Costruiamo un Mini-CMS PHP/MySQL

L’obiettivo di questo articolo non è quello di creare un CMS come WordPress, Drupal, Joomla o altri, ma è quello di capire come funziona un CMS e quali sono le logiche di programmazione che ci sono dietro. Quando abbiamo chiaro il funzionamento possiamo effettuare considerazioni molto più avanzate anche in ottica SEO e, in generare, per ottimizzare al meglio un sito web. Iniziamo!

Le caratteristiche che avrà il nostro Content Management System sono:

Front end:

- home page con la lista degli ultimi articoli
- la pagina di listing con gli articoli
- la pagina che visualizza il singolo articolo

Back end:

- login/logout come Amministratore
- lista di tutti gli articoli
- aggiunta di un nuovo articolo
- modifica di un articolo già esistente
- eliminazione di un articolo

Ad ogni articolo daremo un titolo (intestazione), una descrizione breve e una data di pubblicazione.

Creiamo il database

Il database può essere creato, in modo rapido, tramite PHP MySQL oppure da terminale utilizzando questo comando:

mysql -u username -p

successivamente aggiungiamo la password:

mysql> prompt

digitiamo: create database cms e premiamo “enter” e poi “exit”.

Ora abbiamo creato il nostro database di nome “cms”, ora dobbiamo creare una tabella per archiviare i contenuti.

Creiamo la tabella “articles”

Il nostro mini-cms avrà una sola tabella, articles, che conterrà gli articoli. Creiamo un file di testo chiamato tables.sql e aggiungiamo queste righe di codice:

```
DROP TABLE IF EXISTS articles;
CREATE TABLE articles
(
  id             smallint unsigned NOT NULL auto_increment,
          publicationDate     date     NOT     NULL,
# Data di pubblicazione di un articolo
    title                      varchar(255)  NOT  NULL,
# titolo di un articolo
     summary                           text   NOT   NULL,
# breve descrizione di un articolo
    content                     mediumtext  NOT  NULL,
# contenuto principale di un articolo

  PRIMARY KEY     (id)
);
```

Il codice sopra è lo schema che definisce la tabella “tables”, scritto in SQL che è il linguaggio di manipolazione e creazione di database in MySQL (e in molti altri sistemi di database). Vediamolo nel dettaglio:

ID: è un numero univoco che identifica uno ed un solo articolo.

publicationDate: è un campo data che archivia la data di

pubblicazione di ogni articolo

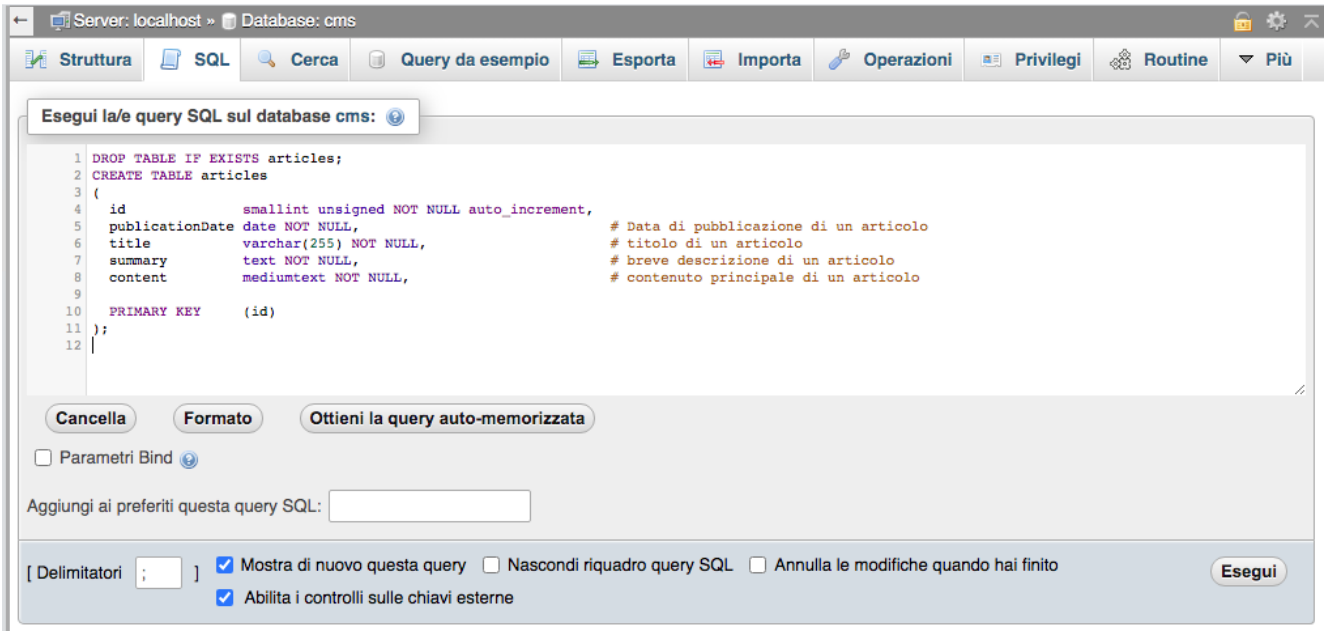
title: è un campo di testo che archivia il titolo degli articoli

summary e **content** sono due campi che memorizzano all'interno del database, per ogni articolo, una breve descrizione e il contenuto vero e proprio.

L'ultima riga all'interno dell'istruzione CREATE TABLE definisce una chiave per la tabella. Una chiave è anche chiamata indice e, semplificando, rende più rapida la ricerca dei dati nella tabella, a scapito di un pò di spazio di archiviazione aggiuntivo.

Rendiamo il campo **id** una CHIAVE PRIMARIA. Ogni tabella può avere una sola CHIAVE PRIMARIA; questa è la chiave che identifica in modo univoco ogni record nella tabella. Inoltre, aggiungendo questa chiave, MySQL può recuperare un articolo basato sul suo ID molto rapidamente.

Possiamo creare questo schema direttamente all'interno del nostro database inserendo le regole SQL, vedere immagine di seguito.



Creiamo un file di configurazione

Per il corretto funzionamento del nostro mini-CMS è necessario creare un file di configurazione dove andremo ad aggiungere le credenziali del database e di accesso al back-end. Vediamo nel dettaglio.

```
<?php
ini_set( "display_errors", true );
date_default_timezone_set('Europe/Rome');         //
http://www.php.net/manual/en/timezones.php
define( "DB_DSN", "mysql:host=localhost;dbname=cms" );
define( "DB_USERNAME", "root" );
define( "DB_PASSWORD", "" );
define( "CLASS_PATH", "classes" );
define( "TEMPLATE_PATH", "templates" );
define( "HOMEPAGE_NUM_ARTICLES", 5 );
define( "ADMIN_USERNAME", "admin" );
define( "ADMIN_PASSWORD", "miapass" );
require( CLASS_PATH . "/Article.php" );

function handleException( $exception ) {
  echo "Si e' verificato un errore, riprovare in seguito
oppure contattare il titolare del sito web";
  error_log( $exception->getMessage() );
}

set_exception_handler( 'handleException' );
?>
```

ini_set()

ini_set() attiva la visualizzazione degli errori nel browser, è molto importante in fase di debugging. Per motivi di sicurezza è bene attivare questa visualizzazione soltanto agli amministratori loggati nel sito web (si può fare con un’istruzione IF e intercettando la sessione dell’utente).

Impostiamo timezone

Il nostro CMS utilizza la data per aggiungere il giorno di pubblicazione degli articoli, quindi è opportuno definire il fuso orario correttamente su Europa/Rome.

Successivamente definiamo i **parametri di accesso al database.**

Impostiamo 2 path

Abbiamo impostato 2 path nel nostro file di configurazione: CLASS_PATH, che è il percorso dei file di classe, e TEMPLATE_PATH, che è dove il nostro script dovrebbe cercare i file dei template HTML. Entrambi questi percorsi sono relativi alla nostra cartella cms di primo livello.

Impostiamo il N° di articoli da mostrare in home page

HOMEPAGE_NUM_ARTICLES definisce il numero massimo di articoli da mostrare in home page, attualmente impostato a “5” ma può essere cambiato in ogni momento.

Credenziali per il back end

Con le 2 constanti ADMIN_USERNAME e ADMIN_PASSWORD impostiamo username/password per accedere al pannello di amministrazione.

Aggiungiamo la classe Articles

Poiché il file di classe Article, che creeremo in seguito, **è necessario per tutti gli script** nella nostra applicazione, lo includiamo qui, così da poter essere riutilizzato in ogni parte del CMS.

Creiamo exception handler

Infine, definiamo handleException (), una funzione per gestire eventuali eccezioni PHP che potrebbero essere sollevate

durante l’esecuzione del nostro codice. La funzione visualizza un messaggio di errore generico e registra il messaggio di eccezione effettivo nel registro degli errori del server web (server log). In particolare, questa funzione migliora la sicurezza gestendo eventuali eccezioni PDO che potrebbero altrimenti visualizzare il nome utente e la password del database nella pagina. Dopo aver definito handleException(), lo impostiamo come gestore di eccezioni chiamando la funzione **set_exception_handler ()** di PHP.

Costruiamo la casse Article

A questo punto siamo pronti per creare la nostra classe PHP che è fondamentale per compiere le principali operazioni sugli articoli: creare, leggere, aggiornare ed eliminare, ovvero l così detto CRUD (create, read, update and delete). Creiamo una nuova cartella chiamata “classes” e al suo interno aggiungiamo un file chiamato Articles.php con il seguente codice:

```
<?php

/**
 * Classe per gestire gli articoli
 */

class Article
{
  // Proprietà

  /**
  * @var int ID dell'articolo dal database
  */
  public $id = null;

  /**
  * @var int  Quando un articolo deve essere pubblicato oppure è stato pubblicato per la prima volta
  */
  public $publicationDate = null;
```

```
  /**
  * @var string Titolo dell'articolo
  */
  public $title = null;

  /**
  * @var string Un breve riassunto dell'articolo
  */
  public $summary = null;

  /**
   * @var string HTML dell'articolo, quindi il contenuto
principale
  */
  public $content = null;



  /**
  * Impostiamo le proprietà dell'oggetto utilizzando i valori
nell'Array
  *
  * @param assoc Valori
  */

  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'];
  }


  /**
  * Impostiamo le proprietà dell'oggetto utilizzando i valori
```

```
di inserimento del modulo di modifica nell'array fornita
  *
  * @param assoc
  */

  public function storeFormValues ( $params ) {

    // Memorizziamo tutti i parametri
    $this->__construct( $params );

    // Processiamo e archiviamo la data di pubblicazione
    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
);
      }
    }
  }


  /**
   * Restituisce un oggetto-Articolo che corrisponde all'ID
articolo specificato
  *
  * @param int ID dell'articolo
  * @return Article|false Ogetto articolo oppure "falso" se il
record non è stato trovato o si è verificato un problema
  */

  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 );
  }


  /**
  * Restituisce tutti gli Article objects nel DB
  *
   * @param int Optional The number of rows to return (default=all)
   * @return Array|false A two-element array : results => array, a list of Article objects; totalRows => Total number of articles
  */

  public static function getList( $numRows=1000000 ) {
    $conn = new PDO( DB_DSN, DB_USERNAME, DB_PASSWORD );
          $sql = "SELECT SQL_CALC_FOUND_ROWS *, UNIX_TIMESTAMP(publicationDate) AS publicationDate FROM articles
           ORDER BY publicationDate DESC 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;
    }

    // Now get the total number of articles that matched the criteria
    $sql = "SELECT FOUND_ROWS() AS totalRows";
    $totalRows = $conn->query( $sql )->fetch();
    $conn = null;
     return ( array ( "results" => $list, "totalRows" => $totalRows[0] ) );
  }
```

```
  /**
  * Inserts the current Article object into the database, and
sets its ID property.
  */

  public function insert() {

    // Does the Article object already have an 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
);

    // Insert the Article
    $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;
  }



  /**
  * Updates the current Article object in the database.
  */

  public function update() {

    // Does the Article object have an 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 );

    // Update the Article
    $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;
  }


  /**
  * Deletes the current Article object from the database.
  */

  public function delete() {

    // Does the Article object have an 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 );

    // Delete the Article
    $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;
  }
```

```
}

?>
```

Il codice del file è un pò lungo ma analizzando le singole sezione, leggendo i commenti, le impostazioni sono chiare.

Creiamo lo script per il front-end

Dopo avere programmato il file article.php la parte più grande e complessa è terminata! procediamo con la creazione del file index.php che si occuperà della visualizzazione delle pagine nel front end del nostro CMS. Creiamo un file index.php con il seguente codice e lo aggiungiamo nella root principale:

```
<?php

require( "config.php" );
$action = isset( $_GET['action'] ) ? $_GET['action'] : "";

switch ( $action ) {
  case 'archive':
    archive();
    break;
  case 'viewArticle':
    viewArticle();
    break;
  default:
    homepage();
}

function archive() {
  $results = array();
  $data = Article::getList();
  $results['articles'] = $data['results'];
  $results['totalRows'] = $data['totalRows'];
  $results['pageTitle'] = "Article Archive | Widget News";
  require( TEMPLATE_PATH . "/archive.php" );
}

function viewArticle() {
```

```
  if ( !isset($_GET["articleId"]) || !$_GET["articleId"] ) {
    homepage();
    return;
  }

  $results = array();
        $results['article']    =    Article::getById(
(int)$_GET["articleId"] );
   $results['pageTitle'] = $results['article']->title . " |
Widget News";
  require( TEMPLATE_PATH . "/viewArticle.php" );
}

function homepage() {
  $results = array();
  $data = Article::getList( HOMEPAGE_NUM_ARTICLES );
  $results['articles'] = $data['results'];
  $results['totalRows'] = $data['totalRows'];
  $results['pageTitle'] = "Widget News";
  require( TEMPLATE_PATH . "/homepage.php" );
}

?>
```

Analizziamo il codice step by step:

includiamo il file config.php

La prima riga di codice include il file config.php che abbiamo creato in precedenza, in modo che tutte le impostazioni di configurazione siano disponibili per lo script. Usiamo **require()** piuttosto che **include(),** require() genera un errore se il file non può essere trovato.

Utilizziamo il parametro di action

Memorizziamo il parametro $ _GET [‘action’] in una variabile chiamata $action, in modo da poter utilizzare il valore in seguito nello script. Prima di fare ciò, controlliamo che il valore $ _GET [‘action’] esista utilizzando isset(). In caso contrario, impostiamo la corrispondente variabile $action su

una stringa vuota (“”).

Quale action utilizzare?

Il blocco switch esamina il parametro di azione nell’URL per determinare quale azione eseguire (visualizzare l’archivio o visualizzare un articolo). Se nell’URL non è presente alcun parametro di azione, lo script visualizza la home page del sito.

Archive()

Questa funzione visualizza un elenco di tutti gli articoli nel database. Lo fa chiamando il metodo **getList()** della classe **Article** che abbiamo creato in precedenza. La funzione quindi memorizza i risultati, insieme al titolo della pagina, in un array associativo **$results** in modo che il modello possa visualizzarli nella pagina. Infine, include il file modello per visualizzare la pagina. (Creeremo i modelli tra un pochi minuti).

viewArticle()

Questa funzione visualizza una **singola pagina di articolo**. Recupera l’ID dell’articolo da visualizzare dal parametro URL **articleId**, quindi chiama il metodo **getById()** della classe **Article** per recuperare l’oggetto articolo, che memorizza nell’array **$results** affinché il modello possa utilizzare. (Se non è stato fornito alcun articleId o l’articolo non è stato trovato, la funzione visualizza semplicemente la home page.)

homepage()

Ora vediamo l’ultima funzione, **homepage()**, mostra la homepage del sito contenente un elenco di fino a HOMEPAGE_NUM_ARTICLES articoli (5 per impostazione predefinita). È molto simile alla funzione **archive()**, tranne per il fatto che passa HOMEPAGE_NUM_ARTICLES al metodo **getList()** per limitare il numero di articoli restituiti.

Ora ci occupiamo del backend

Vediamo il file PHP che si occupa delle funzionalità del backend. Creiamo un file PHP chiamato admin.php nella root principale del nostro mini-CMS, e aggiungiamo il seguente codice:

```
<?php

require( "config.php" );
session_start();
$action = isset( $_GET['action'] ) ? $_GET['action'] : "";
$username  =  isset(  $_SESSION['username']  )  ?
$_SESSION['username'] : "";

if ( $action != "login" && $action != "logout" && !$username )
{
  login();
  exit;
}

switch ( $action ) {
  case 'login':
    login();
    break;
  case 'logout':
    logout();
    break;
  case 'newArticle':
    newArticle();
    break;
  case 'editArticle':
    editArticle();
    break;
  case 'deleteArticle':
    deleteArticle();
    break;
  default:
    listArticles();
}
```

```
function login() {

  $results = array();
  $results['pageTitle'] = "Admin Login | Widget News";

  if ( isset( $_POST['login'] ) ) {

    // User has posted the login form: attempt to log the user
in

      if ( $_POST['username'] == ADMIN_USERNAME &&
$_POST['password'] == ADMIN_PASSWORD ) {

       // Login successful: Create a session and redirect to
the admin homepage
      $_SESSION['username'] = ADMIN_USERNAME;
      header( "Location: admin.php" );

    } else {

      // Login failed: display an error message to the user
        $results['errorMessage'] = "Incorrect username or
password. Please try again.";
      require( TEMPLATE_PATH . "/admin/loginForm.php" );
    }

  } else {

     // User has not posted the login form yet: display the
form
    require( TEMPLATE_PATH . "/admin/loginForm.php" );
  }

}


function logout() {
  unset( $_SESSION['username'] );
  header( "Location: admin.php" );
}
```

```
function newArticle() {

  $results = array();
  $results['pageTitle'] = "New Article";
  $results['formAction'] = "newArticle";

  if ( isset( $_POST['saveChanges'] ) ) {

    // User has posted the article edit form: save the new
article
    $article = new Article;
    $article->storeFormValues( $_POST );
    $article->insert();
    header( "Location: admin.php?status=changesSaved" );

  } elseif ( isset( $_POST['cancel'] ) ) {

    // User has cancelled their edits: return to the article
list
    header( "Location: admin.php" );
  } else {

    // User has not posted the article edit form yet: display
the form
    $results['article'] = new Article;
    require( TEMPLATE_PATH . "/admin/editArticle.php" );
  }

}


function editArticle() {

  $results = array();
  $results['pageTitle'] = "Edit Article";
  $results['formAction'] = "editArticle";

  if ( isset( $_POST['saveChanges'] ) ) {

    // User has posted the article edit form: save the article
changes
```

```
            if   (   !$article   =   Article::getById(
(int)$_POST['articleId'] ) ) {
      header( "Location: admin.php?error=articleNotFound" );
      return;
    }

    $article->storeFormValues( $_POST );
    $article->update();
    header( "Location: admin.php?status=changesSaved" );

  } elseif ( isset( $_POST['cancel'] ) ) {

    // User has cancelled their edits: return to the article
list
    header( "Location: admin.php" );
  } else {

    // User has not posted the article edit form yet: display
the form
          $results['article']   =   Article::getById(
(int)$_GET['articleId'] );
    require( TEMPLATE_PATH . "/admin/editArticle.php" );
  }

}


function deleteArticle() {

  if ( !$article = Article::getById( (int)$_GET['articleId'] )
) {
    header( "Location: admin.php?error=articleNotFound" );
    return;
  }

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


function listArticles() {
```

```
  $results = array();
  $data = Article::getList();
  $results['articles'] = $data['results'];
  $results['totalRows'] = $data['totalRows'];
  $results['pageTitle'] = "All Articles";

  if ( isset( $_GET['error'] ) ) {
       if  (  $_GET['error']  ==  "articleNotFound"  )
$results['errorMessage'] = "Error: Article not found.";
  }

  if ( isset( $_GET['status'] ) ) {
        if  (  $_GET['status']  ==  "changesSaved"  )
$results['statusMessage'] = "Your changes have been saved.";
       if  (  $_GET['status']  ==  "articleDeleted"  )
$results['statusMessage'] = "Article deleted.";
  }

  require( TEMPLATE_PATH . "/admin/listArticles.php" );
}

?>
```

Il file admin.php ha una struttura simile al file index.php. Vediamo le sezioni principali di admin.php.

Inizia la sessione utente, chiamando **session_start()**. Questa funzione PHP avvia una nuova sessione per l'utente, che possiamo utilizzare per monitorare se l'utente è loggato o meno. (Se esiste già una sessione per questo utente, PHP la raccoglie automaticamente e la utilizza).

Parametro di azione e variabile di sessione del nome utente

Successivamente memorizziamo il parametro $ _GET ['action'] in una variabile chiamata $action e la variabile di sessione $ _SESSION ['username'] in $username, in modo da poter utilizzare questi valori in seguito nello script. Prima di fare ciò, controlliamo che questi valori esistano usando isset (). Se un valore non esiste, impostiamo la variabile corrispondente su una stringa vuota ("").

Controlliamo se l’utente è loggato

All’utente non dovrebbe essere consentito di fare nulla a meno che non abbia effettuato l’accesso come amministratore. Quindi la prossima cosa che facciamo è ispezionare **$username** per vedere se la sessione conteneva un valore per la chiave del nome utente, che usiamo per indicare che l’utente è loggato. Se il valore di $username è vuoto – e l’utente non lo è stiamo già tentando di accedere o disconnettersi, quindi visualizziamo la pagina di accesso e usciamo immediatamente.

Decidiamo quale action impostare

Il blocco switch funziona in modo molto simile a quello in index.php: chiama la funzione appropriata in base al valore del parametro URL dell’azione. L’azione predefinita è visualizzare l’elenco degli articoli nel CSM.

login()

login() viene chiamato quando l’utente deve effettuare il login o è in procinto di effettuare il login.
Se l’utente ha inviato il modulo di accesso, che controlliamo cercando il parametro del modulo di accesso, la funzione verifica il nome utente e la password immessi rispetto ai valori di configurazione ADMIN_USERNAME e ADMIN_PASSWORD. Se corrispondono, la chiave di sessione del nome utente viene impostata sul nome utente dell’amministratore, accedendo effettivamente a loro, quindi reindirizziamo il browser allo script admin.php, che quindi visualizza l’elenco degli articoli. Se il nome utente e la password non corrispondono, il modulo di accesso viene nuovamente visualizzato con un messaggio di errore.

Se l’utente non ha ancora inviato il modulo di accesso, la funzione visualizza semplicemente il modulo.

logout()

Questa funzione viene chiamata quando l'utente decide di disconnettersi. Rimuove semplicemente la chiave di sessione del nome utente e reindirizza nuovamente ad admin.php.

newArticle()

Questa funzione permette all'utente di creare un nuovo articolo. Se l'utente ha appena pubblicato il modulo "nuovo articolo", la funzione crea un nuovo oggetto Articolo, memorizza i dati del modulo nell'oggetto chiamando storeFormValues □□(), inserisce l'articolo nel database chiamando **insert()** e reindirizza a l'elenco degli articoli, che mostra un messaggio di stato "Modifiche salvate".

Se l'utente non ha ancora pubblicato il modulo "nuovo articolo", la funzione crea un nuovo oggetto Articolo vuoto senza valori, quindi utilizza il modello editArticle.php per visualizzare il modulo di modifica dell'articolo utilizzando questo oggetto Articolo vuoto.

editArticle()

Questa funzione è simile a **newArticle()**, tranne per il fatto che consente all'utente di modificare un articolo esistente. Quando l'utente salva le modifiche, la funzione recupera l'articolo esistente utilizzando getById(), memorizza i nuovi valori nell'oggetto Article, quindi salva l'oggetto modificato chiamando update(). (Se l'articolo non viene trovato nel database, la funzione visualizza un errore.)

Quando si visualizza il modulo di modifica dell'articolo, la funzione utilizza nuovamente il metodo getById() per caricare i valori del campo dell'articolo corrente nel modulo per la modifica.

deleteArticle()

Se l'utente ha scelto di eliminare un articolo, questa funzione recupera prima l'articolo da eliminare (visualizzando

un errore se l’articolo non è stato trovato nel database), quindi chiama il metodo delete() dell’articolo per rimuovere l’articolo dal Banca dati. Quindi reindirizza alla pagina dell’elenco degli articoli, visualizzando un messaggio di stato “articolo eliminato”.

listArticles()

L’ultima funzione in admin.php mostra un elenco di tutti gli articoli nel CMS per la modifica. La funzione utilizza il metodo getList() della classe Article per recuperare tutti gli articoli, quindi utilizza il modello listArticles.php per visualizzare l’elenco. Lungo il percorso, controlla anche l’errore e lo stato dei parametri di query dell’URL per vedere se nella pagina deve essere visualizzato un messaggio di errore o di stato. In tal caso, crea il messaggio necessario e lo passa al modello per la visualizzazione.

Occupiamoci dei template per il front end

A questo punto abbiamo creato tutte le funzionalità principali del nostro CMS, e ora possiamo occuparci della parte visuale, quale grafica vogliamo dare al CMS? Iniziamo a creare i nostri template HTML per il front-end.

Creiamo una cartella chiamata “**templates**” all’interno della cartella principale “**cms**“. Successivamente creiamo una cartella chiamata “**include**“, in quest’ultima andremo ad aggiungere i template per l’header e il footer che sono presenti (senza variazioni) in ogni pagina.

A questo punto creiamo un nuovo file chiamato header.php dentro la cartella “include” con il seguente codice:

```
<!DOCTYPE html>
<html lang="it">
  <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="Il nostro Logo" />
</a>
```

Il codice sopra riportato è molto semplice: mostra il codice necessario per l'HTML della pagina, utilizza la variabile $result['pageTitle'] (che gli abbiamo passato nello script principale index.php o admin.php) per impostare il titolo, inoltre è presente il link al foglio di stile **style.css**.

A questo punto possiamo creare il file footer.php con al suo interno il seguente codice:

```
<div id="footer">
          Nostro sito web &copy; 2021. Tutti i diritti riservati. <a href="admin.php">Amministrazione sito web (Backend)</a>
      </div>

    </div>
  </body>
</html>
```

Il codice del footer.php si occupa della chiusura di tutti i tag HTML aperti precedentemente e di aggiungere una porzione di contenuto (copyright).

Template HomePage

Creiamo il file homepage.php e lo aggiungiamo nella cartella "templates" con il seguente codice:

```
<?php include "templates/include/header.php" ?>
```

```
      <ul id="headlines">

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

        <li>
          <h2>
              <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>
          </h2>
            <p class="summary"><?php echo htmlspecialchars( $article->summary )?></p>
        </li>

<?php } ?>

      </ul>

      <p><a href="./?action=archive">Article Archive</a></p>

<?php include "templates/include/footer.php" ?>
```

Il codice sopra riportato si occupa di aggiungere i file header.php e footer.php nella pagina e di mostrare la lista degli articoli all'interno di un elenco puntato. È presente un ciclo PHP di un Array di oggetti presenti in $results['articles'] e per ogni articolo mostra la data di pubblicazione, il titolo e il riassunto (summary).

Il titolo è collegato a "." (index.php), passando l'action = **viewArticle**, così come l'ID dell'articolo, nell'URL. Ciò consente all'utente di leggere un articolo facendo clic sul suo titolo. Il template, inoltre, mostra un link all'archivio degli articoli ("./?action=archive").

Template archive.php

Ora ci occupiamo di creare il template per il listing di

contenuti (archive.php) da aggiungere all'interno della cartella "templates", al suo interno aggiungiamo il seguente codice:

```
<?php include "templates/include/header.php" ?>

      <h1>Archivio degli articoli</h1>

      <ul id="headlines" class="archive">

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

        <li>
          <h2>
             <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>
          </h2>
            <p class="summary"><?php echo htmlspecialchars( $article->summary )?></p>
        </li>

<?php } ?>

      </ul>

      <p><?php echo $results['totalRows']?> articoli<?php echo ( $results['totalRows'] != 1 ) ? 's' : '' ?> in totale.</p>

      <p><a href="./">Torna alla Homepage</a></p>

<?php include "templates/include/footer.php" ?>
```

Questo template è molto simile a quello progettato per la homepage, è presente una classe CSS "archive" all'interno della lista <ul> che ci permette di impostare uno stile differente. Inoltre mostra il numero di articoli presenti tramite $results['totalRows'], effettuando un conteggio di righe nel database.

Template viewarticle.php

Siamo arrivati all'ultimo template per il frontend, creiamo un file chiamato **viewArticle.php** da aggiungere all'interno della cartella "templates" con il seguente codice PHP:

```
<?php include "templates/include/header.php" ?>

       <h1 style="width: 75%;"><?php echo htmlspecialchars( $results['article']->title )?></h1>
      <div style="width: 75%; font-style: italic;"><?php echo htmlspecialchars( $results['article']->summary )?></div>
              <div  style="width:  75%;"><?php  echo $results['article']->content?></div>
       <p class="pubDate">Pubblicato il <?php echo date('j F Y', $results['article']->publicationDate)?></p>

      <p><a href="./">Torna alla Homepage</a></p>

<?php include "templates/include/footer.php" ?>
```

Questo template si occupa di rendere visibili le informazioni del singolo articolo, come il titolo, la descrizione, la data di pubblicazione e il link per tornare alla home page.

Occupiamoci dei template per il back end

Ora che abbiamo finito di occuparci della creazione dei template per il front end è giunto il momento di pensare al back end. Iniziamo!

Template login

Creiamo una nuova cartella chiamata "admin" all'interno della cartella "templates" e, al suo interno, creiamo un file chiamato **loginForm.php** con all'interno il seguente codice:

```
<?php include "templates/include/header.php" ?>
```

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

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

        <ul>

          <li>
            <label for="username">Nome utente</label>
             <input type="text" name="username" id="username"
placeholder="Il tuo nome utente da amministratore" required
autofocus maxlength="20" />
          </li>

          <li>
            <label for="password">Password</label>
                <input type="password" name="password"
id="password"  placeholder="La  tua  password"  required
maxlength="20" />
          </li>

        </ul>

        <div class="buttons">
          <input type="submit" name="login" value="Login" />
        </div>

      </form>

<?php include "templates/include/footer.php" ?>
```

Questa pagina contiene il modulo di login, con i campi principali e un messaggio da mostrare in caso di errore.

Template Lista Articoli

Il secondo template dovrà mostrare la lista di tutti gli articoli presenti, creiamo un file chiamato listArticles.php con il seguente codice:

```
<?php include "templates/include/header.php" ?>

      <div id="adminHeader">
        <h2>Amministrazione Articoli</h2>
             <p>Ti sei loggato come <b><?php echo
htmlspecialchars( $_SESSION['username']) ?></b>. <a
href="admin.php?action=logout"?>Disconnettiti</a></p>
      </div>

      <h1>Tutti gli articoli</h1>

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


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

      <table>
        <tr>
          <th>Data di pubblicazione</th>
          <th>Articolo</th>
        </tr>

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

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

```
          <td>
            <?php echo $article->title?>
          </td>
        </tr>

<?php } ?>

      </table>

       <p><?php echo $results['totalRows']?> articoli <?php echo ( $results['totalRows'] != 1 ) ? 's' : '' ?> in totale.</p>

       <p><a href="admin.php?action=newArticle">Aggiungi un nuovo articolo</a></p>

<?php include "templates/include/footer.php" ?>
```

Il codice serve per mostrare la lista di tutti gli articoli con la data di pubblicazione e il titolo, è stato aggiunto un evento JavaScript che permette, dopo aver cliccato sul titolo, di poter modificare l'articolo. Inoltre è presente il conteggio degli articoli scritti e un link per aggiungere un nuovo articolo.

Template Modifica Articolo

Finalmente siamo arrivati all'ultimo template del backend. Creiamo un file PHP chiamato editArticle.php con il seguente codice:

```
<?php include "templates/include/header.php" ?>

      <div id="adminHeader">
        <h2>Amministrazione Articoli</h2>
        <p>Ti sei loggato come <b><?php echo htmlspecialchars( $_SESSION['username'])           ?></b>.           <a href="admin.php?action=logout"?>Disconnettiti</a></p>
      </div>

      <h1><?php echo $results['pageTitle']?></h1>
```

```
          <form action="admin.php?action=<?php echo
$results['formAction']?>" method="post">
         <input type="hidden" name="articleId" value="<?php
echo $results['article']->id ?>"/>

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

        <ul>

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

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

          <li>
            <label for="content">Contenuto articolo</label>
                <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>
          </li>

          <li>
                 <label for="publicationDate">Data di
pubblicazione</label>
              <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 ) : "" ?>" />
          </li>


        </ul>

        <div class="buttons">
          <input type="submit" name="saveChanges" value="Salva modifiche" />
           <input type="submit" formnovalidate name="cancel" value="Cancella" />
        </div>

      </form>

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

<?php include "templates/include/footer.php" ?>
```

Questo modulo (form) è stato creato sia per creare nuovi articoli sia per modificare articoli già esistenti: **admin.php?action=newArticle** oppure **admin.php?action=editArticle** dipende dal valore che passiamo nella variabile $results[‘formAction’].

È presente, inoltre, un campo nascosto, articleId, per tracciare l’ID dell’articolo che si vuole modificare.

Diamo una veste grafica al nostro CMS

Ora non ci resta che creare un foglio di stile CSS per impostare gli stili che vogliamo.

```
/* Stile per il body e il container */

body {
  margin: 0;
  color: #333;
  background-color: #00a0b0;
  line-height: 1.5em;
}

#container {
  width: 960px;
  background: #fff;
  margin: 20px auto;
  padding: 20px;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  border-radius: 5px;
}


/* Logo e footer */

#logo {
  display: block;
  width: 300px;
  padding: 0 660px 20px 0;
  border: none;
  border-bottom: 1px solid #00a0b0;
  margin-bottom: 40px;
}

#footer {
  border-top: 1px solid #00a0b0;
  margin-top: 40px;
  padding: 20px 0 0 0;
  font-size: .8em;
}


/* Headings */
```

```
h1 {
  color: #eb6841;
  margin-bottom: 30px;
  line-height: 1.2em;
}

h2, h2 a {
  color: #edc951;
}

h2 a {
  text-decoration: none;
}


/* Titolo articolo */

#headlines {
  list-style: none;
  padding-left: 0;
  width: 75%;
}

#headlines li {
  margin-bottom: 2em;
}

.pubDate {
  font-size: .8em;
  color: #eb6841;
  text-transform: uppercase;
}

#headlines .pubDate {
  display: inline-block;
  width: 100px;
  font-size: .5em;
  vertical-align: middle;
}

#headlines.archive .pubDate {
```

```
  width: 130px;
}

.summary {
  padding-left: 100px;
}

#headlines.archive .summary {
  padding-left: 130px;
}


/* Header delle pagine di amministrazione */

#adminHeader {
  width: 940px;
  padding: 0 10px;
  border-bottom: 1px solid #00a0b0;
  margin: -30px 0 40px 0;
  font-size: 0.8em;
}


/* Stile per il form */

form {
  margin: 20px auto;
  padding: 40px 20px;
  overflow: auto;
  background: #fff4cf;
  border: 1px solid #666;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  border-radius: 5px;
  -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
  -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
  box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
}


/* Stile per il form  */
```

```
form ul {
  list-style: none;
  margin: 0;
  padding: 0;
}

form ul li {
  margin: .9em 0 0 0;
  padding: 0;
}

form * {
  line-height: 1em;
}


/* Labels */

label {
  display: block;
  float: left;
  clear: left;
  text-align: right;
  width: 15%;
  padding: .4em 0 0 0;
  margin: .15em .5em 0 0;
}


/* The fields */

input, select, textarea {
  display: block;
  margin: 0;
  padding: .4em;
  width: 80%;
}

input, textarea, .date {
  border: 2px solid #666;
  -moz-border-radius: 5px;
```

```
  -webkit-border-radius: 5px;
  border-radius: 5px;
  background: #fff;
}

input {
  font-size: .9em;
}

select {
  padding: 0;
  margin-bottom: 2.5em;
  position: relative;
  top: .7em;
}

textarea {
  font-size: .9em;
  height: 5em;
  line-height: 1.5em;
}

textarea#content {
}

/* Place a border around focused fields */

form *:focus {
  border: 2px solid #7c412b;
  outline: none;
}



input:valid, textarea:valid {
  background: #efe;
}


/* Submit buttons */
```

```
.buttons {
  text-align: center;
  margin: 40px 0 0 0;
}

input[type="submit"] {
  display: inline;
  margin: 0 20px;
  width: 12em;
  padding: 10px;
  border: 2px solid #7c412b;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  border-radius: 5px;
  -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
  -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
  box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
  color: #fff;
  background: #ef7d50;
  font-weight: bold;
  -webkit-appearance: none;
}

input[type="submit"]:hover, input[type="submit"]:active {
  cursor: pointer;
  background: #fff;
  color: #ef7d50;
}

input[type="submit"]:active {
  background: #eee;
  -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8) inset;
  -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8) inset;
  box-shadow: 0 0 .5em rgba(0, 0, 0, .8) inset;
}


/* Tabelle */

table {
  width: 100%;
```

```
  border-collapse: collapse;
}

tr, th, td {
  padding: 10px;
  margin: 0;
  text-align: left;
}

table, th {
  border: 1px solid #00a0b0;
}

th {
  border-left: none;
  border-right: none;
  background: #ef7d50;
  color: #fff;
  cursor: default;
}

tr:nth-child(odd) {
  background: #fff4cf;
}

tr:nth-child(even) {
  background: #fff;
}

tr:hover {
  background: #ddd;
  cursor: pointer;
}


/* Box per gli status ed errori */

.statusMessage, .errorMessage {
  font-size: .8em;
  padding: .5em;
  margin: 2em 0;
```

```
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  border-radius: 5px;
  -moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
  -webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
  -box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
}

.statusMessage {
  background-color: #2b2;
  border: 1px solid #080;
  color: #fff;
}

.errorMessage {
  background-color: #f22;
  border: 1px solid #800;
  color: #fff;
}
```

Finalmente il nostro mini-CMS è pronto!

Conclusioni

Se sei arrivato fin qua è un grande traguardo, complimenti, ovviamente come puoi vedere si tratta di un CMS molto elementare ma è un'ottima base di partenza per aggiungere funzionalità di ogni tipo. Con la programmazione ad oggetti abbiamo visto come si possono manipolare i dati e, ad esempio, aggiungere una funzionalità è relativamente veloce. La maggior parte dei siti web vetrina hanno bisogno di funzionalità molto basilari che, anche partendo da questo mini-cms, con l'aggiunta di un paio di caratteristiche, sono pronti per essere pubblicati.

Due funzionalità che, sicuramente, dovranno essere aggiunte sono:

1. riscrittura delle URL
2. possibilità di aggiungere i principali metadati SEO

(page title e meta description)

Quando abbiamo chiare le logiche dei CMS e della programmazione ad oggetti, creare un CMS di base per siti web “vetrina” è un lavoro complesso (lungo comprensivo di molti interventi) ma non complicato (difficile). Molto spesso un CMS come WordPress o Joomla per creare siti web piccoli, è una soluzione troppo avanzata che richiede effort nella manutenzione troppo elevato per l’obiettivo del cliente.

Un lavoro interessante è quello di partire da un template statico in HTML e convertirlo in questo mini-cms. Pensiamoci!