Decoder Ring: The CMS is Flat

Jerry Spiller

Art Institute of Charleston, yeri.spiller@gmail.com
Decoder Ring — The CMS is Flat

Web folks, do you ever get a nagging feeling that WordPress or another database-driven CMS might not be the best solution for a particular site you’re building?

When I began to create the site for my Webcomic Boggy Down I knew its architecture could be pretty simple and streamlined. Aside from a few one-off static pages like an “About” page, almost all pages would need a recurring blog-like structure that simply housed an image for each page, with enough markup to keep them organized and accessible.

Considering the needs of the project, I found myself asking if the architecture for the underlying content really required the overhead of a database-driven CMS. Did the site need text and media content housed in a SQL database? Or different sets of admins with different levels of privileges? Did most of those admins need WYSIWYG editors? I asked myself those questions about Boggy Down and the answer to each one was “no.”

I had to question if I was looking at WordPress out of habit, despite the fact that another solution might be more appropriate. Simple can be freeing. Why house content in a database if I don’t need to? Why have various users in a database if it’s only me, and server access is all I need?

I know a lot of librarians working on Web projects face similar concerns, so I’ll share the results of my search to find a more streamlined solution for a project that didn’t necessarily benefit from MySQL in its stack. Not that WordPress hasn’t given us a lot and isn’t a great fit for many projects. Still, I’m sure a lot of folks working on presenting a body of images or other media content for digital libraries or special collections, or even reference and instruction, have looked at their back libraries or special collections, or even referred to .md files stored in a folder called content. Each content file typically starts with metadata, with available fields including title, date, author, time, and description (Pico). Like our other choices, HTML structure, and CSS styling are completely up to you.

Boggy Down

Considering the needs of the project, I found myself asking if the architecture for the underlying content really required the overhead of a database-driven CMS. Did the site need text and media content housed in a SQL database? Or different sets of admins with different levels of privileges? Did most of those admins need WYSIWYG editors? I asked myself those questions about Boggy Down and the answer to each one was “no.”

I had to question if I was looking at WordPress out of habit, despite the fact that another solution might be more appropriate. Simple can be freeing. Why house content in a database if I don’t need to? Why have various users in a database if it’s only me, and server access is all I need?

I know a lot of librarians working on Web projects face similar concerns, so I’ll share the results of my search to find a more streamlined solution for a project that didn’t necessarily benefit from MySQL in its stack. Not that WordPress hasn’t given us a lot and isn’t a great fit for many projects. Still, I’m sure a lot of folks working on presenting a body of images or other media content for digital libraries or special collections, or even reference and instruction, have looked at their back libraries or special collections, or even referred to .md files stored in a folder called content. Each content file typically starts with metadata, with available fields including title, date, author, time, and description (Pico). Like our other choices, HTML structure, and CSS styling are completely up to you.

Jekyll

Jekyll is built on Ruby and is described by its developer Tom Preston-Werner as a “blog aware static site generator” (2013). It is released under the free and open source MIT license. Jekyll powers GitHub Pages, so it can handle large sites with multiple lower-level admins providing loads of content. In addition to Markdown and HTML, it also supports the lightweight markup language Textile as well as Ruby’s Liquid templating. It can be extended to work as a blog with a little custom work, or use of its companion Octopress. Octopress provides a blog-ready HTML template for Jekyll and uses Git cloning to support multiple sites (Octopress 2011).

The main caveat I see for Jekyll is that if you’re not accustomed to regularly writing Ruby and maintaining your Ruby environment (I do not and was not) it probably requires too much configuration and too many updates for me, so I had a hard time getting it to work. If your Ruby is strong, I could see this being a great fit for large digital collections.

Pico

Pico is a PHP-based CMS by Gilbert Pellegrom from Dev7studios. It is also released under the MIT license. Pico is built to employ Markdown exclusively, though Markdown files can contain HTML. Pico offers a truly flat directory structure, with .md files stored in a folder called content. Each content file typically starts with metadata, with available fields including title, date, author, time, and description (Pico). Like our other choices, HTML structure, and CSS styling are completely up to you.

Kirby

Kirby is a CMS created by designer and developer Bastian Allgeier. Like Pico, it is PHP-based. After stumbling across Kirby and experimenting with it a bit, it seemed to match up well with Boggy Down’s needs, being light and adaptable without a lot of overhead.

Coding for Kirby should seem instantly familiar to anyone used to writing PHP. Content files use Markdown, in a framework written by Allgeier called Kirbystext. Content files are .txt by default but could be changed to .md or another format in the config file. Kirby wraps the content up in custom, array-like objects based on its directory structure. When writing your PHP and HTML, these can be accessed through its jQuery-inspired API, manifesting a philosophy which might be described as “look for a thing, then use or modify the thing.” See Figure I below.

Kirby’s sensible folder and object hierarchies, along with organized and easy to follow documentation from Allgeier, really set it over the top for me. I found creating and referencing parent-child (and sibling) structure in Kirby more intuitive than in Pico. Allgeier even provides an extremely useful reference cheat sheet (Kirby 2013a). Documentation also includes growing list of tutorials for extending Kirby, including writing blog templates, configuring your RSS feed, hiding your backend content files, and creating “art directed” posts with their own unique styling (Kirby 2013b). I found the tutorials easy to follow and implement. Boggy Down’s main content area, the comics, is just a blog made with the help of Allgeier’s blog tutorial, with the main blog page customized to return only the single most recent post as a page.

Kirby’s default metadata might seem less robust than Pico’s, but it is easily extensible. You can add any field you like. Allgeier offers, “If you need more fields for your template—maybe you want to show a date on that page or a subtitle—simply add more fields to the text file. [...] After each field you have to add four methods with corresponding names: $page->yourvar()” (Kirby 2013c).

From the PHP, Kirby makes these new fields, which you can think of as custom variables, available from methods with corresponding names: $page->yourvar()
In the case of the subtitle example above, that would be:
```ruby
$page->subtitle()
```

The worst thing I could say about Kirby is that it prescribes a directory structure and naming scheme to a certain extent. Visible content should be in a subdirectory of Kirby’s content folder, named according to number-name scheme such as “01-blog” or “02-comics.” Unnumbered folders are “invisible,” meaning they are still Web-accessible but do not appear in auto-generated navigation. Subdirectories follow this same scheme, with the numbers starting over from 01 in each folder.

Again, I found this hierarchy sensible and didn’t consider it limiting to the project at hand. Admittedly, it does mean that media content like my image files are spread out in each individual post subdirectory. If you need to house all media together in one directory or in another structure, as I could see being useful for a digital library admin who didn’t want to reinvent a wheel, you could get around this limitation with Kirby’s custom metadata fields.

After considering your projects needs, I recommend experimenting with each of these flat file systems to see if it’s right for the project and the people working on it. All are available on GitHub (Kirby’s licensing is based on the honor system). Jekyll is very capable and probably fairly simple to use and maintain if you develop in Ruby. If you’re more comfortable in PHP, Pico and Kirby are very similar on first and second glance. I found Kirby easier to work with and extend. Plus, if I had problems Kirby’s documentation cleared things up faster than Pico’s did.

And then there’s the fact that I was making a Webcomic, and what comics creator could resist a name that recalls the King, Jack Kirby?

References


