Impressions of the SSG sprinkles

My impression with trying to use sprinkles, which I hoped could satisfy all my needs.

August 4, 2023

Contents

Reasons to try it

Reasons to try it

I spend some time with trying Sprinkles as a foundation for my own homepage. It was attractive because it seemed to fulfill most of my requirements:

- it is written in Haskell,
- uses `Ginger`, the Haskell implementation for the `Jinja2` theme languages,
- it builds a static site which can be uploaded to a host of my choice,
- is design and setup with `YAML` files,
- clean design using a compiled, site independent engine; all site design is in files,
- support for `markdown` (using `pandoc`).

Some points I perceived as rather negative:

- small (very small) developer and user base[1],
- static site generation is possible, dynamic sites goal,
- issues with caching between invocations (makes development difficult).

[1] Is this better than *roll you own*?

Observations

Observations

During my tests, I found some things which could be improved and which became also ideas which were later included in my SSG design.

Button to get clean start

Provide a method to force a clean restart and, if possible, force it automatically when certain files change. For example, when `project.yml` changes, do a clean restart without user intervention.

Breaking the program in independent services

The monolithic nature with a very large number of dependencies makes it hard to maintain and probably difficult to attract others to contribute.

For example:

- factor out, e.g. string conversion, error handling, file IO[2]
- avoid import of qualified `Prelude`[3]

[2] my *uniform* approach

[3] Despite the well known issues with the current Prelude; I prefer to use it unchanged and work arount the issues, mostly to make code integratable with code written by others.

I think it could be possible to have a separate program for `bake` and one for `serving`, with a common sprinkles-core.

With `ginger` I had the impression that part of the complexity comes from using an extended version of JSON.Value.[4]

[4] I understand the reasons for adding specific types but feel it would be better to wrap the type for `Http` around a JSON.Value and not produce a separate value which then needs conversions etc. etc.

Many small files connected by same parameter names

The structure of the different YAML files to define the design, together with the templates (in `Jinja2`), the `css` styles and others are hard to grasp and keep aligned.

Theme folder

I liked the `Pelikan` approach to have a `theme` folder, which separates the theme from the remainder.

Preview not correctly rendered

The preview operation seems not to properly treat markdown.[5]

[5] The three tick style, which should give a typewriter font, gives spaced text.

Parameter for pages and posts

I would prefer a structure for the pages as two `yaml` docs, separated by `----`, where the first contains proper `yaml` with key - value pairs and the second the markdown text.[6]

[6] Pandoc has adopted this approach and I use it for SSG.

Archetypes for pages

I liked the idea to define *archetypes* for pages and a function, which defines the values and an `archetype` to the proper place and fill the values correctly.

Terminology

I think

- `theme` for the overall layout arrangements (including styles) is a nice word, and I prefer it over blueprints.
- `byline` seems to be an established term for the subtitle of a newspaper and in analogy to a blog.

Conclusion

Conclusion

I was impressed with `sprinkles` enough to try to combine it with `sitepipe` and later `slick` which then moved to use `shake`. I eventually decided to go directly to build on top of `pandoc` and `shake`[7].

[7] all packages can be found on hackage.

Produced with 'daino' (Version versionBranch = [0,1,5,3,3], versionTags = []) from /home/frank/Desktop/myHomepage/Essays/SSGdesign/06rationale/009aboutSprinkles.md with latexTufte81.dtpl
arguments