  |  |
| --- | --- | --- |
| 161 |  | |
| 162 | } |

|  |  |
| --- | --- |
| 163 |  |
| 164 | ?> | |

Данный класс очень похож на [класс Article](http://ruseller.com/lessons.php?rub=37&id=1583) и немного проще. Он содержит три свойства, которые соответствуют полям в таблице categories — id, name и description, конструктор \_\_construct(), который  создает новый объект Category. Также класс имеет метод storeFormValues() для сохранения данных из формы редактирования, методы для получения одной категории по ID и списка категорий, а также методов для вставки, обновления и удаления категории в базе данных.

Нам нужно включить новый класс Category в наши файлы кода, чтобы CMS могла получить доступ к нему. Поэтому включаем файл с классом в файл config.phpв папке cms, также как мы включали Article.php:

|  |  |
| --- | --- |
| 01 | <?php |
| 02 | ini\_set( "display\_errors", true ); | |

|  |  |  |
| --- | --- | --- |
| 03 | date\_default\_timezone\_set( "Australia/Sydney" );  // <http://www.php.net/manual/en/timezones.php> | |
| 04 | define( "DB\_DSN", "mysql:host=localhost;dbname=cms" ); |

|  |  |
| --- | --- |
| 05 | define( "DB\_USERNAME", "username" ); |
| 06 | define( "DB\_PASSWORD", "password" ); |

|  |  |
| --- | --- |
| 07 | define( "CLASS\_PATH", "classes" ); |
| 08 | define( "TEMPLATE\_PATH", "templates" ); | |

|  |  |  |
| --- | --- | --- |
| 09 | define( "HOMEPAGE\_NUM\_ARTICLES", 5 ); | |
| 10 | define( "ADMIN\_USERNAME", "admin" ); |

|  |  |
| --- | --- |
| 11 | define( "ADMIN\_PASSWORD", "mypass" ); |
| 12 | require( CLASS\_PATH . "/Article.php" ); | |

|  |  |  |
| --- | --- | --- |
| 13 | require( CLASS\_PATH . "/Category.php" ); | |
| 14 |  |

|  |  |
| --- | --- |
| 15 | function handleException( $exception ) { |
| 16 | echo "Sorry, a problem occurred. Please try later."; | |

|  |  |  |
| --- | --- | --- |
| 17 | error\_log( $exception->getMessage() ); | |
| 18 | } |

|  |  |
| --- | --- |
| 19 |  |
| 20 | set\_exception\_handler( 'handleException' ); | |

|  |  |
| --- | --- |
| 21 | ?> |

**Модифицируем Article**

Кроме создания  нового класса Category нужно также модифицировать класс  Article для работы с категориями. Ниже приводится код файла Article.php:

|  |  |  |
| --- | --- | --- |
| 001 | <?php | |
| 002 |  |

|  |  |
| --- | --- |
| 003 | /\*\* |
| 004 | \* Класс для управления статьями | |

|  |  |  |
| --- | --- | --- |
| 005 | \*/ | |
| 006 |  |

|  |  |  |
| --- | --- | --- |
| 007 | class Article | |
| 008 | { |

|  |  |  |
| --- | --- | --- |
| 009 | // Свойства | |
| 010 |  |

|  |  |
| --- | --- |
| 011 | /\*\* |
| 012 | \* @var int ID статьи из базы данных | |

|  |  |
| --- | --- |
| 013 | \*/ |
| 014 | public $id = null; | |

|  |  |
| --- | --- |
| 015 |  |
| 016 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 017 | \* @var int Дата публикации статьи | |
| 018 | \*/ |

|  |  |  |
| --- | --- | --- |
| 019 | public $publicationDate = null; | |
| 020 |  |

|  |  |
| --- | --- |
| 021 | /\*\* |
| 022 | \* @var int ID категории статьи | |

|  |  |
| --- | --- |
| 023 | \*/ |
| 024 | public $categoryId = null; | |

|  |  |
| --- | --- |
| 025 |  |
| 026 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 027 | \* @var string Полное название статьи | |
| 028 | \*/ |

|  |  |  |
| --- | --- | --- |
| 029 | public $title = null; | |
| 030 |  |

|  |  |
| --- | --- |
| 031 | /\*\* |
| 032 | \* @var string Резюме статьи | |

|  |  |
| --- | --- |
| 033 | \*/ |
| 034 | public $summary = null; | |

|  |  |
| --- | --- |
| 035 |  |
| 036 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 037 | \* @var string Содержание HTML статьи | |
| 038 | \*/ |

|  |  |  |
| --- | --- | --- |
| 039 | public $content = null; | |
| 040 |  |

|  |  |
| --- | --- |
| 041 |  |
| 042 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 043 | \* Устанавливаем свойства объекта с использованием значений из массива | |
| 044 | \* |

|  |  |  |
| --- | --- | --- |
| 045 | \* @param assoc Значения свойств | |
| 046 | \*/ |

|  |  |
| --- | --- |
| 047 |  |
| 048 | public function \_\_construct( $data=array() ) { | |

|  |  |
| --- | --- |
| 049 | if ( isset( $data['id'] ) ) $this->id = (int) $data['id']; |
| 050 | if ( isset( $data['publicationDate'] ) ) $this->publicationDate = (int) $data['publicationDate']; | |

|  |  |
| --- | --- |
| 051 | if ( isset( $data['categoryId'] ) ) $this->categoryId = (int) $data['categoryId']; |
| 052 | if ( isset( $data['title'] ) ) $this->title = preg\_replace ( "/[^\.\,\-\\_\'\"\@\?\!\:\$ a-zA-Z0-9()]/", "", $data['title'] ); |

|  |  |  |
| --- | --- | --- |
| 053 | if ( isset( $data['summary'] ) ) $this->summary = preg\_replace ( "/[^\.\,\-\\_\'\"\@\?\!\:\$ a-zA-Z0-9()]/", "", $data['summary'] ); | |
| 054 | if ( isset( $data['content'] ) ) $this->content = $data['content']; |

|  |  |  |
| --- | --- | --- |
| 055 | } | |
| 056 |  |

|  |  |
| --- | --- |
| 057 |  |
| 058 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 059 | \* Устанавливаем свойства объекта с использованием значений из формы | |
| 060 | \* |

|  |  |  |
| --- | --- | --- |
| 061 | \* @param assoc Значения из формы | |
| 062 | \*/ |

|  |  |
| --- | --- |
| 063 |  |
| 064 | public function storeFormValues ( $params ) { | |

|  |  |
| --- | --- |
| 065 |  |
| 066 | // Сохраняем все параметры | |

|  |  |  |
| --- | --- | --- |
| 067 | $this->\_\_construct( $params ); | |
| 068 |  |

|  |  |
| --- | --- |
| 069 | // Разбираем и сохраняем дату публикации |
| 070 | if ( isset($params['publicationDate']) ) { | |

|  |  |  |
| --- | --- | --- |
| 071 | $publicationDate = explode ( '-', $params['publicationDate'] ); | |
| 072 |  |

|  |  |
| --- | --- |
| 073 | if ( count($publicationDate) == 3 ) { |
| 074 | list ( $y, $m, $d ) = $publicationDate; | |

|  |  |  |
| --- | --- | --- |
| 075 | $this->publicationDate = mktime ( 0, 0, 0, $m, $d, $y ); | |
| 076 | } |

|  |  |  |
| --- | --- | --- |
| 077 | } | |
| 078 | } |

|  |  |
| --- | --- |
| 079 |  |
| 080 |  |

|  |  |
| --- | --- |
| 081 | /\*\* |
| 082 | \* Возвращаем объект Article соответствующий заданному ID | |

|  |  |
| --- | --- |
| 083 | \* |
| 084 | \* @param int ID статьи | |

|  |  |  |
| --- | --- | --- |
| 085 | \* @return Article|false Объект Article или false, если запись не найдена или в случае другой ошибки | |
| 086 | \*/ |

|  |  |
| --- | --- |
| 087 |  |
| 088 | public static function getById( $id ) { | |

|  |  |
| --- | --- |
| 089 | $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD ); |
| 090 | $sql = "SELECT \*, UNIX\_TIMESTAMP(publicationDate) AS publicationDate FROM articles WHERE id = :id"; | |

|  |  |
| --- | --- |
| 091 | $st = $conn->prepare( $sql ); |
| 092 | $st->bindValue( ":id", $id, PDO::PARAM\_INT ); | |

|  |  |
| --- | --- |
| 093 | $st->execute(); |
| 094 | $row = $st->fetch(); | |

|  |  |
| --- | --- |
| 095 | $conn = null; |
| 096 | if ( $row ) return new Article( $row ); | |

|  |  |  |
| --- | --- | --- |
| 097 | } | |
| 098 |  |

|  |  |
| --- | --- |
| 099 |  |
| 100 | /\*\* | |

|  |  |  |
| --- | --- | --- |
| 101 | \* Возвращает все (или диапазон) объекты Article из базы данных | |
| 102 | \* |

|  |  |  |
| --- | --- | --- |
| 103 | \* @param int Optional Количество возвращаемых строк (по умолчанию = all) | |
| 104 | \* @param int Optional Вернуть статьи только из категории с указанным ID |

|  |  |
| --- | --- |
| 105 | \* @param string Optional Столбц, по которому выполняется сортировка статей (по умолчанию = "publicationDate DESC") |
| 106 | \* @return Array|false Двух элементный массив: results => массив объектов Article; totalRows => общее количество строк |

|  |  |  |
| --- | --- | --- |
| 107 | \*/ | |
| 108 |  |

|  |  |  |
| --- | --- | --- |
| 109 | public static function getList( $numRows=1000000, $categoryId=null, $order="publicationDate DESC" ) { | |
| 110 | $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD ); |

|  |  |
| --- | --- |
| 111 | $categoryClause = $categoryId ? "WHERE categoryId = :categoryId" : ""; |
| 112 | $sql = "SELECT SQL\_CALC\_FOUND\_ROWS \*, UNIX\_TIMESTAMP(publicationDate) AS publicationDate | |

|  |  |
| --- | --- |
| 113 | FROM articles $categoryClause |
| 114 | ORDER BY " . mysql\_escape\_string($order) . " LIMIT :numRows"; | |

|  |  |
| --- | --- |
| 115 |  |
| 116 | $st = $conn->prepare( $sql ); | |

|  |  |
| --- | --- |
| 117 | $st->bindValue( ":numRows", $numRows, PDO::PARAM\_INT ); |
| 118 | if ( $categoryId ) $st->bindValue( ":categoryId", $categoryId, PDO::PARAM\_INT ); | |

|  |  |
| --- | --- |
| 119 | $st->execute(); |
| 120 | $list = array(); | |

|  |  |
| --- | --- |
| 121 |  |
| 122 | while ( $row = $st->fetch() ) { | |

|  |  |  |
| --- | --- | --- |
| 123 | $article = new Article( $row ); | |
| 124 | $list[] = $article; |

|  |  |  |
| --- | --- | --- |
| 125 | } | |
| 126 |  |

|  |  |  |
| --- | --- | --- |
| 127 | // Получаем общее количество статей, которые соответствуют критерию | |
| 128 | $sql = "SELECT FOUND\_ROWS() AS totalRows"; |

|  |  |  |
| --- | --- | --- |
| 129 | $totalRows = $conn->query( $sql )->fetch(); | |
| 130 | $conn = null; |

|  |  |  |
| --- | --- | --- |
| 131 | return ( array ( "results" => $list, "totalRows" => $totalRows[0] ) ); | |
| 132 | } |

|  |  |
| --- | --- |
| 133 |  |
| 134 |  |

|  |  |
| --- | --- |
| 135 | /\*\* |
| 136 | \* Вставляем текущий объек Article в базу данных, устанавливаем его ID. | |

|  |  |  |
| --- | --- | --- |
| 137 | \*/ | |
| 138 |  |

|  |  |  |
| --- | --- | --- |
| 139 | public function insert() { | |
| 140 |  |

|  |  |
| --- | --- |
| 141 | // Есть уже у объекта Article ID? |
| 142 | if ( !is\_null( $this->id ) ) trigger\_error ( "Article::insert(): Attempt to insert an Article object that already has its ID property set (to $this->id).", E\_USER\_ERROR ); | |

|  |  |
| --- | --- |
| 143 |  |
| 144 | // Вставляем статью | |

|  |  |
| --- | --- |
| 145 | $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD ); |
| 146 | $sql = "INSERT INTO articles ( publicationDate, categoryId, title, summary, content ) VALUES ( FROM\_UNIXTIME(:publicationDate), :categoryId, :title, :summary, :content )"; | |

|  |  |
| --- | --- |
| 147 | $st = $conn->prepare ( $sql ); |
| 148 | $st->bindValue( ":publicationDate", $this->publicationDate, PDO::PARAM\_INT ); | |

|  |  |  |
| --- | --- | --- |
| 149 | $st->bindValue( ":categoryId", $this->categoryId, PDO::PARAM\_INT ); | |
| 150 | $st->bindValue( ":title", $this->title, PDO::PARAM\_STR ); |

|  |  |
| --- | --- |
| 151 | $st->bindValue( ":summary", $this->summary, PDO::PARAM\_STR ); |
| 152 | $st->bindValue( ":content", $this->content, PDO::PARAM\_STR ); |

|  |  |
| --- | --- |
| 153 | $st->execute(); |
| 154 | $this->id = $conn->lastInsertId(); | |

|  |  |  |
| --- | --- | --- |
| 155 | $conn = null; | |
| 156 | } |

|  |  |
| --- | --- |
| 157 |  |
| 158 |  |

|  |  |
| --- | --- |
| 159 | /\*\* |
| 160 | \* Обновляем текущий объект Article в базе данных. | |

|  |  |  |
| --- | --- | --- |
| 161 | \*/ | |
| 162 |  |

|  |  |  |
| --- | --- | --- |
| 163 | public function update() { | |
| 164 |  |

|  |  |
| --- | --- |
| 165 | // У объекта Article есть ID? |
| 166 | if ( is\_null( $this->id ) ) trigger\_error ( "Article::update(): Attempt to update an Article object that does not have its ID property set.", E\_USER\_ERROR ); | |

|  |  |
| --- | --- |
| 167 |  |
| 168 | // Обновляем статью | |

|  |  |
| --- | --- |
| 169 | $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD ); |
| 170 | $sql = "UPDATE articles SET publicationDate=FROM\_UNIXTIME(:publicationDate), categoryId=:categoryId, title=:title, summary=:summary, content=:content WHERE id = :id"; | |

|  |  |
| --- | --- |
| 171 | $st = $conn->prepare ( $sql ); |
| 172 | $st->bindValue( ":publicationDate", $this->publicationDate, PDO::PARAM\_INT ); | |

|  |  |  |
| --- | --- | --- |
| 173 | $st->bindValue( ":categoryId", $this->categoryId, PDO::PARAM\_INT ); | |
| 174 | $st->bindValue( ":title", $this->title, PDO::PARAM\_STR ); |

|  |  |
| --- | --- |
| 175 | $st->bindValue( ":summary", $this->summary, PDO::PARAM\_STR ); |
| 176 | $st->bindValue( ":content", $this->content, PDO::PARAM\_STR ); |

|  |  |  |
| --- | --- | --- |
| 177 | $st->bindValue( ":id", $this->id, PDO::PARAM\_INT ); | |
| 178 | $st->execute(); |

|  |  |  |
| --- | --- | --- |
| 179 | $conn = null; | |
| 180 | } |

|  |  |
| --- | --- |
| 181 |  |
| 182 |  |

|  |  |
| --- | --- |
| 183 | /\*\* |
| 184 | \* Удаляем текущий объект Article из базы данных | |

|  |  |  |
| --- | --- | --- |
| 185 | \*/ | |
| 186 |  |

|  |  |  |
| --- | --- | --- |
| 187 | public function delete() { | |
| 188 |  |

|  |  |
| --- | --- |
| 189 | // У объекта Article есть ID? |
| 190 | if ( is\_null( $this->id ) ) trigger\_error ( "Article::delete(): Attempt to delete an Article object that does not have its ID property set.", E\_USER\_ERROR ); | |

|  |  |
| --- | --- |
| 191 |  |
| 192 | // Удаляем объект Article | |

|  |  |
| --- | --- |
| 193 | $conn = new PDO( DB\_DSN, DB\_USERNAME, DB\_PASSWORD ); |
| 194 | $st = $conn->prepare ( "DELETE FROM articles WHERE id = :id LIMIT 1" ); | |

|  |  |  |
| --- | --- | --- |
| 195 | $st->bindValue( ":id", $this->id, PDO::PARAM\_INT ); | |
| 196 | $st->execute(); |

|  |  |  |
| --- | --- | --- |
| 197 | $conn = null; | |
| 198 | } |

|  |  |  |
| --- | --- | --- |
| 199 |  | |
| 200 | } |

|  |  |
| --- | --- |
| 201 |  |
| 202 | ?> | |

Рассмотрим изменения в классе Article подробно:

* **Новое свойство categoryId**

Чтобы установить соответствие статьи и категории мы добавили свойство categoryId для хранения ID категории, которой принадлежит статья. Также мы модифицировали конструктор \_\_construct() для сохранения нового свойства categoryId в только что созданных объектах Article.

* **Изменение метода getList()**

Первоначальный вариант метода getList() возвращал все статьи в базе данных (или заданное количество записей). Так как нам нужно, чтобы CMS выводила список статей в определенной категории, модифицируем метод getList() для принятия опционального аргумента  $categoryId. Если он задан, то возвращаются статьи только из указанной категории.

Если задан $categoryId, создаем строку $categoryClause для выражения WHERE, чтобы вернуть только те статьи, в которых значение поля categoryId соответствует значению categoryId. Затем модифицируем выражения SQL SELECT для включения переменной $categoryClause. Также добавляем новый вызов к $st->bindValue() для привязки заданного $categoryId.

* **Изменяем методы insert() и update()**

Также мы модифицировали методы класса Article  insert() и update() для работы с новым полем categoryId. Изменили запросы SQL INSERT и UPDATE в каждом методе, и добавили новые вызовы к bindValue() для передачи свойства $categoryId в выражения SQL.

[В следующем уроке серии](http://ruseller.com/lessons.php?rub=37&id=1599) мы изменим шаблоны клиентской и серверной частей для работы с категориями.

**Модифицируем скрипт index.php клиентской части**

Добавим в скрипт клиентской части index.php функционал для отображения категорий. Вот так будет выглядеть готовый файл index.php в папке cms:

|  |  |  |
| --- | --- | --- |
| 01 | <?php | |
| 02 |  |

|  |  |
| --- | --- |
| 03 | require( "config.php" ); |
| 04 | $action = isset( $\_GET['action'] ) ? $\_GET['action'] : ""; | |

|  |  |
| --- | --- |
| 05 |  |
| 06 | switch ( $action ) { | |

|  |  |  |
| --- | --- | --- |
| 07 | case 'archive': | |
| 08 | archive(); |

|  |  |
| --- | --- |
| 09 | break; |
| 10 | case 'viewArticle': | |

|  |  |  |
| --- | --- | --- |
| 11 | viewArticle(); | |
| 12 | break; |

|  |  |
| --- | --- |
| 13 | default: |
| 14 | homepage(); | |

|  |  |
| --- | --- |
| 15 | } |
| 16 |  | |

|  |  |
| --- | --- |
| 17 | function archive() { |
| 18 | $results = array(); | |

|  |  |  |
| --- | --- | --- |
| 19 | $categoryId = ( isset( $\_GET['categoryId'] ) && $\_GET['categoryId'] ) ? (int)$\_GET['categoryId'] : null; | |
| 20 | $results['category'] = Category::getById( $categoryId ); |

|  |  |  |
| --- | --- | --- |
| 21 | $data = Article::getList( 100000, $results['category'] ? $results['category']->id : null ); | |
| 22 | $results['articles'] = $data['results']; |

|  |  |  |
| --- | --- | --- |
| 23 | $results['totalRows'] = $data['totalRows']; | |
| 24 | $data = Category::getList(); |

|  |  |
| --- | --- |
| 25 | $results['categories'] = array(); |
| 26 | foreach ( $data['results'] as $category ) $results['categories'][$category->id] = $category; | |

|  |  |  |
| --- | --- | --- |
| 27 | $results['pageHeading'] = $results['category'] ?  $results['category']->name : "Article Archive"; | |
| 28 | $results['pageTitle'] = $results['pageHeading'] . " | Widget News"; |

|  |  |  |
| --- | --- | --- |
| 29 | require( TEMPLATE\_PATH . "/archive.php" ); | |
| 30 | } |

|  |  |
| --- | --- |
| 31 |  |
| 32 | function viewArticle() { | |

|  |  |  |
| --- | --- | --- |
| 33 | if ( !isset($\_GET["articleId"]) || !$\_GET["articleId"] ) { | |
| 34 | homepage(); |

|  |  |  |
| --- | --- | --- |
| 35 | return; | |
| 36 | } |

|  |  |
| --- | --- |
| 37 |  |
| 38 | $results = array(); | |

|  |  |
| --- | --- |
| 39 | $results['article'] = Article::getById( (int)$\_GET["articleId"] ); |
| 40 | $results['category'] = Category::getById( $results['article']->categoryId ); | |

|  |  |  |
| --- | --- | --- |
| 41 | $results['pageTitle'] = $results['article']->title . " | Widget News"; | |
| 42 | require( TEMPLATE\_PATH . "/viewArticle.php" ); |

|  |  |
| --- | --- |
| 43 | } |
| 44 |  | |

|  |  |
| --- | --- |
| 45 | function homepage() { |
| 46 | $results = array(); |

|  |  |  |
| --- | --- | --- |
| 47 | $data = Article::getList( HOMEPAGE\_NUM\_ARTICLES ); | |
| 48 | $results['articles'] = $data['results']; |

|  |  |  |
| --- | --- | --- |
| 49 | $results['totalRows'] = $data['totalRows']; | |
| 50 | $data = Category::getList(); |

|  |  |
| --- | --- |
| 51 | $results['categories'] = array(); |
| 52 | foreach ( $data['results'] as $category ) $results['categories'][$category->id] = $category; | |

|  |  |
| --- | --- |
| 53 | $results['pageTitle'] = "Widget News"; |
| 54 | require( TEMPLATE\_PATH . "/homepage.php" ); | |

|  |  |
| --- | --- |
| 55 | } |
| 56 |  | |

|  |  |
| --- | --- |
| 57 | ?> |

Рассмотрим подробно изменения функций в index.php:

* **archive()**

Оригинальная функция archive() просто выводила список всех статей в CMS. Мы адаптируем функцию archive() так, чтобы она принимала опциональный параметр categoryId. Если categoryId задан, то функция возвращает соответствующий объект Category с помощью вызова Category::getById(), и если действие было успешным, передает ID категории в функцию Article::getList() для получения статей только из соответствующей категории.

Также добавляем код в функцию archive() для получения всех категорий в базе данных и сохранения их в  $results['categories'] с ID категории в качестве ключа. Шаблон страницы архива archive.php использует данный массив для вывода имени категории для каждой статьи.

Создаем переменную $results['pageHeading'], которая содержит либо имя категории (если categoryId было задано), либо текст "Article Archive". Данное значение будет отображаться в заголовке страницы архива. Также создаем переменную $results['pageTitle'] для использования в элементе  <title>. Просто для вывода названия сайта "Widget News" в заголовке страницы.

* **viewArticle()**

Здесь мы делаем только одно добавление: возвращаем объект Category соответствующий статье с помощью вызова метода Category::getById(),  передавая ему значение свойства статьи $categoryId. Полученный объект Category сохраняем $results['category']. Будем использовать данный объект для вывода названия категории в шаблоне viewArticle.php.

* **homepage()**

В функцию homepage() добавляем три строки. Данный код вызывает Category::getList() для получения всех категорий в CMS, затем сохраняем категории в $results['categories'] с ID категории в качестве ключа. То есть мы можем вывести названия категорий для каждой статьи на главной странице.

**Модифицируем скрипт серверной части admin.php**

Файл admin.php содержит все функции администратора для нашей CMS. Нужно сделать несколько изменений и добавлений в данном файле для обработки категорий.

Новый код файла admin.php, будет выглядеть следующим образом:

|  |  |  |
| --- | --- | --- |
| 001 | <?php | |
| 002 |  |

|  |  |  |
| --- | --- | --- |
| 003 | require( "config.php" ); | |
| 004 | session\_start(); |

|  |  |
| --- | --- |
| 005 | $action = isset( $\_GET['action'] ) ? $\_GET['action'] : ""; |
| 006 | $username = isset( $\_SESSION['username'] ) ? $\_SESSION['username'] : ""; | |

|  |  |
| --- | --- |
| 007 |  |
| 008 | if ( $action != "login" && $action != "logout" && !$username ) { | |

|  |  |  |
| --- | --- | --- |
| 009 | login(); | |
| 010 | exit; |

|  |  |
| --- | --- |
| 011 | } |
| 012 |  | |

|  |  |  |
| --- | --- | --- |
| 013 | switch ( $action ) { | |
| 014 | case 'login': |

|  |  |  |
| --- | --- | --- |
| 015 | login(); | |
| 016 | break; |

|  |  |  |
| --- | --- | --- |
| 017 | case 'logout': | |
| 018 | logout(); |

|  |  |
| --- | --- |
| 019 | break; |
| 020 | case 'newArticle': | |

|  |  |  |
| --- | --- | --- |
| 021 | newArticle(); | |
| 022 | break; |

|  |  |  |
| --- | --- | --- |
| 023 | case 'editArticle': | |
| 024 | editArticle(); |

|  |  |
| --- | --- |
| 025 | break; |
| 026 | case 'deleteArticle': | |

|  |  |  |
| --- | --- | --- |
| 027 | deleteArticle(); | |
| 028 | break; |

|  |  |  |
| --- | --- | --- |
| 029 | case 'listCategories': | |
| 030 | listCategories(); |

|  |  |
| --- | --- |
| 031 | break; |
| 032 | case 'newCategory': | |

|  |  |  |
| --- | --- | --- |
| 033 | newCategory(); | |
| 034 | break; |

|  |  |  |
| --- | --- | --- |
| 035 | case 'editCategory': | |
| 036 | editCategory(); |

|  |  |
| --- | --- |
| 037 | break; |
| 038 | case 'deleteCategory': | |

|  |  |  |
| --- | --- | --- |
| 039 | deleteCategory(); | |
| 040 | break; |

|  |  |
| --- | --- |
| 041 | default: |
| 042 | listArticles(); | |

|  |  |
| --- | --- |
| 043 | } |
| 044 |  | |

|  |  |
| --- | --- |
| 045 |  |
| 046 | function login() { | |

|  |  |
| --- | --- |
| 047 |  |
| 048 | $results = array(); | |

|  |  |  |
| --- | --- | --- |
| 049 | $results['pageTitle'] = "Admin Login | Widget News"; | |
| 050 |  |

|  |  |  |
| --- | --- | --- |
| 051 | if ( isset( $\_POST['login'] ) ) { | |
| 052 |  |

|  |  |  |
| --- | --- | --- |
| 053 | // User has posted the login form: attempt to log the user in | |
| 054 |  |

|  |  |  |
| --- | --- | --- |
| 055 | if ( $\_POST['username'] == ADMIN\_USERNAME && $\_POST['password'] == ADMIN\_PASSWORD ) { | |
| 056 |  |

|  |  |  |
| --- | --- | --- |
| 057 | // Login successful: Create a session and redirect to the admin homepage | |
| 058 | $\_SESSION['username'] = ADMIN\_USERNAME; |

|  |  |  |
| --- | --- | --- |
| 059 | header( "Location: admin.php" ); | |
| 060 |  |

|  |  |  |
| --- | --- | --- |
| 061 | } else { | |
| 062 |  |

|  |  |
| --- | --- |
| 063 | // Login failed: display an error message to the user |
| 064 | $results['errorMessage'] = "Incorrect username or password. Please try again."; | |

|  |  |  |
| --- | --- | --- |
| 065 | require( TEMPLATE\_PATH . "/admin/loginForm.php" ); | |
| 066 | } |

|  |  |
| --- | --- |
| 067 |  |
| 068 | } else { | |

|  |  |
| --- | --- |
| 069 |  |
| 070 | // User has not posted the login form yet: display the form | |

|  |  |  |
| --- | --- | --- |
| 071 | require( TEMPLATE\_PATH . "/admin/loginForm.php" ); | |
| 072 | } |

|  |  |  |
| --- | --- | --- |
| 073 |  | |
| 074 | } |

|  |  |
| --- | --- |
| 075 |  |
| 076 |  |

|  |  |
| --- | --- |
| 077 | function logout() { |
| 078 | unset( $\_SESSION['username'] ); | |

|  |  |  |
| --- | --- | --- |
| 079 | header( "Location: admin.php" ); | |
| 080 | } |

|  |  |
| --- | --- |
| 081 |  |
| 082 |  |

|  |  |  |
| --- | --- | --- |
| 083 | function newArticle() { | |
| 084 |  |

|  |  |
| --- | --- |
| 085 | $results = array(); |
| 086 | $results['pageTitle'] = "New Article"; | |

|  |  |  |
| --- | --- | --- |
| 087 | $results['formAction'] = "newArticle"; | |
| 088 |  |

|  |  |  |
| --- | --- | --- |
| 089 | if ( isset( $\_POST['saveChanges'] ) ) { | |
| 090 |  |

|  |  |  |
| --- | --- | --- |
| 091 | // User has posted the article edit form: save the new article | |
| 092 | $article = new Article; |

|  |  |  |
| --- | --- | --- |
| 093 | $article->storeFormValues( $\_POST ); | |
| 094 | $article->insert(); |

|  |  |  |
| --- | --- | --- |
| 095 | header( "Location: admin.php?status=changesSaved" ); | |
| 096 |  |

|  |  |  |
| --- | --- | --- |
| 097 | } elseif ( isset( $\_POST['cancel'] ) ) { | |
| 098 |  |

|  |  |  |
| --- | --- | --- |
| 099 | // User has cancelled their edits: return to the article list | |
| 100 | header( "Location: admin.php" ); |

|  |  |  |
| --- | --- | --- |
| 101 | } else { | |
| 102 |  |

|  |  |  |
| --- | --- | --- |
| 103 | // User has not posted the article edit form yet: display the form | |
| 104 | $results['article'] = new Article; |

|  |  |
| --- | --- |
| 105 | $data = Category::getList(); |
| 106 | $results['categories'] = $data['results']; | |

|  |  |  |
| --- | --- | --- |
| 107 | require( TEMPLATE\_PATH . "/admin/editArticle.php" ); | |
| 108 | } |

|  |  |  |
| --- | --- | --- |
| 109 |  | |
| 110 | } |

|  |  |
| --- | --- |
| 111 |  |
| 112 |  |

|  |  |  |
| --- | --- | --- |
| 113 | function editArticle() { | |
| 114 |  |

|  |  |
| --- | --- |
| 115 | $results = array(); |
| 116 | $results['pageTitle'] = "Edit Article"; | |

|  |  |  |
| --- | --- | --- |
| 117 | $results['formAction'] = "editArticle"; | |
| 118 |  |

|  |  |  |
| --- | --- | --- |
| 119 | if ( isset( $\_POST['saveChanges'] ) ) { | |
| 120 |  |

|  |  |  |
| --- | --- | --- |
| 121 | // User has posted the article edit form: save the article changes | |
| 122 |  |

|  |  |  |
| --- | --- | --- |
| 123 | if ( !$article = Article::getById( (int)$\_POST['articleId'] ) ) { | |
| 124 | header( "Location: admin.php?error=articleNotFound" ); |

|  |  |  |
| --- | --- | --- |
| 125 | return; | |
| 126 | } |

|  |  |
| --- | --- |
| 127 |  |
| 128 | $article->storeFormValues( $\_POST ); | |

|  |  |
| --- | --- |
| 129 | $article->update(); |
| 130 | header( "Location: admin.php?status=changesSaved" ); | |

|  |  |
| --- | --- |
| 131 |  |
| 132 | } elseif ( isset( $\_POST['cancel'] ) ) { | |

|  |  |
| --- | --- |
| 133 |  |
| 134 | // User has cancelled their edits: return to the article list | |

|  |  |  |
| --- | --- | --- |
| 135 | header( "Location: admin.php" ); | |
| 136 | } else { |

|  |  |
| --- | --- |
| 137 |  |
| 138 | // User has not posted the article edit form yet: display the form | |

|  |  |  |
| --- | --- | --- |
| 139 | $results['article'] = Article::getById( (int)$\_GET['articleId'] ); | |
| 140 | $data = Category::getList(); |

|  |  |
| --- | --- |
| 141 | $results['categories'] = $data['results']; |
| 142 | require( TEMPLATE\_PATH . "/admin/editArticle.php" ); | |

|  |  |  |
| --- | --- | --- |
| 143 | } | |
| 144 |  |

|  |  |
| --- | --- |
| 145 | } |
| 146 |  | |

|  |  |
| --- | --- |
| 147 |  |
| 148 | function deleteArticle() { | |

|  |  |
| --- | --- |
| 149 |  |
| 150 | if ( !$article = Article::getById( (int)$\_GET['articleId'] ) ) { | |

|  |  |  |
| --- | --- | --- |
| 151 | header( "Location: admin.php?error=articleNotFound" ); | |
| 152 | return; |

|  |  |  |
| --- | --- | --- |
| 153 | } | |
| 154 |  |

|  |  |
| --- | --- |
| 155 | $article->delete(); |
| 156 | header( "Location: admin.php?status=articleDeleted" ); | |

|  |  |
| --- | --- |
| 157 | } |
| 158 |  | |

|  |  |
| --- | --- |
| 159 |  |
| 160 | function listArticles() { | |

|  |  |
| --- | --- |
| 161 | $results = array(); |
| 162 | $data = Article::getList(); | |

|  |  |
| --- | --- |
| 163 | $results['articles'] = $data['results']; |
| 164 | $results['totalRows'] = $data['totalRows']; | |

|  |  |
| --- | --- |
| 165 | $data = Category::getList(); |
| 166 | $results['categories'] = array(); | |

|  |  |  |
| --- | --- | --- |
| 167 | foreach ( $data['results'] as $category ) $results['categories'][$category->id] = $category; | |
| 168 | $results['pageTitle'] = "All Articles"; |

|  |  |
| --- | --- |
| 169 |  |
| 170 | if ( isset( $\_GET['error'] ) ) { | |

|  |  |  |
| --- | --- | --- |
| 171 | if ( $\_GET['error'] == "articleNotFound" ) $results['errorMessage'] = "Error: Article not found."; | |
| 172 | } |

|  |  |
| --- | --- |
| 173 |  |
| 174 | if ( isset( $\_GET['status'] ) ) { | |

|  |  |
| --- | --- |
| 175 | if ( $\_GET['status'] == "changesSaved" ) $results['statusMessage'] = "Your changes have been saved."; |
| 176 | if ( $\_GET['status'] == "articleDeleted" ) $results['statusMessage'] = "Article deleted."; |

|  |  |  |
| --- | --- | --- |
| 177 | } | |
| 178 |  |

|  |  |  |
| --- | --- | --- |
| 179 | require( TEMPLATE\_PATH . "/admin/listArticles.php" ); | |
| 180 | } |

|  |  |
| --- | --- |
| 181 |  |
| 182 |  |

|  |  |  |
| --- | --- | --- |
| 183 | function listCategories() { | |
| 184 | $results = array(); |

|  |  |
| --- | --- |
| 185 | $data = Category::getList(); |
| 186 | $results['categories'] = $data['results']; | |

|  |  |
| --- | --- |
| 187 | $results['totalRows'] = $data['totalRows']; |
| 188 | $results['pageTitle'] = "Article Categories"; | |

|  |  |
| --- | --- |
| 189 |  |
| 190 | if ( isset( $\_GET['error'] ) ) { | |

|  |  |
| --- | --- |
| 191 | if ( $\_GET['error'] == "categoryNotFound" ) $results['errorMessage'] = "Error: Category not found."; |
| 192 | if ( $\_GET['error'] == "categoryContainsArticles" ) $results['errorMessage'] = "Error: Category contains articles. Delete the articles, or assign them to another category, before deleting this category."; |

|  |  |  |
| --- | --- | --- |
| 193 | } | |
| 194 |  |

|  |  |
| --- | --- |
| 195 | if ( isset( $\_GET['status'] ) ) { |
| 196 | if ( $\_GET['status'] == "changesSaved" ) $results['statusMessage'] = "Your changes have been saved."; | |

|  |  |  |
| --- | --- | --- |
| 197 | if ( $\_GET['status'] == "categoryDeleted" ) $results['statusMessage'] = "Category deleted."; | |
| 198 | } |

|  |  |
| --- | --- |
| 199 |  |
| 200 | require( TEMPLATE\_PATH . "/admin/listCategories.php" ); | |

|  |  |
| --- | --- |
| 201 | } |
| 202 |  | |

|  |  |
| --- | --- |
| 203 |  |
| 204 | function newCategory() { | |

|  |  |
| --- | --- |
| 205 |  |
| 206 | $results = array(); | |

|  |  |  |
| --- | --- | --- |
| 207 | $results['pageTitle'] = "New Article Category"; | |
| 208 | $results['formAction'] = "newCategory"; |

|  |  |
| --- | --- |
| 209 |  |
| 210 | if ( isset( $\_POST['saveChanges'] ) ) { | |

|  |  |
| --- | --- |
| 211 |  |
| 212 | // User has posted the category edit form: save the new category | |

|  |  |
| --- | --- |
| 213 | $category = new Category; |
| 214 | $category->storeFormValues( $\_POST ); | |

|  |  |
| --- | --- |
| 215 | $category->insert(); |
| 216 | header( "Location: admin.php?action=listCategories&status=changesSaved" ); | |

|  |  |
| --- | --- |
| 217 |  |
| 218 | } elseif ( isset( $\_POST['cancel'] ) ) { | |

|  |  |
| --- | --- |
| 219 |  |
| 220 | // User has cancelled their edits: return to the category list | |

|  |  |  |
| --- | --- | --- |
| 221 | header( "Location: admin.php?action=listCategories" ); | |
| 222 | } else { |

|  |  |
| --- | --- |
| 223 |  |
| 224 | // User has not posted the category edit form yet: display the form | |

|  |  |
| --- | --- |
| 225 | $results['category'] = new Category; |
| 226 | require( TEMPLATE\_PATH . "/admin/editCategory.php" ); | |

|  |  |  |
| --- | --- | --- |
| 227 | } | |
| 228 |  |

|  |  |
| --- | --- |
| 229 | } |
| 230 |  | |

|  |  |
| --- | --- |
| 231 |  |
| 232 | function editCategory() { | |

|  |  |
| --- | --- |
| 233 |  |
| 234 | $results = array(); | |

|  |  |  |
| --- | --- | --- |
| 235 | $results['pageTitle'] = "Edit Article Category"; | |
| 236 | $results['formAction'] = "editCategory"; |

|  |  |
| --- | --- |
| 237 |  |
| 238 | if ( isset( $\_POST['saveChanges'] ) ) { | |

|  |  |
| --- | --- |
| 239 |  |
| 240 | // User has posted the category edit form: save the category changes | |

|  |  |
| --- | --- |
| 241 |  |
| 242 | if ( !$category = Category::getById( (int)$\_POST['categoryId'] ) ) { | |

|  |  |  |
| --- | --- | --- |
| 243 | header( "Location: admin.php?action=listCategories&error=categoryNotFound" ); | |
| 244 | return; |

|  |  |  |
| --- | --- | --- |
| 245 | } | |
| 246 |  |

|  |  |  |
| --- | --- | --- |
| 247 | $category->storeFormValues( $\_POST ); | |
| 248 | $category->update(); |

|  |  |  |
| --- | --- | --- |
| 249 | header( "Location: admin.php?action=listCategories&status=changesSaved" ); | |
| 250 |  |

|  |  |  |
| --- | --- | --- |
| 251 | } elseif ( isset( $\_POST['cancel'] ) ) { | |
| 252 |  |

|  |  |  |
| --- | --- | --- |
| 253 | // User has cancelled their edits: return to the category list | |
| 254 | header( "Location: admin.php?action=listCategories" ); |

|  |  |  |
| --- | --- | --- |
| 255 | } else { | |
| 256 |  |

|  |  |
| --- | --- |
| 257 | // User has not posted the category edit form yet: display the form |
| 258 | $results['category'] = Category::getById( (int)$\_GET['categoryId'] ); | |

|  |  |  |
| --- | --- | --- |
| 259 | require( TEMPLATE\_PATH . "/admin/editCategory.php" ); | |
| 260 | } |

|  |  |  |
| --- | --- | --- |
| 261 |  | |
| 262 | } |

|  |  |
| --- | --- |
| 263 |  |
| 264 |  |

|  |  |  |
| --- | --- | --- |
| 265 | function deleteCategory() { | |
| 266 |  |

|  |  |
| --- | --- |
| 267 | if ( !$category = Category::getById( (int)$\_GET['categoryId'] ) ) { |
| 268 | header( "Location: admin.php?action=listCategories&error=categoryNotFound" ); | |

|  |  |  |
| --- | --- | --- |
| 269 | return; | |
| 270 | } |

|  |  |
| --- | --- |
| 271 |  |
| 272 | $articles = Article::getList( 1000000, $category->id ); | |

|  |  |
| --- | --- |
| 273 |  |
| 274 | if ( $articles['totalRows'] > 0 ) { | |

|  |  |  |
| --- | --- | --- |
| 275 | header( "Location: admin.php?action=listCategories&error=categoryContainsArticles" ); | |
| 276 | return; |

|  |  |  |
| --- | --- | --- |
| 277 | } | |
| 278 |  |

|  |  |
| --- | --- |
| 279 | $category->delete(); |
| 280 | header( "Location: admin.php?action=listCategories&status=categoryDeleted" ); | |

|  |  |
| --- | --- |
| 281 | } |
| 282 |  | |

|  |  |
| --- | --- |
| 283 | ?> |

Расмотрим изменения в файле admin.php подробно:

* **Добавления Additions to the switch block**

Нам нужно добавить новый функционал в файл admin.php для обработки вывода, создания, редактирования и удаления категорий. Поэтому мы добавляем блоки listCategories, newCategory, editCategory и deleteCategory case в выражение switch в начале файла. Данные блоки вызывают различные функции для обработки категорий.

* **Изменения в функциях newArticle(), editArticle() и listArticles()**

Нужно сделать небольшие изменения в каждой из перечисленных функций для получения списка категорий в базе данных и использования в форме редактирования и создания статьи. Полученный список категорий хранится в переменной $results['categories']. Для функции listArticles() мы также используем в качестве ключа ID категории для облегчения применения списка в шаблоне listArticles.php.

* **listCategories()**

Данная функция выводит список всех категорий для администратора. Она работает так же как и функция listArticles(). Все категории извлекаются из базы данных с помощью функции  Category::getList(), затем сохраняются в массиве $results['categories']. Также записывается общее количество категорий в переменной $results['totalRows'] и сохраняется название страницы в переменной $results['pageTitle']. Затем производится проверка на наличие кодов статуса в строке запроса и устанавливаются значения переменных $results['errorMessage'] или $results['statusMessage'] соответственно. В завершение включается шаблон listCategories.php для вывода списка категорий.

* **newCategory()**

Функция для добавления новых категорий в базу данных, так же как и newArticle() добавляет новую статью. Если пользователь передал форму редактирования категории, то функция создает новый объект Category object, наполняет его данными из формы, вызывает функцию insert() для вставки категории в базу данных, и перенаправляет на страницу со списком категорий, выводя сообщение о сохранении изменений. Если пользователь нажимает кнопку Cancel в форме, то функция просто перенаправляет на список категорий. Если  пользователь еще не отправил форму, то функция создает новый пустой объект Category для использования в форме, сохраняет его в $results['category'] и включает шаблон editCategory.php для вывода формы редактирования категории.

* **editCategory()**

Данная функция редактирует существующую категорию в базе данных, позволяя пользователю менять название категории и/или описание.она действует по такому же сценарию, что и editArticle().  Если форма редактирования отправлена, загружаем категорию из базы данных, сохраняем новые значения формы в объекте Category, и обновляем категорию в базе данных с помощью метода update(). Если пользователь отказался от редактирования, то функция перенаправляет его на список категорий. Если пользователь еще не отправил форму, то функция загружает категорию, указанную параметром categoryId в строке запроса, сохраняет ее в переменной $results['category'] и включает шаблон editCategory.php для вывода заполненной формы редактирования.

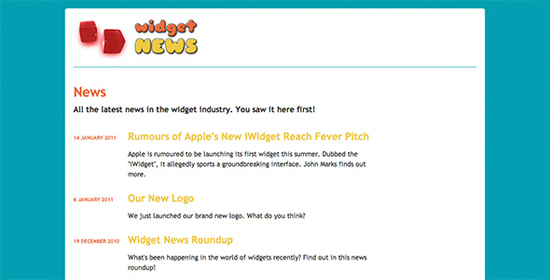
* **deleteCategory()**

Последняя новая функция позволяет удалять категорию из базы данных. Она вызывается, когда пользователь нажимает ссылку Delete This Category (Удалить категорию ) на странице редактирования категорий. Сначала функция получает категорию, заданную параметром запроса categoryId  (если категории нет, то выводится сообщение об ошибке). Затем производится проверка наличия статьей в категории. Если статьи есть , то выводится сообщение об ошибке и происходит выход из функции. Если статей в категории нет, то функция удаляет категорию и перенаправляет на страницу со списком категорий, выводя сообщение об удалении категории.

[В следующем уроке серии](http://ruseller.com/lessons.php?rub=37&id=1601) мы рассмотрим изменения в шаблонах нашей CMS? которые надо сделать для добавления функционала категорий.

## Клиентская часть

Сначала изменим клиентскую часть, для отображения категории статьи в общем списке и на страницах архива.



### homepage.php

Мы сделаем небольшие изменения в шаблоне homepage.php, который выводит главную страницу. Нужно, чтобы категория каждой статьи выводилась ниже названия. Новый код файла cms/templates/homepage.php будет выглядеть так:

|  |  |  |
| --- | --- | --- |
| 01 | <?php include "templates/include/header.php" ?> | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | <ul id="headlines"> | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | <?php foreach ( $results['articles'] as $article ) { ?> | |
| 06 |  |

|  |  |
| --- | --- |
| 07 | <li> |
| 08 | <h2> | |

|  |  |  |
| --- | --- | --- |
| 09 | <span class="pubDate"><?php echo date('j F', $article->publicationDate)?></span><a href=".?action=viewArticle&amp;articleId=<?php echo $article->id?>"><?php echo htmlspecialchars( $article->title )?></a> | |
| 10 | <?php if ( $article->categoryId ) { ?> |

|  |  |  |
| --- | --- | --- |
| 11 | <span class="category">in <a href=".?action=archive&amp;categoryId=<?php echo $article->categoryId?>"><?php echo htmlspecialchars( $results['categories'][$article->categoryId]->name )?></a></span> | |
| 12 | <?php } ?> |

|  |  |
| --- | --- |
| 13 | </h2> |
| 14 | <p class="summary"><?php echo htmlspecialchars( $article->summary )?></p> | |

|  |  |  |
| --- | --- | --- |
| 15 | </li> | |
| 16 |  |

|  |  |  |
| --- | --- | --- |
| 17 | <?php } ?> | |
| 18 |  |

|  |  |  |
| --- | --- | --- |
| 19 | </ul> | |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | <p><a href="./?action=archive">Article Archive</a></p> | |
| 22 |  |

|  |  |
| --- | --- |
| 23 | <?php include "templates/include/footer.php" ?> |

Мы вставили проверку вхождения статьи в категорию с помощью свойства $categoryId. Если статья имеет идентификатор категории, то вставляем элемент <span> с именем категории в код страницы. Для решения используем объект Category из массива $results['categories'], который выбирается с помощью ID категории, а затем выводим свойство name.

Код также помещает имя категории в ссылку, которая ведет на страницу архива для категории, так что пользователь легко может просмотреть все статьи из той же категории.

### archive.php

Шаблон archive.php выводит архив статей. В базовом варианте нашей CMS архив просто выводил список всех статей. Однако, теперь функция archive() в index.php может выводить список статей только из указанной категории. То есть нам нужно внести изменения в шаблон для реализации новых возможностей.

Новый код файла cms/templates/archive.php:

|  |  |  |
| --- | --- | --- |
| 01 | <?php include "templates/include/header.php" ?> | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | <h1><?php echo htmlspecialchars( $results['pageHeading'] ) ?></h1> | |
| 04 | <?php if ( $results['category'] ) { ?> |

|  |  |  |
| --- | --- | --- |
| 05 | <h3 class="categoryDescription"><?php echo htmlspecialchars( $results['category']->description ) ?></h3> | |
| 06 | <?php } ?> |

|  |  |
| --- | --- |
| 07 |  |
| 08 | <ul id="headlines" class="archive"> | |

|  |  |
| --- | --- |
| 09 |  |
| 10 | <?php foreach ( $results['articles'] as $article ) { ?> | |

|  |  |
| --- | --- |
| 11 |  |
| 12 | <li> | |

|  |  |
| --- | --- |
| 13 | <h2> |
| 14 | <span class="pubDate"><?php echo date('j F Y', $article->publicationDate)?></span><a href=".?action=viewArticle&amp;articleId=<?php echo $article->id?>"><?php echo htmlspecialchars( $article->title )?></a> | |

|  |  |
| --- | --- |
| 15 | <?php if ( !$results['category'] && $article->categoryId ) { ?> |
| 16 | <span class="category">in <a href=".?action=archive&amp;categoryId=<?php echo $article->categoryId?>"><?php echo htmlspecialchars( $results['categories'][$article->categoryId]->name ) ?></a></span> | |

|  |  |  |
| --- | --- | --- |
| 17 | <?php } ?> | |
| 18 | </h2> |

|  |  |  |
| --- | --- | --- |
| 19 | <p class="summary"><?php echo htmlspecialchars( $article->summary )?></p> | |
| 20 | </li> |

|  |  |
| --- | --- |
| 21 |  |
| 22 | <?php } ?> | |

|  |  |
| --- | --- |
| 23 |  |
| 24 | </ul> | |

|  |  |
| --- | --- |
| 25 |  |
| 26 | <p><?php echo $results['totalRows']?> article<?php echo ( $results['totalRows'] != 1 ) ? 's' : '' ?> in total.</p> | |

|  |  |
| --- | --- |
| 27 |  |
| 28 | <p><a href="./">Return to Homepage</a></p> | |

|  |  |
| --- | --- |
| 29 |  |
| 30 | <?php include "templates/include/footer.php" ?> | |

Мы изменили заголовок <h1> для вывода значения $results['pageHeading'], а не простой надписи "Article Archive" (архив статей). Новая функция archive() динамически формирует заголовок в зависимости от просматриваемой категории.

Затем мы добавляем код для проверки того, что нужно выводить список для определенной категории, а не всех статей в базе данных. Для этого просматриваем переменную $results['category']. Если она имеется, то нужно выводить список для категории, и код формирует в заголовке описание категории.

Цикл, который формирует вывод статей также изменяется. Новый код запускается, если нужно выводить список всех статей и текущая статья включена в категорию. В таком случае добавляется элемент <span>, который содержит имя категории в виде ссылки на страницу архива категории, чтобы пользователь мог просмотреть все статьи из нее.

### viewArticle.php

Шаблон viewArticle.php выводит страницу статьи. Новый код файла cms/templates/viewArticle.php:

|  |  |  |
| --- | --- | --- |
| 01 | <?php include "templates/include/header.php" ?> | |
| 02 |  |

|  |  |
| --- | --- |
| 03 | <h1 style="width: 75%;"><?php echo htmlspecialchars( $results['article']->title )?></h1> |
| 04 | <div style="width: 75%; font-style: italic;"><?php echo htmlspecialchars( $results['article']->summary )?></div> |

|  |  |
| --- | --- |
| 05 | <div style="width: 75%;"><?php echo $results['article']->content?></div> |
| 06 | <p class="pubDate">Published on <?php echo date('j F Y', $results['article']->publicationDate)?> |

|  |  |
| --- | --- |
| 07 | <?php if ( $results['category'] ) { ?> |
| 08 | in <a href="./?action=archive&amp;categoryId=<?php echo $results['category']->id?>"><?php echo htmlspecialchars( $results['category']->name ) ?></a> | |

|  |  |
| --- | --- |
| 09 | <?php } ?> |
| 10 | </p> |

|  |  |
| --- | --- |
| 11 |  |
| 12 | <p><a href="./">Return to Homepage</a></p> | |

|  |  |
| --- | --- |
| 13 |  |
| 14 | <?php include "templates/include/footer.php" ?> | |

Здесь вносится небольшое добавление для вывода названия категории внизу. Название категории является ссылкой на соответствующую страницу архива.

### Стили клиентской части

Также нужно внести изменения в таблицу стилей style.css для формирования внешнего вида имен категорий и описаний на главной странице и в архиве.

Новый код файла style.css из папки cms:

|  |  |  |
| --- | --- | --- |
| 001 | /\* Style the body and outer container \*/ | |
| 002 |  |

|  |  |
| --- | --- |
| 003 | body { |
| 004 | margin: 0; | |

|  |  |
| --- | --- |
| 005 | color: #333; |
| 006 | background-color: #00a0b0; | |

|  |  |  |
| --- | --- | --- |
| 007 | font-family: "Trebuchet MS", Arial, Helvetica, sans-serif; | |
| 008 | line-height: 1.5em; |

|  |  |
| --- | --- |
| 009 | } |
| 010 |  | |

|  |  |
| --- | --- |
| 011 | #container { |
| 012 | width: 960px; | |

|  |  |
| --- | --- |
| 013 | background: #fff; |
| 014 | margin: 20px auto; | |

|  |  |
| --- | --- |
| 015 | padding: 20px; |
| 016 | -moz-border-radius: 5px; | |

|  |  |  |
| --- | --- | --- |
| 017 | -webkit-border-radius: 5px; | |
| 018 | border-radius: 5px; |

|  |  |
| --- | --- |
| 019 | } |
| 020 |  | |

|  |  |
| --- | --- |
| 021 |  |
| 022 | /\* The logo and footer \*/ | |

|  |  |
| --- | --- |
| 023 |  |
| 024 | #logo { | |

|  |  |  |
| --- | --- | --- |
| 025 | display: block; | |
| 026 | width: 300px; |

|  |  |  |
| --- | --- | --- |
| 027 | padding: 0 660px 20px 0; | |
| 028 | border: none; |

|  |  |  |
| --- | --- | --- |
| 029 | border-bottom: 1px solid #00a0b0; | |
| 030 | margin-bottom: 40px; |

|  |  |
| --- | --- |
| 031 | } |
| 032 |  | |

|  |  |
| --- | --- |
| 033 | #footer { |
| 034 | border-top: 1px solid #00a0b0; | |

|  |  |
| --- | --- |
| 035 | margin-top: 40px; |
| 036 | padding: 20px 0 0 0; | |

|  |  |  |
| --- | --- | --- |
| 037 | font-size: .8em; | |
| 038 | } |

|  |  |
| --- | --- |
| 039 |  |
| 040 |  |

|  |  |  |
| --- | --- | --- |
| 041 | /\* Headings \*/ | |
| 042 |  |

|  |  |
| --- | --- |
| 043 | h1 { |
| 044 | color: #eb6841; | |

|  |  |  |
| --- | --- | --- |
| 045 | margin-bottom: 30px; | |
| 046 | line-height: 1.2em; |

|  |  |
| --- | --- |
| 047 | } |
| 048 |  | |

|  |  |
| --- | --- |
| 049 | h2, h2 a { |
| 050 | color: #edc951; | |

|  |  |
| --- | --- |
| 051 | } |
| 052 |  | |

|  |  |
| --- | --- |
| 053 | h2 a { |
| 054 | text-decoration: none; | |

|  |  |
| --- | --- |
| 055 | } |
| 056 |  | |

|  |  |  |
| --- | --- | --- |
| 057 | h3.categoryDescription { | |
| 058 | margin-top: -20px; |

|  |  |  |
| --- | --- | --- |
| 059 | margin-bottom: 40px; | |
| 060 | } |

|  |  |
| --- | --- |
| 061 |  |
| 062 |  |

|  |  |  |
| --- | --- | --- |
| 063 | /\* Article headlines \*/ | |
| 064 |  |

|  |  |
| --- | --- |
| 065 | #headlines { |
| 066 | list-style: none; | |

|  |  |  |
| --- | --- | --- |
| 067 | padding-left: 0; | |
| 068 | width: 75%; |

|  |  |
| --- | --- |
| 069 | } |
| 070 |  | |

|  |  |
| --- | --- |
| 071 | #headlines li { |
| 072 | margin-bottom: 2em; | |

|  |  |  |
| --- | --- | --- |
| 073 | clear: both; | |
| 074 | } |

|  |  |
| --- | --- |
| 075 |  |
| 076 | .pubDate { | |

|  |  |  |
| --- | --- | --- |
| 077 | font-size: .8em; | |
| 078 | color: #eb6841; |

|  |  |  |
| --- | --- | --- |
| 079 | text-transform: uppercase; | |
| 080 | } |

|  |  |
| --- | --- |
| 081 |  |
| 082 | #headlines .pubDate { | |

|  |  |  |
| --- | --- | --- |
| 083 | display: block; | |
| 084 | width: 100px; |

|  |  |  |
| --- | --- | --- |
| 085 | padding-top: 4px; | |
| 086 | float: left; |

|  |  |
| --- | --- |
| 087 | font-size: .5em; |
| 088 | vertical-align: middle; | |

|  |  |
| --- | --- |
| 089 | } |
| 090 |  | |

|  |  |  |
| --- | --- | --- |
| 091 | #headlines.archive .pubDate { | |
| 092 | width: 130px; |

|  |  |
| --- | --- |
| 093 | } |
| 094 |  | |

|  |  |
| --- | --- |
| 095 | .summary { |
| 096 | padding-left: 100px; | |

|  |  |
| --- | --- |
| 097 | } |
| 098 |  | |

|  |  |  |
| --- | --- | --- |
| 099 | #headlines.archive .summary { | |
| 100 | padding-left: 130px; |

|  |  |
| --- | --- |
| 101 | } |
| 102 |  | |

|  |  |
| --- | --- |
| 103 | .category { |
| 104 | font-style: italic; | |

|  |  |  |
| --- | --- | --- |
| 105 | font-weight: normal; | |
| 106 | font-size: 60%; |

|  |  |
| --- | --- |
| 107 | color: #999; |
| 108 | display: block; | |

|  |  |  |
| --- | --- | --- |
| 109 | line-height: 2em; | |
| 110 | } |

|  |  |
| --- | --- |
| 111 |  |
| 112 | .category a { | |

|  |  |
| --- | --- |
| 113 | color: #999; |
| 114 | text-decoration: underline; | |

|  |  |
| --- | --- |
| 115 | } |
| 116 |  | |

|  |  |
| --- | --- |
| 117 |  |
| 118 | /\* "You are logged in..." header on admin pages \*/ | |

|  |  |
| --- | --- |
| 119 |  |
| 120 | #adminHeader { | |

|  |  |
| --- | --- |
| 121 | width: 940px; |
| 122 | padding: 0 10px; | |

|  |  |  |
| --- | --- | --- |
| 123 | border-bottom: 1px solid #00a0b0; | |
| 124 | margin: -30px 0 40px 0; |

|  |  |  |
| --- | --- | --- |
| 125 | font-size: 0.8em; | |
| 126 | } |

|  |  |
| --- | --- |
| 127 |  |
| 128 |  |

|  |  |  |
| --- | --- | --- |
| 129 | /\* Style the form with a coloured background, along with curved corners and a drop shadow \*/ | |
| 130 |  |

|  |  |
| --- | --- |
| 131 | form { |
| 132 | margin: 20px auto; | |

|  |  |  |
| --- | --- | --- |
| 133 | padding: 40px 20px; | |
| 134 | overflow: auto; |

|  |  |
| --- | --- |
| 135 | background: #fff4cf; |
| 136 | border: 1px solid #666; | |

|  |  |
| --- | --- |
| 137 | -moz-border-radius: 5px; |
| 138 | -webkit-border-radius: 5px; | |

|  |  |
| --- | --- |
| 139 | border-radius: 5px; |
| 140 | -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |

|  |  |  |
| --- | --- | --- |
| 141 | -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |
| 142 | box-shadow: 0 0 .5em rgba(0, 0, 0, .8); |

|  |  |
| --- | --- |
| 143 | } |
| 144 |  | |

|  |  |
| --- | --- |
| 145 |  |
| 146 | /\* Give form elements consistent margin, padding and line height \*/ | |

|  |  |
| --- | --- |
| 147 |  |
| 148 | form ul { | |

|  |  |  |
| --- | --- | --- |
| 149 | list-style: none; | |
| 150 | margin: 0; |

|  |  |  |
| --- | --- | --- |
| 151 | padding: 0; | |
| 152 | } |

|  |  |
| --- | --- |
| 153 |  |
| 154 | form ul li { | |

|  |  |  |
| --- | --- | --- |
| 155 | margin: .9em 0 0 0; | |
| 156 | padding: 0; |

|  |  |
| --- | --- |
| 157 | } |
| 158 |  | |

|  |  |
| --- | --- |
| 159 | form \* { |
| 160 | line-height: 1em; | |

|  |  |
| --- | --- |
| 161 | } |
| 162 |  | |

|  |  |
| --- | --- |
| 163 |  |
| 164 | /\* The field labels \*/ | |

|  |  |
| --- | --- |
| 165 |  |
| 166 | label { | |

|  |  |  |
| --- | --- | --- |
| 167 | display: block; | |
| 168 | float: left; |

|  |  |
| --- | --- |
| 169 | clear: left; |
| 170 | text-align: right; | |

|  |  |
| --- | --- |
| 171 | width: 15%; |
| 172 | padding: .4em 0 0 0; | |

|  |  |  |
| --- | --- | --- |
| 173 | margin: .15em .5em 0 0; | |
| 174 | } |

|  |  |
| --- | --- |
| 175 |  |
| 176 |  |

|  |  |  |
| --- | --- | --- |
| 177 | /\* The fields \*/ | |
| 178 |  |

|  |  |  |
| --- | --- | --- |
| 179 | input, select, textarea { | |
| 180 | display: block; |

|  |  |
| --- | --- |
| 181 | margin: 0; |
| 182 | padding: .4em; | |

|  |  |  |
| --- | --- | --- |
| 183 | width: 80%; | |
| 184 | } |

|  |  |
| --- | --- |
| 185 |  |
| 186 | input, textarea, .date { | |

|  |  |
| --- | --- |
| 187 | border: 2px solid #666; |
| 188 | -moz-border-radius: 5px; | |

|  |  |  |
| --- | --- | --- |
| 189 | -webkit-border-radius: 5px; | |
| 190 | border-radius: 5px; |

|  |  |  |
| --- | --- | --- |
| 191 | background: #fff; | |
| 192 | } |

|  |  |
| --- | --- |
| 193 |  |
| 194 | input { | |

|  |  |  |
| --- | --- | --- |
| 195 | font-size: .9em; | |
| 196 | } |

|  |  |
| --- | --- |
| 197 |  |
| 198 | select { | |

|  |  |
| --- | --- |
| 199 | padding: 0; |
| 200 | margin-bottom: 2.5em; | |

|  |  |  |
| --- | --- | --- |
| 201 | position: relative; | |
| 202 | top: .7em; |

|  |  |
| --- | --- |
| 203 | } |
| 204 |  | |

|  |  |
| --- | --- |
| 205 | textarea { |
| 206 | font-family: "Trebuchet MS", Arial, Helvetica, sans-serif; | |

|  |  |  |
| --- | --- | --- |
| 207 | font-size: .9em; | |
| 208 | height: 5em; |

|  |  |  |
| --- | --- | --- |
| 209 | line-height: 1.5em; | |
| 210 | } |

|  |  |
| --- | --- |
| 211 |  |
| 212 | textarea#content { | |

|  |  |  |
| --- | --- | --- |
| 213 | font-family: "Courier New", courier, fixed; | |
| 214 | } |

|  |  |  |
| --- | --- | --- |
| 215 |  | |
| 216 |  |

|  |  |  |
| --- | --- | --- |
| 217 | /\* Place a border around focused fields \*/ | |
| 218 |  |

|  |  |
| --- | --- |
| 219 | form \*:focus { |
| 220 | border: 2px solid #7c412b; | |

|  |  |  |
| --- | --- | --- |
| 221 | outline: none; | |
| 222 | } |

|  |  |
| --- | --- |
| 223 |  |
| 224 |  |

|  |  |  |
| --- | --- | --- |
| 225 | /\* Display correctly filled-in fields with a green background \*/ | |
| 226 |  |

|  |  |  |
| --- | --- | --- |
| 227 | input:valid, textarea:valid { | |
| 228 | background: #efe; |

|  |  |
| --- | --- |
| 229 | } |
| 230 |  | |

|  |  |
| --- | --- |
| 231 |  |
| 232 | /\* Submit buttons \*/ | |

|  |  |
| --- | --- |
| 233 |  |
| 234 | .buttons { | |

|  |  |  |
| --- | --- | --- |
| 235 | text-align: center; | |
| 236 | margin: 40px 0 0 0; |

|  |  |
| --- | --- |
| 237 | } |
| 238 |  | |

|  |  |  |
| --- | --- | --- |
| 239 | input[type="submit"] { | |
| 240 | display: inline; |

|  |  |  |
| --- | --- | --- |
| 241 | margin: 0 20px; | |
| 242 | width: 12em; |

|  |  |
| --- | --- |
| 243 | padding: 10px; |
| 244 | border: 2px solid #7c412b; | |

|  |  |
| --- | --- |
| 245 | -moz-border-radius: 5px; |
| 246 | -webkit-border-radius: 5px; | |

|  |  |
| --- | --- |
| 247 | border-radius: 5px; |
| 248 | -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |

|  |  |  |
| --- | --- | --- |
| 249 | -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |
| 250 | box-shadow: 0 0 .5em rgba(0, 0, 0, .8); |

|  |  |
| --- | --- |
| 251 | color: #fff; |
| 252 | background: #ef7d50; | |

|  |  |
| --- | --- |
| 253 | font-weight: bold; |
| 254 | -webkit-appearance: none; | |

|  |  |
| --- | --- |
| 255 | } |
| 256 |  | |

|  |  |  |
| --- | --- | --- |
| 257 | input[type="submit"]:hover, input[type="submit"]:active { | |
| 258 | cursor: pointer; |

|  |  |  |
| --- | --- | --- |
| 259 | background: #fff; | |
| 260 | color: #ef7d50; |

|  |  |
| --- | --- |
| 261 | } |
| 262 |  | |

|  |  |  |
| --- | --- | --- |
| 263 | input[type="submit"]:active { | |
| 264 | background: #eee; |

|  |  |
| --- | --- |
| 265 | -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8) inset; |
| 266 | -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8) inset; | |

|  |  |  |
| --- | --- | --- |
| 267 | box-shadow: 0 0 .5em rgba(0, 0, 0, .8) inset; | |
| 268 | } |

|  |  |
| --- | --- |
| 269 |  |
| 270 |  |

|  |  |  |
| --- | --- | --- |
| 271 | /\* Tables \*/ | |
| 272 |  |

|  |  |
| --- | --- |
| 273 | table { |
| 274 | width: 100%; | |

|  |  |  |
| --- | --- | --- |
| 275 | border-collapse: collapse; | |
| 276 | } |

|  |  |
| --- | --- |
| 277 |  |
| 278 | tr, th, td { | |

|  |  |  |
| --- | --- | --- |
| 279 | padding: 10px; | |
| 280 | margin: 0; |

|  |  |  |
| --- | --- | --- |
| 281 | text-align: left; | |
| 282 | } |

|  |  |
| --- | --- |
| 283 |  |
| 284 | table, th { | |

|  |  |  |
| --- | --- | --- |
| 285 | border: 1px solid #00a0b0; | |
| 286 | } |

|  |  |
| --- | --- |
| 287 |  |
| 288 | th { | |

|  |  |
| --- | --- |
| 289 | border-left: none; |
| 290 | border-right: none; | |

|  |  |  |
| --- | --- | --- |
| 291 | background: #ef7d50; | |
| 292 | color: #fff; |

|  |  |  |
| --- | --- | --- |
| 293 | cursor: default; | |
| 294 | } |

|  |  |
| --- | --- |
| 295 |  |
| 296 | tr:nth-child(odd) { | |

|  |  |  |
| --- | --- | --- |
| 297 | background: #fff4cf; | |
| 298 | } |

|  |  |
| --- | --- |
| 299 |  |
| 300 | tr:nth-child(even) { | |

|  |  |  |
| --- | --- | --- |
| 301 | background: #fff; | |
| 302 | } |

|  |  |
| --- | --- |
| 303 |  |
| 304 | tr:hover { | |

|  |  |  |
| --- | --- | --- |
| 305 | background: #ddd; | |
| 306 | cursor: pointer; |

|  |  |
| --- | --- |
| 307 | } |
| 308 |  | |

|  |  |
| --- | --- |
| 309 |  |
| 310 | /\* Status and error boxes \*/ | |

|  |  |
| --- | --- |
| 311 |  |
| 312 | .statusMessage, .errorMessage { | |

|  |  |  |
| --- | --- | --- |
| 313 | font-size: .8em; | |
| 314 | padding: .5em; |

|  |  |
| --- | --- |
| 315 | margin: 2em 0; |
| 316 | -moz-border-radius: 5px; | |

|  |  |  |
| --- | --- | --- |
| 317 | -webkit-border-radius: 5px; | |
| 318 | border-radius: 5px; |

|  |  |
| --- | --- |
| 319 | -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); |
| 320 | -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |

|  |  |  |
| --- | --- | --- |
| 321 | -box-shadow: 0 0 .5em rgba(0, 0, 0, .8); | |
| 322 | } |

|  |  |
| --- | --- |
| 323 |  |
| 324 | .statusMessage { | |

|  |  |  |
| --- | --- | --- |
| 325 | background-color: #2b2; | |
| 326 | border: 1px solid #080; |

|  |  |  |
| --- | --- | --- |
| 327 | color: #fff; | |
| 328 | } |

|  |  |
| --- | --- |
| 329 |  |
| 330 | .errorMessage { | |

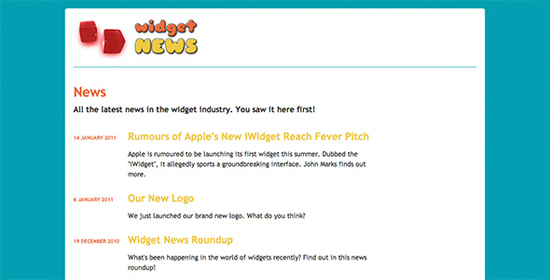
|  |  |  |
| --- | --- | --- |
| 331 | background-color: #f22; | |
| 332 | border: 1px solid #800; |

|  |  |  |
| --- | --- | --- |
| 333 | color: #fff; | |
| 334 | } |

Добавляем селектор h3.categoryDescription для описания категории, идентификатор #headlines для имени категории и класс .category для вывода названия категории на главной странице и в архиве.

## Серверная часть

Осталось изменить шаблоны серверной части. Нужно добавить пару новых шаблонов для вывода списка, добавления, редактирования и удаления категорий. Также нужно модифицировать страницу списка статей для вывода категорий, форму редактирования статьи, чтобы администратор мог добавить категорию для статьи, и заголовок страницы администратора для включения пункта меню, включающего редактор категории.



### Добавляем шаблон listCategories.php

Создаем шаблон listCategories.php, который выводит список категорий в базе данных. Он очень похож на listArticles.php (шаблон списка статей).

Код файла listCategories.php в папке cms/templates/admin:

|  |  |
| --- | --- |
| 01 | <?php include "templates/include/header.php" ?> |
| 02 | <?php include "templates/admin/include/header.php" ?> | |

|  |  |
| --- | --- |
| 03 |  |
| 04 | <h1>Article Categories</h1> | |

|  |  |
| --- | --- |
| 05 |  |
| 06 | <?php if ( isset( $results['errorMessage'] ) ) { ?> | |

|  |  |  |
| --- | --- | --- |
| 07 | <div class="errorMessage"><?php echo $results['errorMessage'] ?></div> | |
| 08 | <?php } ?> |

|  |  |
| --- | --- |
| 09 |  |
| 10 |  |

|  |  |
| --- | --- |
| 11 | <?php if ( isset( $results['statusMessage'] ) ) { ?> |
| 12 | <div class="statusMessage"><?php echo $results['statusMessage'] ?></div> | |

|  |  |  |
| --- | --- | --- |
| 13 | <?php } ?> | |
| 14 |  |

|  |  |  |
| --- | --- | --- |
| 15 | <table> | |
| 16 | <tr> |

|  |  |  |
| --- | --- | --- |
| 17 | <th>Category</th> | |
| 18 | </tr> |

|  |  |
| --- | --- |
| 19 |  |
| 20 | <?php foreach ( $results['categories'] as $category ) { ?> | |

|  |  |
| --- | --- |
| 21 |  |
| 22 | <tr onclick="location='admin.php?action=editCategory&amp;categoryId=<?php echo $category->id?>'"> | |

|  |  |
| --- | --- |
| 23 | <td> |
| 24 | <?php echo $category->name?> | |

|  |  |  |
| --- | --- | --- |
| 25 | </td> | |
| 26 | </tr> |

|  |  |
| --- | --- |
| 27 |  |
| 28 | <?php } ?> | |

|  |  |
| --- | --- |
| 29 |  |
| 30 | </table> | |

|  |  |
| --- | --- |
| 31 |  |
| 32 | <p><?php echo $results['totalRows']?> categor<?php echo ( $results['totalRows'] != 1 ) ? 'ies' : 'y' ?> in total.</p> | |

|  |  |
| --- | --- |
| 33 |  |
| 34 | <p><a href="admin.php?action=newCategory">Add a New Category</a></p> | |

|  |  |
| --- | --- |
| 35 |  |
| 36 | <?php include "templates/include/footer.php" ?> | |

Данный шаблон достаточно простой. Он включает стандартный заголовок страницы и файл заголовка администратора, где выводится также сообщения о статусе и ошибках. Затем выводится таблица категорий в базе данных. Цикл проходит по всем категориям в массиве $results['categories'] и формирует строку таблицы с именем категории. Строка таблицы ссылается на admin.php?action=editCategory, передавая ID категории, так что администратор может редактировать категорию, нажав на строку таблицы.

Шаблон также выводит общее количество категорий в базе данных, включая ссылку функцию добавления новой категории.

### Добавляем editCategory.php

Шаблон editCategory.php выводит форму редактирования, позволяя администратору добавлять категорию или редактировать существующую. Он очень похож на editArticle.php, но немного проще.

Код файла editCategory.php в папке cms/templates/admin:

|  |  |
| --- | --- |
| 01 | <?php include "templates/include/header.php" ?> |
| 02 | <?php include "templates/admin/include/header.php" ?> | |

|  |  |
| --- | --- |
| 03 |  |
| 04 | <h1><?php echo $results['pageTitle']?></h1> | |

|  |  |
| --- | --- |
| 05 |  |
| 06 | <form action="admin.php?action=<?php echo $results['formAction']?>" method="post"> | |

|  |  |  |
| --- | --- | --- |
| 07 | <input type="hidden" name="categoryId" value="<?php echo $results['category']->id ?>"/> | |
| 08 |  |

|  |  |
| --- | --- |
| 09 | <?php if ( isset( $results['errorMessage'] ) ) { ?> |
| 10 | <div class="errorMessage"><?php echo $results['errorMessage'] ?></div> | |

|  |  |  |
| --- | --- | --- |
| 11 | <?php } ?> | |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | <ul> | |
| 14 |  |

|  |  |
| --- | --- |
| 15 | <li> |
| 16 | <label for="name">Category Name</label> | |

|  |  |  |
| --- | --- | --- |
| 17 | <input type="text" name="name" id="name" placeholder="Name of the category" required autofocus maxlength="255" value="<?php echo htmlspecialchars( $results['category']->name )?>" /> | |
| 18 | </li> |

|  |  |
| --- | --- |
| 19 |  |
| 20 | <li> | |

|  |  |
| --- | --- |
| 21 | <label for="description">Description</label> |
| 22 | <textarea name="description" id="description" placeholder="Brief description of the category" required maxlength="1000" style="height: 5em;"><?php echo htmlspecialchars( $results['category']->description )?></textarea> | |

|  |  |  |
| --- | --- | --- |
| 23 | </li> | |
| 24 |  |

|  |  |  |
| --- | --- | --- |
| 25 | </ul> | |
| 26 |  |

|  |  |
| --- | --- |
| 27 | <div class="buttons"> |
| 28 | <input type="submit" name="saveChanges" value="Save Changes" /> | |

|  |  |  |
| --- | --- | --- |
| 29 | <input type="submit" formnovalidate name="cancel" value="Cancel" /> | |
| 30 | </div> |

|  |  |
| --- | --- |
| 31 |  |
| 32 | </form> | |

|  |  |
| --- | --- |
| 33 |  |
| 34 | <?php if ( $results['category']->id ) { ?> | |

|  |  |  |
| --- | --- | --- |
| 35 | <p><a href="admin.php?action=deleteCategory&amp;categoryId=<?php echo $results['category']->id ?>" onclick="return confirm('Delete This Category?')">Delete This Category</a></p> | |
| 36 | <?php } ?> |

|  |  |
| --- | --- |
| 37 |  |
| 38 | <?php include "templates/include/footer.php" ?> | |

В шаблон включаются файлы заголовков, затем выводится содержание  $results['pageTitle'], которое имеет значения либо "New Article Category"(новая категория статьи), либо "Edit Article Category"(редактирование категории статьи). И создается форма, которая отправляется в admin.php со значением переменной $results['formAction'] ("newCategory" или "editCategory") в параметре action.

Сама форма включает скрытое поле categoryId для отслеживания ID текущей редактируемой статьи (если она есть). Выводятся сообщения об ошибках, поля для названия категории и описания, а также кнопки для сохранения и отмены изменений.

Шаблон также включает ссылку на удаление текущей редактируемой категории и нижний колонтитул страницы.

### Изменяем listArticles.php

Нужно изменить шаблон для списка статей, чтобы реализовать новый функционал. Сохраняем новый код файла cms/templates/admin/listArticles.php:

|  |  |
| --- | --- |
| 01 | <?php include "templates/include/header.php" ?> |
| 02 | <?php include "templates/admin/include/header.php" ?> | |

|  |  |
| --- | --- |
| 03 |  |
| 04 | <h1>All Articles</h1> | |

|  |  |
| --- | --- |
| 05 |  |
| 06 | <?php if ( isset( $results['errorMessage'] ) ) { ?> | |

|  |  |  |
| --- | --- | --- |
| 07 | <div class="errorMessage"><?php echo $results['errorMessage'] ?></div> | |
| 08 | <?php } ?> |

|  |  |
| --- | --- |
| 09 |  |
| 10 |  |

|  |  |
| --- | --- |
| 11 | <?php if ( isset( $results['statusMessage'] ) ) { ?> |
| 12 | <div class="statusMessage"><?php echo $results['statusMessage'] ?></div> | |

|  |  |  |
| --- | --- | --- |
| 13 | <?php } ?> | |
| 14 |  |

|  |  |  |
| --- | --- | --- |
| 15 | <table> | |
| 16 | <tr> |

|  |  |  |
| --- | --- | --- |
| 17 | <th>Publication Date</th> | |
| 18 | <th>Article</th> |

|  |  |  |
| --- | --- | --- |
| 19 | <th>Category</th> | |
| 20 | </tr> |

|  |  |
| --- | --- |
| 21 |  |
| 22 | <?php foreach ( $results['articles'] as $article ) { ?> | |

|  |  |
| --- | --- |
| 23 |  |
| 24 | <tr onclick="location='admin.php?action=editArticle&amp;articleId=<?php echo $article->id?>'"> | |

|  |  |  |
| --- | --- | --- |
| 25 | <td><?php echo date('j M Y', $article->publicationDate)?></td> | |
| 26 | <td> |

|  |  |  |
| --- | --- | --- |
| 27 | <?php echo $article->title?> | |
| 28 | </td> |

|  |  |
| --- | --- |
| 29 | <td> |
| 30 | <?php echo $results['categories'][$article->categoryId]->name?> | |

|  |  |  |
| --- | --- | --- |
| 31 | </td> | |
| 32 | </tr> |

|  |  |
| --- | --- |
| 33 |  |
| 34 | <?php } ?> | |

|  |  |
| --- | --- |
| 35 |  |
| 36 | </table> | |

|  |  |
| --- | --- |
| 37 |  |
| 38 | <p><?php echo $results['totalRows']?> article<?php echo ( $results['totalRows'] != 1 ) ? 's' : '' ?> in total.</p> | |

|  |  |
| --- | --- |
| 39 |  |
| 40 | <p><a href="admin.php?action=newArticle">Add a New Article</a></p> | |

|  |  |
| --- | --- |
| 41 |  |
| 42 | <?php include "templates/include/footer.php" ?> | |

Мы добавили новый столбец для категорий в списке статей. В цикле формирования строк происходит поиск соответствующего объекта Category по ID в массиве $results['categories'] и выводится название категории.

### Редактируем editArticle.php

Также нужно модифицировать форму редактирования статьи editArticle.php для добавления возможности назначать категорию статье.

Новый код файла cms/templates/admin/editArticle.php:

|  |  |
| --- | --- |
| 01 | <?php include "templates/include/header.php" ?> |
| 02 | <?php include "templates/admin/include/header.php" ?> | |

|  |  |
| --- | --- |
| 03 |  |
| 04 | <h1><?php echo $results['pageTitle']?></h1> | |

|  |  |
| --- | --- |
| 05 |  |
| 06 | <form action="admin.php?action=<?php echo $results['formAction']?>" method="post"> | |

|  |  |  |
| --- | --- | --- |
| 07 | <input type="hidden" name="articleId" value="<?php echo $results['article']->id ?>"/> | |
| 08 |  |

|  |  |
| --- | --- |
| 09 | <?php if ( isset( $results['errorMessage'] ) ) { ?> |
| 10 | <div class="errorMessage"><?php echo $results['errorMessage'] ?></div> | |

|  |  |  |
| --- | --- | --- |
| 11 | <?php } ?> | |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | <ul> | |
| 14 |  |

|  |  |
| --- | --- |
| 15 | <li> |
| 16 | <label for="title">Article Title</label> | |

|  |  |  |
| --- | --- | --- |
| 17 | <input type="text" name="title" id="title" placeholder="Name of the article" required autofocus maxlength="255" value="<?php echo htmlspecialchars( $results['article']->title )?>" /> | |
| 18 | </li> |

|  |  |
| --- | --- |
| 19 |  |
| 20 | <li> | |

|  |  |
| --- | --- |
| 21 | <label for="summary">Article Summary</label> |
| 22 | <textarea name="summary" id="summary" placeholder="Brief description of the article" required maxlength="1000" style="height: 5em;"><?php echo htmlspecialchars( $results['article']->summary )?></textarea> | |

|  |  |  |
| --- | --- | --- |
| 23 | </li> | |
| 24 |  |

|  |  |
| --- | --- |
| 25 | <li> |
| 26 | <label for="content">Article Content</label> | |

|  |  |  |
| --- | --- | --- |
| 27 | <textarea name="content" id="content" placeholder="The HTML content of the article" required maxlength="100000" style="height: 30em;"><?php echo htmlspecialchars( $results['article']->content )?></textarea> | |
| 28 | </li> |

|  |  |
| --- | --- |
| 29 |  |
| 30 | <li> | |

|  |  |  |
| --- | --- | --- |
| 31 | <label for="categoryId">Article Category</label> | |
| 32 | <select name="categoryId"> |

|  |  |  |
| --- | --- | --- |
| 33 | <option value="0"<?php echo !$results['article']->categoryId ? " selected" : ""?>>(none)</option> | |
| 34 | <?php foreach ( $results['categories'] as $category ) { ?> |

|  |  |  |
| --- | --- | --- |
| 35 | <option value="<?php echo $category->id?>"<?php echo ( $category->id == $results['article']->categoryId ) ? " selected" : ""?>><?php echo htmlspecialchars( $category->name )?></option> | |
| 36 | <?php } ?> |

|  |  |  |
| --- | --- | --- |
| 37 | </select> | |
| 38 | </li> |

|  |  |
| --- | --- |
| 39 |  |
| 40 | <li> | |

|  |  |
| --- | --- |
| 41 | <label for="publicationDate">Publication Date</label> |
| 42 | <input type="date" name="publicationDate" id="publicationDate" placeholder="YYYY-MM-DD" required maxlength="10" value="<?php echo $results['article']->publicationDate ? date( "Y-m-d", $results['article']->publicationDate ) : "" ?>" /> | |

|  |  |  |
| --- | --- | --- |
| 43 | </li> | |
| 44 |  |

|  |  |
| --- | --- |
| 45 |  |
| 46 | </ul> | |

|  |  |
| --- | --- |
| 47 |  |
| 48 | <div class="buttons"> | |

|  |  |
| --- | --- |
| 49 | <input type="submit" name="saveChanges" value="Save Changes" /> |
| 50 | <input type="submit" formnovalidate name="cancel" value="Cancel" /> | |

|  |  |  |
| --- | --- | --- |
| 51 | </div> | |
| 52 |  |

|  |  |  |
| --- | --- | --- |
| 53 | </form> | |
| 54 |  |

|  |  |
| --- | --- |
| 55 | <?php if ( $results['article']->id ) { ?> |
| 56 | <p><a href="admin.php?action=deleteArticle&amp;articleId=<?php echo $results['article']->id ?>" onclick="return confirm('Delete This Article?')">Delete This Article</a></p> | |

|  |  |  |
| --- | --- | --- |
| 57 | <?php } ?> | |
| 58 |  |

|  |  |
| --- | --- |
| 59 | <?php include "templates/include/footer.php" ?> |

Добавляем поле Article Category (категория статьи) к форме. Оно представляет меню select, которое содержит список всех категорий в базе данных, получаемый из массива $results['categories']. Каждый элемент option хранит ID категории в атрибуте value и выводит название категории. Если происходит редактирование существующей статьи, то код выполняет проверку соответствия текущего ID категории статьи и ID выбранной категории. Если они совпадают, то добавляется атрибут selected для предварительного выбора опции в списке.

Элемент select также включает опцию "(none)"(нет категории) вверху списка с нулевым значением. Таким образом, администратор может создавать статьи, которые не включены ни в какую категорию.

### Редактируем заголовок страниц серверной части

Также нам нужно изменить заголовок, который выводится на страницах серверной части. Пока он включает небольшое меню ( Edit Articles (редактировать статьи) и Log Out(выход)). Нам нужно добавить пункт Edit Categories (редактировать категории), чтобы дать возможность администратору просматривать, редактировать, добавлять и удалять категории.

В базовом варианте заголовок был встроен в каждый шаблон серверной части (listArticles.php и editArticle.php). Вынесем код заголовка в отдельный файл header.php. Теперь изменения нужно будет вносить в одно место.

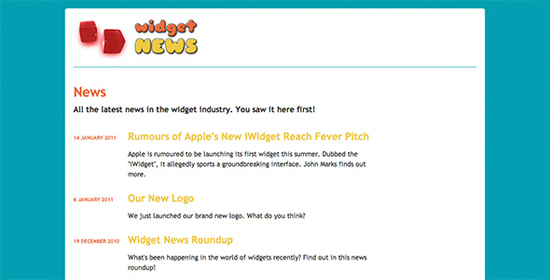
Шаблоны listArticles.php и editArticle.php уже изменены для включения файла заголовка, а новые файлы  listCategories.php и editCategories.php сразу содержат правильный код.

Итак, создаем папку include в каталоге cms/templates/admin и сохраняем следующий код в файле header.php, который размещается в новой папке:

|  |  |
| --- | --- |
| 1 | <div id="adminHeader"> |
| 2 | <h2>Widget News Admin</h2> | |

|  |  |  |
| --- | --- | --- |
| 3 | <p>You are logged in as <b><?php echo htmlspecialchars( $\_SESSION['username']) ?></b>. <a href="admin.php?action=listArticles">Edit Articles</a> <a href="admin.php?action=listCategories">Edit Categories</a> <a href="admin.php?action=logout"?>Log Out</a></p> | |
| 4 | </div> |

Код представляет разметку, которая ранее была встроена в шаблоны listArticles.php и editArticle.php с добавлением новой опции "Edit Categories", которая ссылается на admin.php?action=listCategories.



Теперь наша CMS позволяет работать с категориями статьей. В следующих уроках мы разберемся, как добавить возможность вставки изображений.