Tutorial: Let’s Build a Lightweight Blog System (part 1)

Martin Angelov March 14th, 2013

jQuery Trickshots is our new epic jQuery tips and tricks book. Check it out! [1]
In this tutorial, we will be making a lightweight blog system with PHP and CSS3. The blog posts will be stored in a folder as files (in the markdown format). The system will not use a database and will not have an admin panel, which makes it perfect for small blogs with only one author. It will have an RSS feed and a JSON read-only API, and will feature a completely responsive design that adapts to the screen size of the device.

In this first part, we are going to write the PHP backend, and in part 2 [2], we will code the HTML and CSS. Let’s begin!

The posts

Posts will be named in the following pattern: **2013-03-14_url-of-the-post.md** and they will be stored in the **posts/** folder. The first part is the date of the post (you can optionally include a time as well if you have more than one post for that day), followed by an underscore, and the title of the post. The underscore is used by the system to separate the title from the date. This convention also has the benefit that posts can easily be sorted by publication date. When displayed on your blog, this post will have an URL like **http://example.com/2013/03/url-of-the-post**

Each post will be in the markdown format (hence the extension). The only requirement is that the first line of the markdown document is formatted as a H1 heading. For example:

```
# The title of the blog post

Body of the blog post.
```

This will make the first line a H1 heading. If this is the first time you’ve seen markdown, the next section is for you.

What is markdown

Markdown [3] is a lightweight format for writing text. It has a minimal and intuitive syntax, which is transformed into HTML. This makes it perfect for our simple blog system, as it won't have an admin panel nor database – all the posts will be stored in a folder as files.

We can't simply display the raw markdown directly, first we have to transform it into HTML. There are a number of libraries that can do this (in every major language), but for this tutorial I chose the PHP Markdown library [4], as it integrates nicely with composer.

Composer

Composer [5] is a dependency manager for PHP. With it, you can declare what libraries your project depends on, and they will be downloaded automatically for you. We will use it to include two libraries in this project:

- PHP Markdown [6], which I mentioned above;
- RSS Writer [7], which will output the RSS feed for the blog.

You could simply download these libraries and use them straight away, but let's see how we can do the same with composer (note that you need to follow these steps only if you are starting from scratch; the zip accompanying the tutorial already includes all the necessary files).

First we have to declare which packages the current project depends on, in a file named **composer.json**:

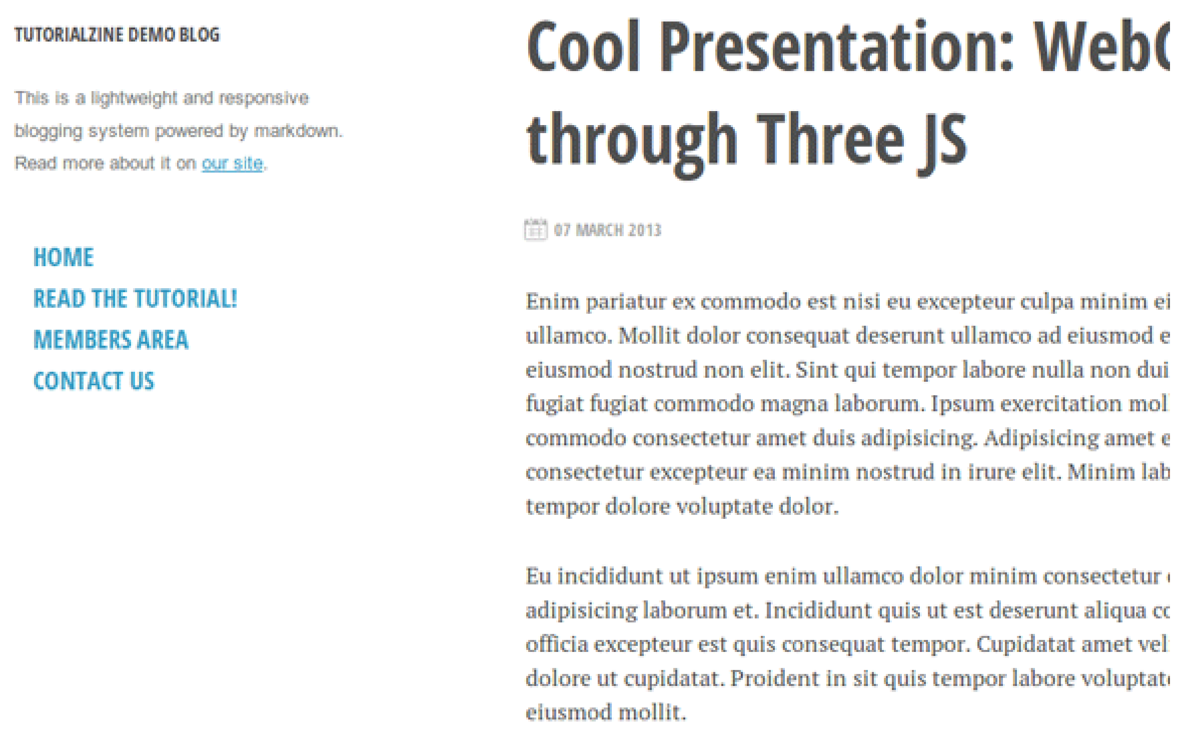
```
{
  "require": {
  "dflydev/markdown": "v1.0.2",
  "suin/php-rss-writer": ">=1.0"
  }
}
```

You can obtain the specific identifiers for the libraries and their versions from the composer package repository [8], or by following the instructions that each library included on its github repo.

The next steps are to install composer into your project (follow these instructions [9]), and to run the install command. This will download all the appropriate libraries and place them in the **vendor/** folder. All that is left is to include the composer autoloader in your php scripts:

```
require 'vendor/autoload.php';
```

This will let you create instances of the libraries without having to include their PHP files individually.



TUTORIALZINE DEMO BLOG

This is a lightweight and responsive blogging system powered by markdown. Read more about it on our site.

HOME
READ THE TUTORIAL!
MEMBERS AREA
CONTACT US

Cool Presentation: Web
through Three JS

07 MARCH 2013

Enim pariatur ex commodo est nisi eu excepteur culpa minim ei
ullamco. Mollit dolor consequat deserunt ullamco ad eiusmod e
eiusmod nostrud non elit. Sint qui tempor labore nulla non dui
fugiat fugiat commodo magna laborum. Ipsum exercitation mol
commodo consectetur amet duis adipisicing. Adipisicing amet e
consectetur excepteur ea minim nostrud in irure elit. Minim lab
tempor dolore voluptate dolor.

Eu incididunt ut ipsum enim ullamco dolor minim consectetur
adipisicing laborum et. Incididunt quis ut est deserunt aliqua c
officia excepteur est quis consequat tempor. Cupidatat amet vel
dolore ut cupidatat. Proident in sit quis tempor labore voluptat
eiusmod mollit.

[10]

Lightweight PHP Blog System

The configuration file

I am using one additional PHP library - Dispatch [11]. This is a tiny routing framework that I’ve mentioned before [12]. It will let us listen for requests on specific URLs and render views. Unfortunately it doesn’t have Composer support at the moment, so we have to download it separately and include it into the project.

A neat feature of this framework is that it lets you write configuration settings in an

INI file, which you can access with the **config()** function call. There are a number of settings you need to fill in for your blog to work:

app/config.ini

```
; The URL of your blog
site.url = http://demo.tutorialzine.com/2013/03/simple-php-blogging-system/

; Blog info

blog.title = "Tutorialzine Demo Blog"
blog.description = "This is a lightweight and responsive blogging system.."
blog.authorbio = "Created by ..."

posts.perpage = 5

; Framework config. No need to edit.
views.root = app/views
views.layout = layout
```

These settings are used throughout the views, so when you are setting up a new blog you won’t need to edit anything else.

The PHP code

Here is our main PHP file:

index.php

```
// This is the composer autoloader. Used by
// the markdown parser and RSS feed builder.
require 'vendor/autoload.php';

// Explicitly including the dispatch framework,
// and our functions.php file
require 'app/includes/dispatch.php';
require 'app/includes/functions.php';

// Load the configuration file
config('source', 'app/config.ini');

// The front page of the blog.
// This will match the root url
get('/index', function () {

    $page = from($_GET, 'page');
    $page = $page ? (int)$page : 1;

    $posts = get_posts($page);

    if(empty($posts) || $page < 1){
            // a non-existing page
            not_found();
        }

     render('main',array(
        'page' => $page,
        'posts' => $posts,
        'has_pagination' => has_pagination($page)
    ));
});

// The post page
get('/:year/:month/:name',function($year, $month, $name){

    $post = find_post($year, $month, $name);

    if(!$post){
        not_found();
    }

    render('post', array(
        'title' => $post->title .' ⋅ ' . config('blog.title'),
        'p' => $post
    ));
});

// The JSON API
get('/api/json',function(){

    header('Content-type: application/json');

    // Print the 10 latest posts as JSON
    echo generate_json(get_posts(1, 10));
});

// Show the RSS feed
get('/rss',function(){

    header('Content-Type: application/rss+xml');

    // Show an RSS feed with the 30 latest posts
    echo generate_rss(get_posts(1, 30));
});

// If we get here, it means that
// nothing has been matched above

get('.*',function(){
    not_found();
});

// Serve the blog
dispatch();
```

The **get()** function of Dispatch creates a pattern that is matched against the currently visited URL. If they match, the callback function is executed and no more patterns are matched. The last **get()** call sets up a pattern that matches every URL in which case we show a 404 error.

Some of the functions you see above are not part of the Dispatch framework. They are specific to the blog and are defined in the *functions.php* file:

- **get_post_names()** uses the glob function to find all the posts and to sort them in alphabetical order (as the date is the first part of the file name, this effectively sorts them by their publishing date);
- **get_posts()** takes the list of names returned by get_post_names() and parses the files. It uses the Markdown library to convert them to HTML;
- **find_post()** searches for a post by month, day and name;
- **has_pagination()** is a helper function that is used in the views (we will look at them in the next part);
- **not_found()** is a wrapper around Dispatch's error function but renders a view as the message;
- **generate_rss()** uses the RSS Library we mentioned above, and turns an array of posts into a valid RSS feed;
- **generate_json()** is only a wrapper around json_encode, but I included it for consistency with the generate_rss function.

And here is the source:

app/includes/functions.php

```
use dflydev\markdown\MarkdownParser;
use \Suin\RSSWriter\Feed;
use \Suin\RSSWriter\Channel;
use \Suin\RSSWriter\Item;

/* General Blog Functions */

function get_post_names(){

    static $_cache = array();

    if(empty($_cache)){

        // Get the names of all the
        // posts (newest first):
```

```
        $_cache = array_reverse(glob('posts/*.md'));
    }

    return $_cache;
}

// Return an array of posts.
// Can return a subset of the results
function get_posts($page = 1, $perpage = 0){

    if($perpage == 0){
        $perpage = config('posts.perpage');
    }

    $posts = get_post_names();

    // Extract a specific page with results
    $posts = array_slice($posts, ($page-1) * $perpage, $perpage);

    $tmp = array();

    // Create a new instance of the markdown parser
    $md = new MarkdownParser();

    foreach($posts as $k=>$v){

        $post = new stdClass;

        // Extract the date
        $arr = explode('_', $v);
        $post->date = strtotime(str_replace('posts/','',$arr[0]));

        // The post URL
        $post->url = site_url().date('Y/m', $post->date).'/'.str_replace('.md','',$arr[1]);

        // Get the contents and convert it to HTML
        $content = $md->transformMarkdown(file_get_contents($v));

        // Extract the title and body
        $arr = explode('</h1>', $content);
        $post->title = str_replace('<h1>','',$arr[0]);
        $post->body = $arr[1];

        $tmp[] = $post;
    }

    return $tmp;
}

// Find post by year, month and name
function find_post($year, $month, $name){

    foreach(get_post_names() as $index => $v){
        if( strpos($v, "$year-$month") !== false && strpos($v, $name.'.md') !== false){

            // Use the get_posts method to return
            // a properly parsed object

            $arr = get_posts($index+1,1);
            return $arr[0];
        }
    }

    return false;
}

// Helper function to determine whether
// to show the pagination buttons
function has_pagination($page = 1){
    $total = count(get_post_names());

    return array(
        'prev'=> $page > 1,
        'next'=> $total > $page*config('posts.perpage')
    );
}

// The not found error
function not_found(){
    error(404, render('404', null, false));
}

// Turn an array of posts into an RSS feed
function generate_rss($posts){

    $feed = new Feed();
    $channel = new Channel();

    $channel
        ->title(config('blog.title'))
        ->description(config('blog.description'))
        ->url(site_url())
        ->appendTo($feed);

    foreach($posts as $p){

        $item = new Item();
        $item
            ->title($p->title)
            ->description($p->body)
            ->url($p->url)
            ->appendTo($channel);
    }

    echo $feed;
}

// Turn an array of posts into a JSON
```

```
function generate_json($posts){
    return json_encode($posts);
}
```

At the top of the file, we have a number of **use** statements, which import the necessary namespaces (read more about PHP’s namespaces [13]).

The last thing we need to do, is to rewrite all requests so they hit index.php. Otherwise our fancy routes wouldn’t work. We can do this with an .htaccess file:

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

RewriteCond $1 !^(index\.php)
RewriteRule ^(.*)$ index.php/$1 [L]
```

If a file or folder does not exist, the request will be redirected to index.php.

It’s a wrap!

Continue with the second part [14], where we are creating the views and the responsive CSS design.

by Martin Angelov

Martin is a web developer with an eye for design from Bulgaria. He founded Tutorialzine in 2009 and publishes new tutorials weekly.

Tutorials [15]

- PHP [16]

1. http://tutorialzine.com/books/jquery-trickshots/
2. http://tutorialzine.com/2013/03/simple-php-blogging-system-part-2/
3. http://en.wikipedia.org/wiki/Markdown
4. https://github.com/dflydev/dflydev-markdown
5. http://getcomposer.org/doc/00-intro.md
6. https://github.com/dflydev/dflydev-markdown
7. https://github.com/suin/php-rss-writer
8. https://packagist.org/
9. https://packagist.org/
10. http://demo.tutorialzine.com/2013/03/simple-php-blogging-system/
11. http://noodlehaus.github.com/dispatch/
12. http://tutorialzine.com/2013/02/24-cool-php-libraries-you-should-know-about/
13. http://php.net/manual/en/language.namespaces.importing.php
14. http://tutorialzine.com/2013/03/simple-php-blogging-system-part-2/
15. http://tutorialzine.com/category/tutorials/
16. http://tutorialzine.com/tag/php/