Rationale

Why another wiki engine?

Marco Pessotto

Marco Pessotto
Rationale
Why another wiki engine?

amusewiki.org

and the possibility to have the archive distributed and decentralized was just too good to be ignored.

To sum-up: the markup is decoupled by everything and can work standalone on a single file or a whole tree. The archive is decoupled by everything else and is just a git archive which follows some simple directory-naming conventions.

The web application is then in charge to extract the relevant information from the texts, store it into the database, keep them up-to-date, and use it (hating databases doesn't mean I don't recognize their usefulness when properly used – it's just using them as text archives that it's so bad).

DBIx::Class was handy to provide another layer of abstraction between the web application itself and the archive, so the code could be shared among the Catalyst part, and the back-end part (which compiles the texts, updates the database and the git archive).

For the layout, which I admit was kind of hacked together by a programmer (myself), not a designer, I went straight for bootstrap and jquery and some other useful javascript libraries (many are used around). It's not the best out there, but I feel that definitively I could have done much worse.

appears to work well and is extensible enough to accomodate any future need (being Perl code, and not an hack).

I truly hate to read on screen. For the same reason, an EPUB output would have been nice as well, and given that the HTML was already there, a couple of CPAN modules were put together (notably Template Toolkit and EBook::EPUB), and a module to wrap the pieces together on the command line was created: Text::Amuse::Compile. This code provides also a command line utility to generate the formats on the command line.

Last but not the least, experience showed that people are used to type the character '' and have it rendered as `` depending on the position and the language used. Same goes for the dash and other typographical elements. Also, there was the need for some code capable (to some extend) to take some HTML code and convert it into the markup. For this I wrote the code that later became Text::Amuse::Preprocessor.

The modules above provide a way to work locally on the texts without needing any access to the internet (they install the command line scripts as well).

The web front-end

Sometimes writing Amusewiki itself felt like reinventing the wheel for the n^{th} time, so, while for the core modules I kept the dependencies at the minimum, for the web pieces I used many available modules on CPAN everywhere was possible.

The application itself is built on DBIx::Class, Catalyst and Template Toolkit. The almost "standard" tools for web things written in Perl out there. I considered Dancer as well, but given that the previous incarnation of amusewiki used Dancer (and I had the feeling it was already becoming too messy), I chose to jump on the catalyst train, and I don't have any regret because it's really *elegant*.

A note about the database. I'm not a database fan. Actually, I don't like them at all for managing texts. I think that the various db-driven CMS out there are just doing the wrong thing, because they're coupling the texts with the application.

For Amusewiki, being that it's archive- and library-oriented, I wanted the texts stored in a directory tree under revision control. Given that Git now is ubiquitous, choosing it was the easy part. Also, the revision history

Contents

Markup and formats

Apparently, the world is full of wiki engines, some of them good or very good. So the question is if this particular engine was really needed.

The problem AmuseWiki wants to address is, anyway, very practical. I needed and wanted a decent range of output format. Not just a PDF output (MediaWiki does that as well, for example), but a nice, good, readable PDF. Now, given that the procedure needs to be completely automated, the perfection is hardly reached, but we can do better than simply render an HTML page and stuff the output in a PDF container. TeX has been around more than 30 years by now. So the idea was of course to use that.

Then I needed a markup, possibly an existing one. Markdown was considered, of course, but then discarded, because the original specification didn't support footnotes, explicitly permitted random inline HTML, and some other questionable (in my very humble opinion) design choices. Creating another markdown dialect would have bring me to square zero.

Given that I'm a Emacs user, I encountered Emacs Muse some years ago, and I truly liked the syntax. By now the project is more or less stalled, but the elisp code still works, is distributed in Debian, has a nice manual, and provided a first reference implementation for the output.

The other alternative would have been the org-mode markup, but, beside to be very large and complicated by lots of plugins that people would expect to work, it have some (again) questionable markup elements for things often used as quotations.

All considered, the muse markup was small, compact and expressive enough for my needs. The bottom line is: every lightweight markup needs something like a couple of minutes to be learned, so better choose one I like, not the one everyone uses, and have a manual for that.

In turn, this could have been another questionable design choice, but that's the upside of being the author of some software: you write it your way.

The markup implementation was developed in Perl and is available on CPAN as Text::Amuse. I added the "A" prefix because the markup is not 1:1 compatible with the original one.

Once the markup was able to produce HTML and TeX code, I needed something to create the imposed versions of the PDF, i.e., a PDF file which can be printed, folded and clipped to create booklets. During the past years, I tried various different solutions, which worked (to some extent), but weren't really satisfying. This resulted in PDF::Imposition, which so far